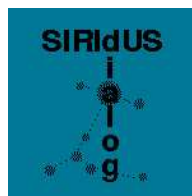

Flexible Dialogue

Staffan Larsson, Gabriel Amores, Elena Karagjosova,
David Milward, Dimitra Tsovalzi

Distribution: PUBLIC



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

Deliverable D1.4

October, 2002

IST-1999-10516 SIRIDUS

Specification, Interaction and Reconfiguration in Dialogue Understanding Systems

Göteborg University

Department of Linguistics

Telefónica Investigación y Desarrollo SA Unipersonal

Speech Technology Division

Universität des Saarlandes

Department of Computational Linguistics

Universidad de Sevilla

Departamento de Lengua Inglesa

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

The SIRIDUS Project Administrator
Universität des Saarlandes
Fachrichtung 5.7
Allgemeine Linguistik Gebäude 17.2
D-66041
Saarbrücken
Germany

Copies of reports and other material can also be accessed from the project's homepage
<http://www.ling.gu.se/projekt/siridus>

©2002, 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.

Responsibility for the authorship is divided as follows. Staffan Larsson was the overall editor and wrote Chapters 2, 3 and 4. Chapter 5 was written by Elena Karagjosova with assistance from Kepa Rodriguez and Ivana Kruijff-Korbayova. Chapter 6 was written by Dimitra Tsovalzi. Chapter 7 was written by Gabriel Amores. Chapter 8 was written by David Milward. Chapters 1 and 9 were written jointly by the authors.

Contents

1	Introduction	15
1.1	Grounding	16
1.2	Addressing unraised issues	19
1.3	Flexible menu-based dialogue	20
1.4	Negotiative dialogue	21
1.5	Conditional responses	21
1.6	Tutorial dialogue	23
1.7	Conflicts	24
1.8	Over informative answers and clarification questions	24
I	Flexible Issue-based Dialogue Management	27
2	Grounding issues	29
2.1	Introduction	29
2.1.1	Dialogue examples	30
2.2	Background	32

2.2.1	Clark: Adding to the common ground	32
2.2.2	Ginzburg: QUD-based utterance processing protocols	34
2.2.3	Allwood: Interactive Communication Management	37
2.3	Preliminary discussion	38
2.3.1	Levels of action in dialogue	38
2.3.2	Reaction level feedback	39
2.3.3	Levels of understanding	41
2.3.4	Some comments on Ginzburg's protocol	42
2.4	Feedback and related behaviour in human-human dialogue	44
2.4.1	Classifying explicit feedback	44
2.4.2	Positive, negative, and neutral feedback	45
2.4.3	Eliciting and non-eliciting feedback	46
2.4.4	Form of feedback	46
2.4.5	Meta-level and object-level feedback	47
2.4.6	Fragment feedback / clarification ellipsis	47
2.4.7	Own Communication Management	48
2.4.8	Repair and request for repair	48
2.4.9	Request for feedback	49
2.5	Update strategies for grounding	49
2.5.1	Optimistic and pessimistic strategies	49
2.5.2	Grounding updates and action levels	50
2.5.3	The cautious strategy	51

2.6	Feedback and grounding strategies for GODIS	52
2.6.1	Grounding strategies for dialogue systems	53
2.6.2	“Implicit” and “explicit” verification in dialogue systems	54
2.6.3	Issue-based grounding in GODIS	54
2.6.4	Enhancing the information state to handle feedback	57
2.6.5	Feedback and sequencing dialogue moves	59
2.6.6	Grounding of user utterances in GODIS2	63
2.6.7	Grounding of system utterances in GODIS2	84
2.6.8	Evidence requirements and implicit grounding	92
2.6.9	Sequencing ICM: reraising issues and loading plans	96
2.7	Further implementation issues	99
2.7.1	Update module	99
2.7.2	Selection module	100
2.8	Discussion	102
2.8.1	Some grounding-related phenomena not handled by GODIS2	102
2.8.2	Towards an issue-based account of grounding and action levels	103
2.8.3	Comparison to Traum’s computational theory of grounding	103
2.9	Summary	105
3	Addressing unraised issues	107
3.1	Introduction	107
3.2	Some limitations of GODIS2	108

3.3	The nature(s) of QUD	109
3.3.1	Ginzburg’s definition of QUD	109
3.3.2	Open questions not available for ellipsis resolution	110
3.3.3	Open but not explicitly raised questions	111
3.3.4	Global and local QUD	111
3.3.5	Some other notions of what a QUD might be	113
3.4	Question Accommodation	113
3.4.1	Background: Accommodation	114
3.4.2	Accommodation, interpretation, and tacit moves	114
3.4.3	Extending the notion of accommodation	115
3.5	Formalizing question accommodation	116
3.5.1	Information state in GoDIS3	116
3.6	Varieties of question accommodation and reaccommodation	118
3.6.1	Issue accommodation: from dialogue plan to ISSUES	119
3.6.2	Local question accommodation: from ISSUES to QUD	122
3.6.3	Issue clarification	123
3.6.4	Dependent issue accommodation: from domain resource to ISSUES . . .	125
3.6.5	Dependent issue clarification	130
3.6.6	Question reaccommodation	132
3.6.7	Opening up implicit grounding issues	135
3.7	Further implementation issues	144
3.7.1	Dialogue moves	144

3.7.2	GoDIS3 update module	144
3.7.3	Selection module	149
3.8	Discussion	150
3.8.1	Phrase spotting and syntax in flexible dialogue	150
3.8.2	Relaxing constraints using denial and dependent reaccommodation . . .	151
3.8.3	“Smart” interpretation	152
3.8.4	Separating understanding, acceptance, and integration	153
3.8.5	Accommodation and the speaker’s own utterances	154
3.8.6	Accommodation vs. normal integration	155
3.8.7	Dependent issue accommodation in VoiceXML?	155
3.9	Summary	156
4	Action-oriented and negotiative dialogue	157
4.1	Introduction	157
4.2	Issues and actions in action-oriented dialogue	158
4.3	Extending GoDIS to handle action oriented dialogue	158
4.3.1	Enhancing the information state	158
4.3.2	Dialogue moves	160
4.4	Interacting with menu-based devices	161
4.4.1	Connecting devices to GoDIS	161
4.4.2	From menu to dialogue plan	164
4.4.3	Extending the resolves relation for menu-based AOD	165

4.5	Implementation of the VCR control domain	165
4.6	Update rules and dialogue examples	167
4.6.1	Integrating and rejecting requests	167
4.6.2	Executing device actions	168
4.6.3	Selecting and integrating confirm-moves	169
4.6.4	Dialogue example: menu traversal and multiple threads	170
4.6.5	Action accommodation and clarification	172
4.6.6	Dialogue examples: action accommodation and clarification	174
4.7	Issues under negotiation in negotiative dialogue	174
4.7.1	Sidner's theory of negotiative dialogue	175
4.7.2	Negotiation as discussing alternatives	178
4.7.3	Issues Under Negotiation (IUN)	180
4.7.4	An example	181
4.8	Discussion	183
4.8.1	Negotiation in inquiry-oriented dialogue	183
4.8.2	Rejection, negotiation and downshifting	183
4.8.3	Dialogue structure and issue-based dialogue management	185
4.9	Summary	186
5	Conditional responses	187
5.1	Introduction	187
5.2	The nature of conditional responses	190

5.2.1	Meaning	191
5.2.2	Dialogue behavior	195
5.3	Implementation	201
5.3.1	User questions	203
5.3.2	The search	207
5.3.3	Production of CRs	214
5.3.4	Interpretation of CRs	231
5.4	Summary and future work	232
6	Tutorial Dialogues	235
6.1	Introduction	235
6.2	Characteristics of Tutorial Dialogues	235
6.2.1	Guided Problem Solving	236
6.2.2	Tutoring Methods	238
6.2.3	Hinting	238
6.2.4	Explanations	239
6.2.5	Collaborative Responses in Tutorial Dialogues	240
6.3	Obligations-based Modeling of Tutorial Dialogues	241
6.4	Reconfigurability of TRINDIKIT, GODIS and IMDiS for Tutorial Dialogues . . .	243
6.4.1	Dialogue Moves and Dialogue Context	243
6.4.2	Semantic Representation and Lexicon	245
6.4.3	Planning	246

6.4.4	Collaborative Responses and Planning	251
6.4.5	Mixed Initiative	252
6.4.6	Obligation Modeling in TRINDIKIT	253
6.4.7	Miscellaneous	253
6.5	Conclusion	254

II Issues in Flexible Dialogue 257

7 Flexibility and Cooperative Behaviour in Natural Command Language Dialogues 259

7.1	Introduction	259
7.2	Natural Command Language Dialogues	260
7.2.1	Siridus	260
7.2.2	D'Homme	260
7.3	Comparing NCLDs with other types of dialogue	261
7.3.1	Functional Embedding	261
7.4	Adding flexibility through dialogue commands	262
7.5	Adding flexibility by expanding the linguistic coverage of the system	265
7.6	Cooperative behaviour in NCLDs	265
7.7	Sources of conflict in NCLDs	266
7.8	Advanced Cooperation	268
7.9	DISC Guidelines	270
7.10	Conclusion	271

8	Over informative answers and clarification questions	273
III	Summary and conclusions	277
9	Summary and conclusions	279
9.1	Overall summary and conclusions	279
9.2	Dialogue genres	281
9.2.1	Relation to Dahlbäck's dialogue taxonomy	283
9.2.2	Relation to Allen et. al.'s dialogue classification	284
9.3	Future research areas in flexible issue-based dialogue management	286
9.3.1	Developing the issue-based approach to grounding	286
9.3.2	Other dialogue and activity types	288
9.3.3	Semantics	290
9.3.4	General inference	291
9.3.5	Natural language input and output	291
9.3.6	Applications and evaluation	292

Chapter 1

Introduction

This deliverable presents various strains of work on flexible dialogue management conducted in SIRIDUS. By “flexible dialogue management” we mean, approximately, mechanisms needed for dealing with dialogue phenomena that fall outside the scope of current commercial systems, such as those based on (plain) VoiceXML (McGlashan *et al.* (2001)).

We will be dealing with a host of dialogue phenomena that we regard as requiring flexible dialogue management:

- grounding and feedback
- addressing unraised issues
- action-oriented dialogue, including menu-based dialogue
- negotiative dialogue
- tutorial dialogue
- conditional responses
- clarification subdialogues
- cooperation and collaboration

In the remaining chapters of this deliverable we will investigate these phenomena and provide techniques for dealing with them in dialogue systems. Most of the work presented here also involves implementations of these techniques. The deliverable is divided into two main parts, the

first collecting contributions dealing with flexible issue-based dialogue management in GoDiS, and the second collecting work based on other approaches to dialogue management..

In this introduction, we will briefly try to motivate why handling these phenomena are important for dialogue systems. Further motivations can be found in the respective chapters.

1.1 Grounding

In all dialogue, issues concerning contact, perception, understanding and acceptance of utterances are of central importance. We refer to these as “meta-issues”, or “grounding issues”. We give an account of these issues where the concepts of optimism and pessimism regarding grounding are employed. A partial-coverage model of feedback related to grounding is motivated from the perspective of usefulness in a dialogue system, and implemented. This allows the system to produce and respond to feedback concerning issues dealing with the grounding of utterances.

We will first attempt to give an impression of how grounding is handled in current commercial dialogue systems. We will then show examples of grounding in human-human dialogue, and show an example of grounding behaviour in GoDiS.

In the literature concerning practical dialogue systems (e.g. San-Segundo *et al.*, 2001), grounding is often reduced to verification of the system’s recognition of user utterances. Two common ways of handling verification are described as “explicit” and “implicit” verification, exemplified in (1) (example from San-Segundo *et al.*, 2001).

- (1) a. I understood you want to depart from Madrid. Is that correct? [explicit]
- b. You leave from Madrid. Where are you arriving at? [implicit]

Actually, both “explicit” and “implicit” feedback contain a verbatim repetition or a reformulation of the original utterance, and in this sense they are both explicit. The actual base for the distinction is what we have here referred to as polarity: “explicit” verification is neutral (and eliciting and interrogative) whereas “implicit” verification is positive.

In human-human dialogue, explicit confirmations occur in noisy environments and in situations where understanding is critical (e.g. when arranging a meeting in a busy airport). Given that verification is presumably a rather marginal phenomena in human-human dialogue, it is perhaps surprising that it is often the only aspect of feedback covered in dialogue systems literature. Firstly, because it is usually not necessary for humans to verify what they (think they) have heard;

that is, it is a rather uncommon grounding procedure in human-human dialogue. Second, because it only involves part of the full spectrum of feedback behaviour, excluding e.g. acceptance-related feedback behaviour.

Of course, verification of user utterances are of central importance in dialogue systems, given the quality of current speaker-independent speech recognition. This explains to some extent why verification is often the only aspect of feedback handled by current systems - it is simply necessary. However, this is no reason not to explore further the possible uses of a wider range of feedback behaviour in dialogue systems.

The human-human dialogue excerpt¹ in (2) shows two common kinds of feedback. J's "mm" shows that J (thinks that he) understood P's previous utterance; P's "pardon" shows that P was not able to hear J's previous utterance. The example also includes a hesitation sound ("um") from J. (P is a customer and J a travel agent.)

- (2) P : öm (.) flyg ti paris
 um (.) flight to paris
 J : mm (.) ska du ha en returbiljett
 mm (.) do you want a return ticket
 P : va sa du
 pardon
 J : ska du ha en tur å retur
 do you want a round trip

The feedback in (2) consisted of conventionalized feedback words ("mm", "pardon"). However, feedback may also be more explicit and repeat the central content of the previous utterance, as K's second feedback utterance in (3).

- (3) B : ja ska va framme i [1 göteborg]1 e e ungefär vi nietiden
 om de finns nå tidit [2 morgonflyg]2
 I need to be in Gothenburg er er around nine if there is an
 early morning flight
 K : [1 m]1
 m
 K : [2 vi]2 nietiden m vi ska se
 Around nine m let's see

The function of an utterance answering a question is not primarily to give feedback, but rather to provide task-related information. However, an answer also shows that the previous question was

¹Except where noted, the human-human dialogues in this chapter have been collected by the University of Lund as part of the SDS project. We quote transcriptions done in Göteborg as part of the same project. The (.) symbol indicates a pause.

understood and integrated. Example (4) shows that feedback is sometimes given in reaction to a question before the question is answered.

- (4) J : sen måste du ha e sån där intenationellt studentkort också
ha du de
then you need one of those international student cards do you have that
P : mm nä
mm no

However, often human-human communication proceeds without explicit feedback, as in the excerpt from a telephone dialogue in (5).

- (5) K : vart ska du åka
where are you going
B : e jag ska åka till göteborg
er i'm going to gothenburg
K : vilket datum
what date
B : den sjätte juni
the sixth of june
K : vill du åka me flyg eller me tåg
do you want to fly or take the train

By contrast, in current commercial systems it is common to check the system's interpretation of recognized user input asking a *y/n*-question, as seen in the (invented) dialogue in (6). This behaviour is much less common in human-human dialogue, but may occur in noisy environments or when communicating over an unreliable communication channel (e.g. when using a mobile phone on a train).

- (6) S : What day do you wish to travel?
U : Friday
S : Friday the 5th of April, is that correct?

A similar but not identical behaviour does occur in human-human dialogue, as in (7), taken from the London-Lund corpus and used by Clark and Schaefer (1989a). In this example, B gives an explicit report of how A's utterance was interpreted, whereupon A confirms that the interpretation is correct by saying "yes".

- (7) A : F . six two
B : F six two
A : yes
B : thanks very much

The dialogue in (DIALOGUE 1.1) gives an impression of what feedback capabilities we aim to add to GoDIS in this chapter.

(DIALOGUE 1.1)

S> Welcome to the travel agency!

S> hello

U> price information please

S> You asked about price. How do you want to travel?

U> pardon

S> How do you want to travel?

U> by plane

S> By plane, is that correct?

U> yes

S> Okay. What city do you want to go to?

U> lododo

S> Pardon?

U> london

S> Okay, to london. What city do you want to go from?

U> lyons

S> Sorry, Lyons is not in the database. So, What city do you want to go

from?

U> paris

...

U> What about connecting flights?

S> You asked about connecting flights. Sorry, I cannot answer that.

1.2 Addressing unraised issues

In real dialogue, one often sees utterances which can be construed as addressing issues which have not been explicitly raised in the dialogue. To enable more flexible dialogue behaviour, we make a distinction between a local and a global QUD (referring to the latter as “open issues”, or just “issues”). The notions of question and issue accommodation are then introduced to allow the system to be more flexible in the way utterances are interpreted relative to the dialogue context.

Question accommodation allows the system to understand answers addressing issues which have not yet been raised. In cases of ambiguity, where an answer matches several possible questions, clarification dialogues may be needed.

One of the action levels to which grounding applies is that of pragmatic understanding, i.e. making sense of the meaning of an utterance in the current dialogue context. Some basic mechanisms for grounding on the understanding level have been implemented in the version of GoDiS described in Chapter 2. However, the kinds of dialogues handled by this system are still rather rigid and system-controlled.

We introduce the notion of question accommodation to allow the system to be more flexible in the way utterances are interpreted relative to the dialogue context. Among other things, question accommodation allows the system to understand answers to questions which have not yet been asked, and to understand such answers even before any issue has been explicitly raised. In cases of ambiguity, clarification dialogues may be needed. Although some very basic mechanisms for dealing with unrequested information exists on VoiceXML, there is no support e.g. for clarification subdialogues. Also, VoiceXML provides only very limited support for information sharing between subtasks and dealing with several simultaneous tasks; these are all dealt with by GoDiS.

Question accommodation combined with (very basic) belief revision abilities also allows GoDiS to reaccommodate questions which have previously been resolved. Finally, a version of reaccommodation, where reaccommodation of one issue requires reaccommodation of a dependent issue as well, allows for successive modifications of database queries.

1.3 Flexible menu-based dialogue

While menu interfaces are ubiquitous in modern technology they are often tedious and frustrating. The mechanisms of accommodation used in GoDiS offers the possibility of allowing the user to present several pieces of relevant information at one time or to present information in the order in which the user finds most natural. This means that users can use their own conception of the knowledge space and not be locked to that of the designer of the menu system.

We extend our theory and the GoDiS system to handle action-oriented dialogue (AOD), which involve DPs performing non-communicative actions such as e.g. reserving tickets. In addition to issues and questions under discussion, this system also has to keep track of actions. The concept of issue accommodation is extended to include action accommodation. We illustrate AOD with an implementation of a VCR control system, whose dialogue plans are based on menus.

Although menu-based dialogue is possible also in VoiceXML, the limitations mentioned in the previous section apply here as well, i.e., problems with information sharing between subtasks,

switching between several tasks, and user utterances that fit with several tasks.

1.4 Negotiative dialogue

In GoDiS, we regard negotiative dialogue as dialogue where several alternative solutions (answers) to a problem (question or issue) can be discussed and compared before a solution is finally settled on. Sidner (1994a) is aware of this aspect of negotiation, and notes that “maintaining more than one open proposal is a common feature of human discourses and negotiations.” What we want to do is to find a way of capturing this property independently of grounding and of other aspects of negotiation, and use it as a minimal requirement on any dialogue that is to be regarded as negotiative.

GoDiS is readily extendible to handle negotiative dialogue (Larsson (2002b)) where it is important to be able to refer to and discuss previously mentioned referents. VoiceXML does not support more complex dialogue; for example, there is no support for representing mentioned referents and implement referent identification.

1.5 Conditional responses

A flexible and natural dialogue is characterized by different ways of responding to questions. For instance, typical responses in an inquiry-oriented dialogue (a.k.a Information-Seeking Dialogue, ISD) are the short answers. Such answers contain just the response particles *yes*, *no* or *ok* and only affirm or negate the propositional content of the question without conveying any additional information. However, there are situations in which such answers may be insufficiently collaborative with respect to the task. For instance, in the travel domain, the task is to determine a set of parameters of a possible journey with respect to a database to which only the system has access. In a typical application like GoDiS, the system collects from the user a set of parameters constraining a journey by asking questions. It then performs a database search with these constraints. If the search succeeds, GoDiS returns the price of the journey. Otherwise, it indicates the search has failed.

However, an information-seeking dialogue often does not stop at the result of the database search, be it negative or positive. The user may revise or refine some parameter(s) and initiate a new search. For example, after the system has indicated that the search has failed, the user may try to change a parameter to achieve successful search, say by changing the departure day. Then, upon success, the user may continue by trying to further constrain the search by specifying an additional parameter, for example the airline. However, such continuations can become dull as

the user is trying to find out which combinations of parameters succeed, as in (8c)-(8f).

- (8) a. U: Can I fly on the second?
- b. S: Yes.
- c. U: Can I fly with Ryanair?
- d. S: Sorry, there is nothing matching your request.
- e. U: What about Lufthansa?
- f. S: Sorry, there is nothing matching your request.

In order to avoid such unnatural and inefficient dialogues, we can enable the system to help the user in finding a satisfiable set of parameters by providing responses that help the user to revise or refine the initial parameters. This involves the system being able to indicate a parameter to relax upon failed database search, or to indicate a parameter to keep in cases where some hits are found but they are too many to enumerate, and thus the search criteria need to be refined. One useful way of accomplishing collaboration in the parameter revision and refinement phase is by providing a *conditional response* (CR): a positive or negative response clarifying the condition(s) under which this response holds. It is the making of the condition explicit that makes the response collaborative. For example, a negative CR can be collaborative by mentioning such parameter(s) in the condition, whose relaxation could result in a positive response instead (*economy class* in (b)). A positive CR can be collaborative by mentioning such parameter(s) that are necessary for preserving the positive response (*business class* in (12c)).

- (9) a. U: Can I fly on the second?
- b. S: Not if you want to fly economy class.
- c. S': Yes, if you can fly business class.

The result of the database search, and therefore the answer to a user's question, can also be contingent on a parameter the user has not specified. In this case, too, it makes sense to indicate this contingency to the user: in (10b) the system gives a positive answer to the question and also indicates that the database search is successful (and thus the answer to the question is positive) as long as an additional parameter, namely the SAS airline, is assumed.

- (10) a. U: Can I fly on the second?
- b. S: Yes, if you can fly with SAS.

To simulate this collaborative system behaviour, we implement CRs in the ISU approach as an additional kind of responses that can be produced and interpreted by GoDiS. Both the production and interpretation of CRs in GoDiS involve some modification and extension of the update and selection rules as well as the analysis, generation and search components in GoDiS.

1.6 Tutorial dialogue

In Chapter 6 we address the issue of developing a tutorial dialogue system using the TrindiKit toolkit and the GoDiS system as a way of evaluating the potential of the framework implemented within TrindiKit/GoDiS to deal with more complex dialogue phenomena.

There are several reasons for choosing tutorial dialogue as an evaluation measure. First, it is a genre that has not been addressed within the Trindi or Siridus project. But also in more general terms, tutorial dialogues have characteristics which make them different from other genres, such as information seeking or action-oriented dialogues.

In particular, tutorial dialogues present a challenge for dialogue management, because they exhibit more complex patterns of mixed dialogue and task initiative. This is necessary especially if the tutor's strategy is to encourage active learning by carefully guiding the student through solving the problem at hand while avoiding to reveal any part of it directly. A consequence of that is that the tutor's behaviour does not appear cooperative at first glance, because she does not answer the student's questions to the best of her ability, and she avoids revealing information (Section 6.2).

Another challenge in tutorial dialogue is in the role of planning, plan recognition and reasoning. In order to provide efficient feedback to the student, the tutor needs to follow the student's reasoning rather than superimpose her own reasoning on the student. In addition, the tutor needs to evaluate the student's contributions on a scale of "correctness" rather than straightforwardly dismissing (or correcting) wrong answers. The tutor also needs to reason about possible sources of mistakes, so as to provide appropriate explanations and give relevant hints. Therefore, a lot of consideration has to go into the way domain knowledge is modeled and used in the system (Section 6.4.3).

Giving useful and efficient feedback to the student also requires tailoring of the tutor's contributions to the context and to the student model. This in turn requires elaborate dialogue move taxonomies, and rich models of the dialogue state and history, in order to support fine-grained decisions about what the tutor should convey (Section 6.4.1).

Finally, having to deal with a complex interplay of factors is a challenge for the overall dialogue modeling task from an engineering point of view. A modular system design is a prerequisite for the development of such a complex system.

1.7 Conflicts

Enabling a dialogue system with the desired flexibility and cooperative behaviour is especially relevant in the case of NCLDs. First, since the system in NCLD applications is usually *dispensable*. Its role is to make our lives easier. If it fails to achieve that goal, the user may just ignore it and proceed to perform the desired function in the ordinary way (by pressing a sequence of digits in the telephone pad, light switches, etc.).

In this kind of scenario the system should be as cooperative as possible, trying to avoid a situation of frustration on the other participant which would indeed lead to interrupting any communication between them. So, what is usually modelled in NCLD systems is a *collaborative* behaviour in which the participants are in some sense working together to reach the desired outcome.

Second, systems implementing NCLDs should incorporate cooperative behaviour in order to anticipate and provide an adequate response to conflicts arising in the course of the dialogue.

Conflict arises from different sources in NCLDs.

1. First, the user may just not know exactly what functionality is available, or the parameters which they require. Providing help is one way of solving this conflict.
2. Second, given the dynamic nature of the domains in which we are working, one of the agents (the user) may just ignore the exact state of the world at that precise moment. Requesting information is then a collaborative activity which the system should model.
3. A third type of conflict arises when one of the agents wrongly believes that a specific goal is possible, but the state of the world does not permit its accomplishment.
4. The last (and more productive) source of conflict arises from misunderstandings stemming from misrecognition.

Finally, in addition to reacting to possible conflicts, some advanced modes of cooperation may actually transfer the initiative to the system so that it anticipates what action the user might want to perform, or how she usually likes to perform a specific action.

1.8 Over informative answers and clarification questions

Long prompts which provide the possible responses a user can make (e.g. “Do you want current accounts, borrowing, saving, insurance or foreign exchange”) can become tiresome, especially

if the number of options is large. However, as soon as we adopt shorter prompts e.g. “which service do you want?” there is a potential for mismatch between the level of detail the user supplies, and what is expected by the system. The user may provide more detail e.g. “I want home insurance” or “I want to insure a Ford Focus 1.6”, or less e.g. “banking”. If the user provides more information this should not be thrown away, since it might be useful later in the dialogue. If the user supplies less, the system should provide a clarification question which shows that it understood the users response, but requires more detail e.g. “what kind of banking do you require?”.

As we move from system initiated dialogues to more free ranging dialogues in ontologically rich domains, the potential for mismatch increases hugely, and there becomes a need for a general mechanism to deal with clarifications. In this deliverable we discuss the use of ontological information in the Linguamatics Dialogue Manager to automatically generate appropriate clarification questions.

Part I

Flexible Issue-based Dialogue Management

Chapter 2

Grounding issues

2.1 Introduction

In the basic GODIS system reported in Larsson *et al.* (2002), we assumed “perfect communication” in the sense that all utterances were assumed to be correctly perceived and understood, and fully accepted¹. Of course, these assumptions are unrealistic both in human-human and human-computer conversation. A useful dialogue system needs to be able to deal with cases of miscommunication and rejections.

We will not attempt to give a complete computational theory about the grounding process in human-human dialogue. Rather, we will provide a basic issue-based account, influenced by Ginzburg, which tries to cover the main phenomena that a dialogue system needs to be able to handle. For instance, the fact that speech recognition is much harder for machines than for humans may motivate different grounding strategies for handling system utterances than for handling user utterances.

First, we provide some dialogue examples where various kinds of feedback are used. We then review and discuss some relevant background, and discuss general types and features of feedback as it appears in human-human dialogue. Next, we discuss the concept of grounding from an information update point of view, and introduce the concepts of optimistic, cautious and pessimistic grounding strategies. This is followed by the main section of this chapter, where we relate grounding and feedback to dialogue systems, discuss the implementation of issue-based grounding and feedback in GODIS2, and provide dialogue examples showing the system’s behaviour and how it relates to internal updates. We then review additional implementation issues, and provide a final discussion.

¹This chapter is a slightly altered version of Chapter 3 in Larsson (2002a).

2.1.1 Dialogue examples

The human-human dialogue excerpt² in (1) shows two common kinds of feedback. J's "mm" shows that J (thinks that he) understood P's previous utterance; P's "pardon" shows that P was not able to hear J's previous utterance. The example also includes a hesitation sound ("um") from J. (P is a customer and J a travel agent.)

- (1) P : öm (.) flyg ti paris
 um (.) flight to paris
 J : mm (.) ska du ha en returbiljett
 mm (.) do you want a return ticket
 P : va sa du
 pardon
 J : ska du ha en tur å retur
 do you want a round trip

The feedback in (1) consisted of conventionalized feedback words ("mm", "pardon"). However, feedback may also be more explicit and repeat the central content of the previous utterance, as K's second feedback utterance in (2).

- (2) B : ja ska va framme i [₁ göteborg]₁ e e ungefär vi nietiden
 om de finns nå tidit [₂ morgonflyg]₂
 I need to be in Gothenburg er er around nine if there is an
 early morning flight
 K : [₁ m]₁
 m
 K : [₂ vi]₂ nietiden m vi ska se
 Around nine m let's see

The function of an utterance answering a question is not primarily to give feedback, but rather to provide task-related information. However, an answer also shows that the previous question was understood and integrated. Example (3) shows that feedback is sometimes given in reaction to a question before the question is answered.

- (3) J : sen måste du ha e sån där intenationellt studentkort också
 ha du de
 then you need one of those international student cards do
 you have that
 P : mm nä
 mm no

²Except where noted, the human-human dialogues in this chapter have been collected by the University of Lund as part of the SDS project. We quote transcriptions done in Göteborg as part of the same project. The (.) symbol indicates a pause.

However, often human-human communication proceeds without explicit feedback, as in the excerpt from a telephone dialogue in (4).

- (4) K : vart ska du åka
where are you going
B : e jag ska åka till göteborg
er i'm going to gothenburg
K : vilket datum
what date
B : den sjätte juni
the sixth of june
K : vill du åka me flyg eller me tåg
do you want to fly or take the train

By contrast, in current commercial systems it is common to check the system's interpretation of recognized user input asking a y/n-question, as seen in the (invented) dialogue in (5). This behaviour is much less common in human-human dialogue, but may occur in noisy environments or when communicating over an unreliable communication channel (e.g. when using a mobile phone on a train).

- (5) S : What day do you wish to travel?
U : Friday
S : Friday the 5th of April, is that correct?

A similar but not identical behaviour does occur in human-human dialogue, as in (6), taken from the London-Lund corpus and used by Clark and Schaefer (1989a). In this example, B gives an explicit report of how A's utterance was interpreted, whereupon A confirms that the interpretation is correct by saying "yes".

- (6) A : F . six two
B : F six two
A : yes
B : thanks very much

The dialogue in (DIALOGUE 2.1) gives an impression of what feedback capabilities we aim to add to GODIS in this chapter.

(DIALOGUE 2.1)

S> Welcome to the travel agency!

S> hello
 U> price information please
 S> You asked about price. How do you want to travel?
 U> pardon
 S> How do you want to travel?
 U> by plane
 S> By plane, is that correct?
 U> yes
 S> Okay. What city do you want to go to?
 U> lododo
 S> Pardon?
 U> london
 S> Okay, to london. What city do you want to go from?
 U> lyons
 S> Sorry, Lyons is not in the database. So, What city do you want to go from?
 U> paris
 ...
 U> What about connecting flights?
 S> You asked about connecting flights. Sorry, I cannot answer that.

2.2 Background

In this section, we provide a brief overview and of some previous accounts of grounding and feedback.

2.2.1 Clark: Adding to the common ground

Clark and Schaefer (1989a) describe grounding as the process of adding to the common ground (see Larsson *et al.* (2002)). They also formulate the *grounding criterion*:

Grounding criterion: The contributor and the partners mutually believe that the partners have understood what the contributor meant, to a criterion sufficient for current purposes. (Clark and Schaefer, 1989a, p. 148)

To achieve this, each grounding process goes through two phases:

- **Presentation phase:** A presents utterance u for B to consider. He does so on the assumption that, if B gives evidence e or stronger, he can believe that B understands what A means by u .
- **Acceptance phase.** B accepts utterance u by giving evidence e' that he believes he understands what A means by u . He does so on the assumption that, once A registers evidence e' , he will also believe that B understands. (Clark and Schaefer, 1989a, p. 151)

Clark (1996) argues that utterances involve actions on (at least) four different levels:

(7)

Level	Speaker A's actions	Addressee B's actions
4	A is proposing a joint project w to B	B is considering A's proposal of w
3	A is signalling that p for B	B is recognizing that p
2	A is presenting signal s to B	B is identifying s
1	A is executing behaviour t for B	B is attending to t

Examples of joint projects are adjacency pairs, e.g. one DP asking a question and the other answering it. According to Clark, these four levels of action constitute an *action ladder*, and as such it is subject to the principle of *downward evidence*: “In a ladder of actions, evidence that one level is complete is also evidence that all levels below it are complete”. For example, if H understands u , H must also have perceived u and H and S must have established contact; however, H may not accept u .

In Clark and Schaefer (1989a), it is unclear whether grounding includes the proposal / consideration level in addition to understanding³. However, in Clark (1996), grounding is redefined to include all levels of action, i.e. attention, identification, recognition and consideration.

To ground a thing (...) is to establish it as part of common ground well enough for current purposes. (...) On this hypothesis, grounding should occur at all levels of communication. (Clark, 1996, p.221, italics in original)

We will adopt this general use of the term grounding to include all four action levels. Also, we assume that the acceptance phase (potentially) concerns all four action levels, rather than only understanding⁴.

³The definition suggests only understanding is involved, but some examples indicate that utterances which are rejected because of being inappropriate are not grounded.

⁴The term “acceptance phase” is a bit unfortunate, since “acceptance” is used by e.g. Ginzburg to designate the proposal-consideration action level.

Clark lists five ways to signal that a contribution has been successfully interpreted and accepted, ordered from weakest to strongest:

- Continued attention
- Relevant next contribution
- Acknowledgement: “uh-huh”, nodding, etc.
- Demonstration: reformulation, collaborative completion
- Display: verbatim display of presentation

The presentation and acceptance phases both focus on externally observable communicative behaviour. However, corresponding to presentations by a speaker on each level of action there is also an “internal” action carried out by the addressee.

Clark views the proposal-consideration process in terms of negotiation, where an utterance such as an assertion or a question is seen as a proposal for a joint project, followed by a response to this proposal. Clark follows Goffman (1976) and Stenström (1984) in distinguishing four main types of responses to proposals of joint projects:

1. full compliance, e.g. answering a question [acceptance]
2. alteration of project, where *H* alters the proposed project to something he is willing to comply with; Clark asserts that alterations may be cooperative (in which case the altered project is still relevant to the original one) or uncooperative [alteration]
3. declination of project, where *H* is unable or unwilling to comply with the project as proposed. Declinations are often performed by offering a reason or justification for declining the proposal. Clark gives the response “I don’t know” to a question as an example of declination. [rejection]
4. withdrawal from project, where *H* withdraws from considering the proposal, e.g. by deliberately ignoring a question and changing the topic [withdrawal]

2.2.2 Ginzburg: QUD-based utterance processing protocols

Ginzburg offers an issue-based model of grounding on the understanding and acceptance levels by positing two kinds of grounding-related questions: meaning-questions and acceptance questions⁵. If A produces utterance *u*, B is faced with a meaning-question, roughly “What does *u*

⁵The latter term is ours. It refers to Ginzburg’s MAX-QUD questions discussed below.

mean?”. If B cannot find an answer to this question, B should produce an utterance identical or related to the meaning-question, e.g. “What do you mean?”. If B manages to find an answer to this question, he proceeds to consider the acceptance-question, roughly “Should *u* be accepted?”.

Ginzburg’s utterance processing protocol (pt. 1) Ginzburg formulates his theory in terms of an utterance processing protocol. Assuming the other DP *A* has uttered *u*, this is roughly what happens in the first part of the protocol:

B is faced with the *content-question* $q_{content}(u)$, which we formalize as $?x.content(u, x)$, paraphrasable roughly as “What does *u* mean (given the current context)?”. To answer this question, *B* must be able to provide a contextual interpretation *c* of *u*. This involves, among other things, finding referents for NPs. If *B* is not able to answer $q_{content}(u)$, *B* places $q_{content}(u)$ on QUD and produces a $q_{content}(u)$ -specific utterance, e.g. a request for clarification. Once an answer to $q_{content}(u)$ has been found, *B* can be said to have an understanding of *u* (which may, of course, be a misunderstanding).

Ginzburg notes that utterances behave differently with regard to acceptance depending on whether they have propositions or questions as content. A proposition *p* can be accepted in two ways: as a fact or as a topic (issue) of discussion. In the latter case, the question under discussion is, roughly, whether *p* should be accepted as a fact (at least for the purposes of current discussion) or not. Accepting a proposition entails accepting it also as an issue for discussion (although the “discussion” in this case will consist only of the acceptance of *p* as a fact). The exchanges in (8) show some examples of reactions to assertions (note that these examples are not Ginzburg’s).

- (8) a. A: The train leaves at 10 a.m. [answer/assert *p*]
 B: OK, thanks. [accept *p*]
- b. A: The train leaves at 10 a.m. [answer/assert *p*]
 B: No it doesn’t! [reject *p*, accept ?*p* for discussion]
- c. A: The train leaves at 10 a.m. [answer/assert *p*]
 B: I’d prefer not to discuss this right now [reject ?*p* for discussion]
- d. A: The train leaves at 10 a.m. [answer/assert *p*]
 B: Nice weather, isn’t it [ignore *p*]

Questions, by contrast, can only be accepted as issues for discussion. However, accepting *q* does not necessarily result in answering *q*. On this account, answering *q* should be viewed as one possible way of displaying internal acceptance of *q*; however, contrary to Clark we also allow the possibility of displaying acceptance of *q* without answering *q*.

- (9) a. A: Where do you want to go [ask q]
B: Paris [answer q, implicitly accept q]
- b. A: Where do you want to go [ask q]
B: Hmmm, good question... Do you have any recommendations? [explicitly accept q]
- c. A: Where do you want to go [ask q]
B: That's none of your business [explicitly reject q because of unwillingness]
- d. A: Where do you want to go [ask q]
B: I don't know [explicitly reject q because of inability]
- e. A: Where do you want to go [ask q]
B: I'd like to travel in April [ignore q, answer other question]
- f. A: Where do you want to go [ask q]
B: Do you have a student discount? [ignore q, ask other question]

Ginzburg's utterance processing protocol, pt. 2 As we saw above in Section 2.2.2, according to Ginzburg's utterance processing protocol, for a DP B to understand an utterance u amounts to finding an answer to the content-question q_c .

Once B is able to find an answer c which resolves $?x.\text{content}(u, x)$, B is faced with the question $q_{\text{accept}}(c)$ of whether or not to accept c for discussion, formalized by Ginzburg as $?MAX\text{-}QUD(c)$ ("Whether c should become QUD-maximal"⁶). At this point, the protocol is different for questions and propositions ("facts"). If c is a question and B answers $q_{\text{accept}}(c)$ negatively (rejects c for discussion), B pushes c on QUD and produces a c -specific utterance (e.g. "I don't want to discuss that"). If $q_{\text{accept}}(c)$ is answered positively and B accepts c , c will be added to QUD and B will produce a c -specific utterance, e.g. an answer to the question c .

If c instead is a proposition and if B answers $q_{\text{accept}}(c)$ negatively B should push $q_{\text{accept}}(c)$ on QUD and produce a $q_{\text{accept}}(c)$ -specific utterance. But if $q_{\text{accept}}(c)$ is answered positively, B must now consider the question whether c , i.e. $?c$. If the answer is negative (i.e. B does not accept c), the corresponding y/n -question $?c$ is pushed on QUD. This amounts to accepting $?c$ for discussion, which is not the same as accepting c . If B answers $q_{\text{accept}}(c)$ positively, B should add c to her FACTS.

⁶This means that the arguably more intuitive interpretation of $?MAX\text{-}QUD(c)$ as "whether c is maximal on QUD" is wrong.

For clarity, we reproduce the full protocol in a more schematic way:

try to find an answer resolving $q_{content}(u) = ?x.content(u, x)$

- no answer found \rightarrow push $q_{content}(u)$ on QUD, produce $q_{content}(u)$ -specific utterance
- answer c found \rightarrow
 - c is a question \rightarrow consider $q_{accept}(c) = ?MAX-QUD(c)$
 - * decide on “no” \rightarrow push $q_{accept}(c)$ on QUD, produce $q_{accept}(c)$ -specific utterance [reject c]
 - * decide on “yes” \rightarrow push c on QUD, produce c -specific utterance [accept c]
 - c is a proposition \rightarrow consider $q_{accept}(?c)$
 - * no \rightarrow push $q_{accept}(?c)$ on QUD, produce $q_{accept}(?c)$ -specific utterance [reject $?c$ as topic for discussion]
 - * yes \rightarrow consider $?c$ [accept $?c$ as topic for discussion]
 - no \rightarrow push $?c$ on QUD, produce $?c$ -specific utterance [reject c as fact]
 - yes \rightarrow add c to FACTS [accept c as fact]

Note that there are a number of decisions that need to be made by B , and for each of these decisions there is the possibility of rejecting u on the corresponding level. For a question, there is only one way of rejecting it (once the content question has been resolved): to reject it as a question under discussion. This amounts to refusing to discuss the question. For a proposition p , there are two different ways of rejecting it. Firstly, one may reject the issue “whether p ” completely; this amounts to refusing to discuss whether p is true or not. Alternatively, one may accept “whether p ” for discussion but reject p as a fact.

2.2.3 Allwood: Interactive Communication Management

Allwood (1995) uses the concept of “Interactive Communication Management” to designate all communication dealing with the management of dialogue interaction. This includes feedback but also *sequencing* and *turn management*. Sequencing “concerns the mechanisms, whereby a dialogue is structured into sequences, subactivities, topics etc. ...”.

Here, we will use the term ICM as a general term for coordination of the common ground, which in an information state update approach comes to mean explicit signals (e.g. utterances) enabling coordination of updates to the common ground. While feedback is associated with the grounding of specific utterances, ICM signals in general does not need to concern any specific utterance.

As will be seen below, we will also be making use of various other parts of Allwood’s “activity-based pragmatics” (Allwood, 1995), including Allwood’s action level terminology, the concept of Own Communication Management (OCM), and various distinctions concerning ICM.

2.3 Preliminary discussion

In the previous section we have seen examples of different ways of accounting for grounding and feedback. We feel that they all offer useful insights, and that they together can serve as a basis for our further explorations.

Therefore, in this section we will discuss the accounts presented in Section 2.2, relate them to each other, and establish some basic principles and terminological conventions.

2.3.1 Levels of action in dialogue

Both Allwood (1995) and Clark (1996) distinguish four levels of action involved in communication (S is the speaker of utterance u , H is the hearer/addressee). They use slightly different terminologies; here we use Allwood’s terminology and add Clark’s (and, for the reaction level, also Ginzburg’s) corresponding terms in parenthesis. The definitions are mainly derived from Allwood.

- Reaction (acceptance, consideration): whether H has integrated (the content of) u
- Understanding (recognition): whether H understands u
- Perception (identification): whether H perceives u
- Contact (attention): whether H and S have contact, i.e. if they have established a channel of communication

These levels of action are involved in all dialogue, and to the extent that contact, perception, understanding and acceptance can be said to be negotiated, all human-human dialogue has an element of negotiation built in. Note that the above list of levels is formulated in terms of the hearer’s perspective.

Given that grounding is concerned with all levels, it follows that four aspects of an utterance u in a dialogue between H and S can in principle be represented in the common ground, one for each action level:

- whether u has been integrated (taken up, accepted)
- whether u has been understood
- whether u has been perceived
- whether S and H have contact

Also, grounding-related feedback may concern any (and possibly several) of these levels.

The level referred to as reaction/acceptance/consideration in the list above is defined differently by different authors. Allwood calls it “reaction (to main evocative intention)”, Ginzburg talks about “acceptance”, and Clark uses the term “consideration (of joint project)”. Perhaps it could be argued that these different definitions are not concerned with the exact same phenomena. Since we want to use the distinction rather than debate it, we choose to emphasize the similarities rather than the differences.

2.3.2 Reaction level feedback

Once an utterance has been understood (or is believed to be understood), in the sense that the hearer has interpreted the utterance to have a meaning and purpose which is relevant in the activity (as perceived by the hearer), the hearer must decide what to do with the utterance. Should he, for example, try to answer the question that was asked, or refuse? Should he choose to commit to an asserted proposition, or raise objections?

The reaction process which follows the understanding of a move M can be analytically divided into three substeps:

- consideration: whether or not to accept and integrate M (and consequently (try to) act on the evocative intention)
- integration: updating the common ground according to M
- feedback: signalling the results of consideration of M

The division of the reaction phase into consideration and feedback is also made in Allwood (1995), using the terms “evaluation” and “report” (respectively), and (though perhaps not so explicitly) in Clark (1996). However, the integration step is not (at least not explicitly) included in either of these accounts.

In the consideration phase, the DP investigates whether he can and wants to accept the proposed joint project or not. If not, he needs to decide whether to alter, decline, or ignore the proposal.

We will use the term *integration* for the silent (internal) consequence of deciding to accept (comply with) a proposed joint project, modelled as the process of updating one's view of the common ground with the full effects of a performed move. By "the full effects" we mean, for example, taking a proposition to be true (at least for the purposes of the conversation) or taking a question as being under discussion. In relation to Clark's use of "uptake", we would say that uptake signals integration. (Of course, uptake may be more than merely a signal that a previous utterance has been integrated, e.g. in the case of answering a question.)

In the feedback phase, the results of the consideration process (acceptance or rejection of issue or proposition) are signalled. Allowing for the possibility of silent acceptance means that the feedback phase is optional, that is, utterances can be accepted without any feedback being produced.

Issue and fact acceptance Extending Clark's terminology, we can call the acceptance of a proposition or question as a topic for discussion *issue acceptance*, and the acceptance of a proposition as a fact *fact acceptance*. (We will also use the term *proposition acceptance* for the latter). The former kind of acceptance is available both for questions and propositions, while the latter is available only for propositions. Correspondingly, we can make a distinction between issue rejection and fact rejection in the case of propositions.

Reasons for utterance rejection If a question is asked and the addressee DP decides not to accept it (explicitly or implicitly), this may be explained in at least two ways:

- unwillingness: DP does not want to discuss the issue, e.g. because DP believes other information is more important at the moment
- inability: DP is not able to discuss the issue, e.g. because of confidentiality or lack of knowledge

Regarding the update effects of declining a question, there seems to be an important difference between being unable to answer a question (as e.g. in the case where the response is "I don't know"), and being unwilling to answer it. In the former case, it is not clear that the question is actually rejected as a topic for discussion. The addressee of the question may think that he might eventually come up with an answer (as a result of new information or inference); in this case "I don't know" can be interpreted as "I don't know right now, but I'll keep the question in mind". In this case, the question might not have to be explicitly raised again before being responded to.

In the case where the rejection displays unwillingness to answer the question (e.g. “No comment”, “I will not answer that”, “That’s none of your business”), it is much clearer that the question is actually rejected as a topic for discussion.

There is also a difference between questions and propositions regarding the reasons for rejection. As Ginzburg notes, asserted propositions may be rejected as issues for discussion, but even if accepted as issues they may be rejected as facts. So for propositions, the consideration phase is more complex than for questions, potentially involving two decisions (e.g. rejecting the asserted proposition as a fact, but accepting it as an issue).

Rejecting a proposition as an issue can be explained by the same kinds of reasons as for any issue. Rejecting a proposition as a fact may be caused, for example, by the addressee having a conflicting belief, or not trusting the speaker. It may also be explained by a belief that accepting the proposition will not serve the goals of the DPs. If a customer in a travel agency asserts that the destination city of her flight is Kuala Lumpur, when in fact the agency only serves destinations in Europe. Of course, the proposition that the customer *wants* to travel to Kuala Lumpur can hardly be rejected by the clerk; however, the proposition that Kuala Lumpur is the destination city of a trip that the clerk will provide information about can be rejected⁷. This kind of example is especially relevant for database search systems, where information about the user’s desires and intentions is not stored as such.

2.3.3 Levels of understanding

Concerning the levels of action described in Section 2.2.1, we can make further distinctions between different levels of understanding, corresponding to three levels of meaning. These sub-levels give a finer grading to the level of understanding. (A similar distinction is also used by Ginzburg (forth)).

- domain-dependent and discourse-dependent meaning (roughly, “content” in the terminology of Barwise and Perry, 1983 and Kaplan, 1979)
 - referential meaning , e.g. referents of pronouns, temporal expressions
 - pragmatic: the relevance of *u* in the current context
- discourse-independent (but possibly domain-dependent) meaning (roughly corresponding to “meaning” in the terminology of Barwise and Perry, 1983 and Kaplan, 1979), e.g. static word meanings

⁷The reason behind the clerk’s rejection in this case is that accepting the user’s answer would make the clerk unable to perform the requested communicative action to answer the user’s question about price.

By “discourse-independent” we mean “independent of the dynamic dialogue context” (modelled in GODIS by the information state proper). However, discourse-independent meaning may still be dependent on static aspects of the activity/domain. It is obvious that these levels of meaning are intertwined and do not have perfectly clear boundaries. Nevertheless, we believe they are useful as analytical approximations.

Since dialogue systems usually operate in limited domains, we will assume that we do not have to deal with ambiguities which are resolved by static knowledge related to the domain. For example, a dialogue system for accessing bank accounts does not have to know that “bank” may also refer to the bank of a river; it is simply very unlikely (though of course not impossible) that the word will be used with this meaning in the activity. It can be argued whether this is always a good strategy, but for now we accept this as a reasonable simplification.

2.3.4 Some comments on Ginzburg’s protocol

The first thing to note about Ginzburg’s grounding protocol is that it does not specify exactly what kind of feedback should be produced. The notion of question-specificity (see Larsson *et al.* (2002)) is a minimal requirement that needs to be supplemented with additional heuristics to decide on exactly what feedback to provide. Also, it does not specify how a DP decides when a satisfactory interpretation has been found, or how to resolve the content- and acceptance questions. These are all things we need to be specific about when implementing a dialogue system. (Of course, to the extent they are domain-dependent, we would not expect to find them in a general theory of dialogue. Whether they are domain-dependent or not is, on our view, an open question.)

Second, Ginzburg seems to assume a certain degree of freedom concerning the sharedness of QUD. According to the grounding protocol, DPs are free to add a grounding-related question q to QUD without informing the other DP(s), provided this is followed by an utterance specific to the added question. In fact, the mechanism of question accommodation that will be presented in Chapter 3 provides an explanation of how DPs can understand answers to unasked questions. However, it is not clear that this should allow the speaker to modify QUD *before* uttering the q -specific utterance. It seems inconsistent to say that a DP that assumes QUD is shared can modify QUD without having given any indication of this to the other DP; how would the other DP be able to know about this modification before the q -specific utterance has been made? So it appears that Ginzburg has a notion of QUD as not necessarily entirely shared, and this is slightly different from the notion of QUD we are using. (See also Section 3.8.5.) Note that on our account different DPs can still have different views of QUD (in the case of as yet undiscovered misunderstandings); however, they always assume that the other DP has the same view of QUD as themselves, or else they would not be assuming that it was shared.

Third, Ginzburg only deals with understanding and acceptance; contact and perception are left

out. So the protocol above does not deal explicitly with cases where a DP is unsure which words were uttered (however, it deals with perception indirectly since understanding is based on perception).

Relation between Clark's and Ginzburg's accounts

It seems possible to draw some parallels between the two accounts reviewed above. Clark's "recognition of meaning" would presumably be modelled by Ginzburg as finding an answer to the content-question. Similarly, Clark's "consideration of proposal" is modelled as consideration of the acceptance question.

Clark talks about joint projects in "track 2" (meta-communication, as opposed to "track 1", for task-level communication) as involving speakers (often implicitly) raising various issues related to grounding, e.g. "Do you understand this?", and the responder answering these issues (often implicitly). This fits well with the issue-based approach proposed by Ginzburg. A single utterance may include both feedback and domain-level information. Domain-level information is on track 1 while ICM is on track 2.

On Clark's account, there is no asymmetry between questions and propositions concerning acceptance. The question-related counterpart of accepting a proposition as a fact, according to Clark, is answering the question. However, it can be argued that answering the question is merely an external behaviour caused by (and acting as positive feedback concerning) the actual acceptance of the question as an issue. Clark's question-related counterpart of rejecting a proposition as a fact (*declination*) is answering e.g. "I don't know" (Clark, 1996, p. 204), and thereby signalling lack of ability or willingness to "comply with the project as proposed". Presumably, "No comment" or "I refuse to answer that question" would also count as rejections on the same level, which indicates that they are really issue-rejections. The "withdrawal" that Clark talks about, where the addressee deliberately ignores a proposal, seems to be equally applicable to both assertions and questions. (In fact, Lewin (2000) views cases where a DP answers a different question than the one that was asked, thereby withdrawing from the proposed question, as rejections.)

2.4 Feedback and related behaviour in human-human dialogue

By feedback we mean behaviour whose primary function is to deal with grounding of utterances in dialogue⁸. This distinguishes feedback from behaviour whose primary function is related to the domain-level task at hand, e.g. getting price information. Non-feedback behaviour in this sense includes asking and answering task-level questions, giving instructions, etc. (cf. the “Core Speech Acts” of Poesio and Traum, 1998b). Answering a domain-level question (e.g. saying “Paris” in response to “What city do you want to go to?”) certainly involves aspects of grounding and acceptance, since it shows that the question was understood and accepted. However, the primary function of a domain-level answer is to resolve the question, not to show that it was understood and accepted.

In this section we will attempt to give an overview of various aspects of feedback. We will return to sequencing ICM in Section 2.6.9.

2.4.1 Classifying explicit feedback

To get an overview of the range of explicit feedback behaviour that exists in human-human dialogue, we will classify feedback according to five criteria. We will assume that DP S has just uttered or is uttering u to DP H , when the feedback utterance f (uttered by H to S) occurs.

- level of action / basic communicative function (contact, perception, understanding, reaction / acceptance)
- polarity (positive / negative): whether f indicates contact / perception / understanding / acceptance or lack thereof
- eliciting / non-eliciting: whether f is intended to evoke a response (e.g. a reformulation or a reason to accept some content)
- form of f : single word, repetition etc.
- content of f : object-level or meta-level

The action level criterion has been explained above; the others will be explained presently. The criteria of basic communicative function, polarity, eliciting/non-eliciting, and surface form are all derived from Allwood *et al.* (1992) and Allwood (1995).

⁸Since this thesis is not concerned with multimodal dialogue, we will only discuss verbal feedback.

2.4.2 Positive, negative, and neutral feedback

Positive feedback indicates one or several of contact, perception, understanding, and integration, while negative feedback indicates lack thereof.

While there are clear cases of positive (“uhuh”, “ok”) and negative (“pardon?”, “I don’t understand”) feedback, there are also some cases which are not so clear. For example, are check-questions (e.g. “To Paris?” in response to “I want to go to Paris”) positive or negative? If positive feedback shows understanding, and negative feedback lack of understanding, then check-questions are somewhere in between; they indicate understanding but also that the lack of confidence in that understanding.

Here we will assume a third category of *neutral* feedback for check-questions and similar feedback types. If negative feedback indicates a lack of understanding, neutral feedback indicates lack of confidence in one’s understanding.

Negative feedback can be caused by failure to integrate *U* on any of the levels of action in dialogue:

- lack of contact - H did not notice that S said something
- lack of perception - H did not hear what S said
- lack of understanding on a semantic/pragmatic level - H recognized all the words, but could not extract a content
 - context-independent meaning, e.g. word meanings
 - context-dependent meaning, e.g. referents
 - pragmatic meaning, i.e. the relevance of S’s utterance in relation to the context
- rejection of content (lack of acceptance)

For negative feedback, detecting the level with which the feedback is concerned is important for being able to respond appropriately. Here are some possibilities for the different levels:

- contact: try to establish contact (“Hey there”)
- perception: speak louder, articulate
- understanding
 - meaning: reformulate

- pragmatic meaning: reformulate, or explain how the utterance is relevant
- rejection: abandon or argue for the acceptance of the content

2.4.3 Eliciting and non-eliciting feedback

We will use the term “eliciting feedback”, borrowed from Allwood *et al.* (1992), to refer to feedback utterances intended to elicit a response, or more specifically utterances u' such that u' is intended to make S respond to u' because H is not sure about how to interpret S 's utterance u . Check-questions (both y/n - and alternative-questions) are seen as eliciting feedback in this sense. Eliciting feedback can also occur after S 's utterance u is finished.

Feedback on all levels of action can be eliciting: contact (“are you there?”), perception (“what did you say?”), understanding (“what do you mean?”) and acceptance (“why do you say that?”).

2.4.4 Form of feedback

As with all utterances, feedback utterances can have various syntactic forms:

- assertion
 - declarative (“I heard you say ‘go to Paris’.”, “You want to go to Paris.”)
- imperative (“Please repeat.”)
- interrogative
 - y/n -question (“Did you say ‘Paris’?”, “Do you want to go to Paris?”)
 - wh -question (“What did you say?”, “What do you mean?”, “Where do you want to go?”)
 - alternative-question (“Did you say ‘Paris’ or ‘Ferris’?”, “Do you want to go to Paris, France or Paris, Texas?”)
- ellipsis (“Paris?”, “to Paris.”)
- conventional feedback phrases and words (“Pardon?”, “Beg you pardon?”, “Okay”)

Apart from showing the speaker that one has understood, feedback in the form of an explicit declarative report, repetition or reformulation has the additional function of making sure that

the understanding is actually correct, by providing a chance for correction. A *y/n*-question has a similar function, but it indicates less confidence in the interpretation (i.e. is more neutral) and has a stronger eliciting element than an assertion; a question requires an answer, while an assertion can often be assumed to be accepted in the absence of protest.

A related dimension of classification is how the form of the feedback utterance relates to the previous utterance. One way of giving positive feedback is to simply repeat verbatim the previous utterance (e.g. “To Paris.” in response to “To Paris.”). A similar strategy is to provide a reformulation (e.g. “Your destination city is Paris, the capital of France.”). The latter is perhaps a stronger signal of understanding than the former, since a verbatim repetition does not in principle require that the utterance was understood.

2.4.5 Meta-level and object-level feedback

A final distinction can be made depending on whether the feedback explicitly talks about what the speaker said or meant, in which case the feedback can be said to be *meta-level* feedback, or if instead it talks about the subject matter of the dialogue, in which case we talk about *object-level* feedback.

- Meta-level
 - perception (“Did you say ‘Paris’?”)
 - understanding (“Did you mean that you want to go to Paris?”)
- Object-level (“Do you want to go to Paris?”)

This distinction does not necessarily apply to all kinds of feedback. For example, for conventional phrases like “Pardon?” and elliptical phrases like “Paris?” it is not clear if they refer to what the speaker said or meant, or about the subject matter of the dialogue, or neither.

2.4.6 Fragment feedback / clarification ellipsis

Often, feedback does not concern a complete utterance, but only a part of it; this is the case, for example, with failure to identify a referent to an NP. We can refer to this kind of feedback as *fragment feedback* (exemplified in (10)) and contrast it with *complete feedback* which concerns a whole utterance (as in (11)).

- (10) A: I met Jim yesterday.
 B: Who? [negative partial understanding ICM]
 Who did you say you met? [negative partial understanding ICM + partial positive understanding ICM]
 Jim? [interrogative partial understanding ICM]
 Jim Jones? [intermediate partial understanding]
 Jim Jones or Jim Lewis? [intermediate partial understanding]
 No, it was Bob that you met [partial acceptance, partial rejection¹]
- (11) A: I met Jim yesterday.
 B: Pardon? [negative complete perception]
 B: What do you mean? [negative complete understanding]
 B: Liar! [(probably) complete rejection]

It is also possible to give negative partial feedback to one part of an utterance and simultaneously give positive partial feedback to some other part, as when B says “Who did you say you met?”; here, B gives positive feedback that B understood that A met someone, but negative feedback concerning who A met. Cooper and Ginzburg (2001) discuss negative partial feedback using the term “clarification ellipsis” and give an account in the QUD framework.

2.4.7 Own Communication Management

A further aspect related to feedback and ICM is what Allwood refers to as Own Communication Management, which involves hesitation sounds, such as “um”, “er” etc., (which also have the effect of keeping the turn), and self-corrections (initiated either by the speaker or by the hearer). It should also be noted that some feedback behaviour also has an OCM aspect; for example, one can explicitly accept a question to “buy time” for coming up with an answer.

2.4.8 Repair and request for repair

One type of behaviour very closely related to feedback, and also to Own Communication Management (see below), is what Traum refers to as “other-initiated repair”.

¹That is, acceptance of the proposition “A met someone”, but rejection of the proposition ‘the person that A met was Jim’.

- other-initiated other-repair: repair by A (“You mean Paris.”)
- other-initiated self-repair: request for repair by A (“Do you mean Paris?”)

This overlaps with what Clark calls *alteration*, i.e. the case where A accepts an altered version of the proposed joint project.

2.4.9 Request for feedback

Above, we have analyzed feedback produced by the addressee A in response to an utterance u produced by a speaker S . An additional type of feedback behaviours which are produced by the speaker of u are requests for feedback, e.g. “Do you understand?”, “Got that?”.

- understanding (“Got that?”, “Do you understand?”)
- acceptance (“OK?”, “Do you agree?”)

2.5 Update strategies for grounding

After having reviewed grounding-related interactive communication management in human-human dialogue, we will now explore update strategies related to grounding. In this section, we introduce the concepts of optimism, caution, and pessimism regarding grounding update strategies.

2.5.1 Optimistic and pessimistic strategies

According to Clark and Schaefer (1989a), many models of dialogue make a tacit idealization (1) that DPs assume that the content of each utterance is automatically added to the common ground. Some models make the weaker idealization (2) that DPs assume that the content of each utterance is automatically added to the common ground unless there is evidence to the contrary. Clark and Schaefer argue against these idealizations and propose to replace them with “systematic procedures” for establishing mutual belief regarding the addressee’s understanding of utterances.

Following Clark, more recent computational models of grounding (e.g. Traum (1994) and Ginzburg (forth)) take it for granted that utterances are not taken to be grounded until some

form of feedback has been produced. This feedback need not be explicit: for example, a relevant answer to a question shows that the DP producing the answer has understood and accepted the question posed in the previous utterance. That is, DPs do not make assumptions about grounding until there is some evidence.

However, as noted by Traum this cannot apply to all utterances. If it did, each confirmation of understanding would again have to be confirmed and so on *ad infinitum*. In our terminology, this means that it is necessary to assume optimism on some level. In Traum’s model, acknowledgements are optimistically assumed to be grounded (that is, they do not need to be acknowledged themselves before being added to the common ground) whereas any other conversational acts must receive some acknowledgement before being grounded.

While we believe that Clark is correct in criticizing tacit idealizations about DPs assumptions regarding grounding, we also believe that these tacit assumptions, if made in an explicit and systematic fashion, are not necessarily incorrect or more idealizing than Clark’s alternative. Furthermore, they deserve to be explored both theoretically and for practical use in dialogue systems. We should also keep open the possibility that DPs may make different assumptions regarding grounding depending on various factors of the context.

If issues of tacitness are put aside, it seems that what we are dealing with are different accounts of when an utterance is to be regarded as grounded. The need for such an assumption arises partly because the communicative behaviour itself does not completely determine whether an utterance is grounded. At some point, a DP must simply assume that an utterance has been grounded, and we believe that the main difference between them can be described in terms of *optimism* and *pessimism*.

The first type of “tacit idealization” described by Clark ((1) above) can thus be restated as an optimistic grounding assumption; a DP adhering to this assumption will assume, for any utterance *u*, that *u* is grounded as soon as it has been uttered, with no regard to feedback. The second type of “idealization” ((2) above) can be restated as a pessimistic grounding assumption, since it requires some way of determining the absence of negative feedback before grounding is assumed. Clark’s suggested assumption is also pessimistic, since DPs adhering to it will require positive evidence before assuming that an utterance is grounded.

2.5.2 Grounding updates and action levels

From an information state update perspective, it seems sensible to regard an utterance as grounded when it has been added to the common ground. Each action level connected to an utterance can be associated with a certain type of update. To assume grounding on the perception level can be seen as updating the common ground with the assumed surface form of the utterance; to assume grounding on the understanding level is to update the common ground with a semantic repre-

sensation of the utterance. Finally, to assume an utterance has been grounded on the acceptance level is to update the common ground with the intended effects of the utterance (e.g. pushing a question on QUD). Thus, the grounding assumption can be divided into four independent assumptions, one for each of these levels; we will concentrate on the understanding and integration levels.

The independence of these assumptions means e.g. that it is possible to make an optimistic assumption about understanding but a pessimistic one about acceptance. This would mean assuming that an utterance was understood as soon as it was uttered, but requiring positive evidence before it is assumed to be accepted.

2.5.3 The cautious strategy

Clark seems to assume that once an utterance has been grounded, there is no turning back; the grounding assumption cannot be undone. That is, the moment information about an utterance is added to the common ground there is no way (short of general strategies for belief revision) of understanding negative feedback and reacting to it by modifying or removing the grounded material.

However, we believe that there is a difference between assuming an utterance as grounded (added to the common ground) and giving up the possibility of modifying or correcting the grounded material. This opens up a new kind of grounding strategy not included in Clark's account: the cautious strategy.

For a DP using a cautious strategy, it is possible to assume an utterance as being grounded, while still being able to understand and react appropriately to negative feedback. This requires (1) that negative feedback, which is often underspecified in the sense that it does not explicitly identify which part of an utterance it concerns, can be correctly interpreted, and (2) that the DP can revise the common ground in a way which undoes all effects of the erroneous assumption that the utterance was grounded. A simple example is shown in (12).

- (12) A : Do I need a visa?
A optimistically assumes that "does A need a visa?" is now under discussion.
B : Pardon?
A correctly interprets B's utterance as negative feedback (probably on the perception level) regarding the previous utterance, and retracts the assumption that "does A need a visa?" is on QUD.

On this view, the updates associated with grounding involves two steps: adding the material to

the common ground, and consequently, removing the possibility of (easily) undoing the updates from the first step.

One advantage of the cautious strategy is that inferences resulting from an utterance can be drawn immediately, without having to wait for feedback, while not requiring costly strategies for general belief revision in cases where the grounding assumption turns out to be premature.

We leave open the question of which strategy is the most cognitively plausible; in fact, the most reasonable assumption is probably that an intelligent combination of different strategies is the most realistic model. But we do believe that the cautious strategy deserves the same attention as the pessimistic strategy advocated by Clark. Moreover, we do not want to preclude the possibility that even the optimistic strategy might come in handy sometimes. As always, questions of cognitive plausibility are best resolved by empirical experimentation rather than by rhetoric. What we want to do here, apart from implementing useful grounding mechanisms for a dialogue system, is to show that the cautious approach is at least a possible alternative.

To repeat, we will use the qualification “optimistic”, “cautious” and “pessimistic” for grounding update strategies, with the following meanings:

- optimistic grounding update: DGB⁹ is updated immediately after u was produced (and, in the case of utterances produced by other DP, understood and accepted); no backtracking mechanism available
- cautious grounding update: DGB is updated immediately after u was produced (and, in the case of utterances produced by other DP, understood and accepted); however, backtracking is available
- pessimistic grounding update: DGB is updated when positive evidence of grounding have been acquired

2.6 Feedback and grounding strategies for GODIS

In the previous sections, we discussed grounding-related ICM (and in particular feedback) and grounding strategies in human-human dialogue. In this section, we discuss feedback and grounding from the perspective of dialogue systems in general, and GODIS in particular.

Most (if not all) dialogue systems today have an asymmetrical treatment of grounding, i.e. the grounding of system utterances is handled very differently from the grounding of user utterances.

⁹Dialogue Gameboard, i.e. the SHARED part of the information state in GODIS. See Larsson *et al.* (2002).

Typically, the system will provide fairly elaborate feedback on user input, usually in the form of questions such as “Did you say you want to go to Paris?”. The user must then answer these questions affirmatively before the system will go on. The reason for this, of course, is the low quality of speech recognition.

2.6.1 Grounding strategies for dialogue systems

In this section, we discuss grounding update strategies from the viewpoint of their usefulness in dialogue systems. The two main factors determining the usefulness of an update strategy in a dialogue system is usability (including efficiency of dialogue interaction) and the efficiency of internal processing. On this view, pessimism has the disadvantage that it makes dialogue less efficient since each utterance must be explicitly grounded and accepted; for user utterances this is often achieved by asking check questions to which the user must reply before communication can proceed.

However, the cautiously optimistic approach has the disadvantage that revision is necessary when grounding or acceptance fails, which happens if the user responds negatively to the feedback. The solution presented solves the revision problem by keeping relevant parts of previous information states around.

A common method is to use the recognition score of a user utterance to determine the feedback behaviour from the system, given that the system has understood the utterance sufficiently. We believe that the best solution for an experimental speech-to-speech dialogue system is to switch between grounding update strategies depending on the reliability of communication (which depends on noisiness of environment, previous ratio of successful vs. unsuccessful communication, etc). We link pessimistic and optimistic grounding update to interrogative and positive feedback, respectively. Interrogative feedback from the system raises a question regarding the meaning of a previous utterance, which would not make sense if the system had already assumed that a certain answer to the meaning question had been grounded; in effect, this would amount to raising a question whose answer is (already) assumed to be part of the common ground. Similarly, giving positive feedback corresponds naturally to the case where some interpretation is deemed to be already grounded.

A more sophisticated method for determining what grounding and feedback strategy to use would also take into consideration the degree of relevance of a certain interpretation in the current dialogue context. If a recognition hypothesis which does not have the highest score is nevertheless more relevant than the hypothesis with the highest score, this could result in choosing the former hypothesis. This has not been implemented in GODiS, and is an area for future research.¹⁰

¹⁰For example, one could assess the degree of relevance of a certain *answer-move* by checking how many accommodation steps (see Chapter 3) would be necessary before integrating the question.

2.6.2 “Implicit” and “explicit” verification in dialogue systems

In the literature concerning practical dialogue systems (e.g. San-Segundo *et al.*, 2001), grounding is often reduced to verification of the system’s recognition of user utterances. Two common ways of handling verification are described as “explicit” and “implicit” verification, exemplified in (13) (example from San-Segundo *et al.*, 2001).

- (13) a. I understood you want to depart from Madrid. Is that correct? [explicit]
b. You leave from Madrid. Where are you arriving at? [implicit]

Actually, both “explicit” and “implicit” feedback contain a verbatim repetition or a reformulation of the original utterance, and in this sense they are both explicit. The actual base for the distinction is what we have here referred to as polarity: “explicit” verification is neutral (and eliciting and interrogative) whereas “implicit” verification is positive.

In human-human dialogue, explicit confirmations occur in noisy environments and in situations where understanding is critical (e.g. when arranging a meeting in a busy airport). Given that verification is presumably a rather marginal phenomena in human-human dialogue, it is perhaps surprising that it is often the only aspect of feedback covered in dialogue systems literature. Firstly, because it is usually not necessary for humans to verify what they (think they) have heard; that is, it is a rather uncommon grounding procedure in human-human dialogue. Second, because it only involves part of the full spectrum of feedback behaviour, excluding e.g. acceptance-related feedback behaviour.

Of course, verification of user utterances are of central importance in dialogue systems, given the quality of current speaker-independent speech recognition. This explains to some extent why verification is often the only aspect of feedback handled by current systems - it is simply necessary. However, this is no reason not to explore further the possible uses of a wider range of feedback behaviour in dialogue systems.

2.6.3 Issue-based grounding in GODIS

In this section we outline a (partially) issue-based account of grounding in terms of information state updates, inspired by Ginzburg’s account of content questions and acceptance-questions. However, we make significant departures from Ginzburg’s account, for various reasons.

A basic idea of the account used in GODIS2 is that meta-issues (the content and acceptance

questions) do not always have to be represented explicitly. However, in certain cases it is useful to represent grounding issues explicitly.

Content questions in GODIS

We regard explicit interrogative understanding feedback as explicitly raising content questions, which may be responded to explicitly or implicitly. We also refer to these as *understanding questions*. Explicit interrogative feedback is very relevant for dialogue systems, where poor speech recognition often makes it necessary for the system to explicitly verify each recognized user utterance, giving the user a chance to correct any misunderstandings.

Interrogative feedback can in principle be *wh*-questions (“What do you mean?”), *y/n*-questions (“Do you mean Paris?”, “Paris, is that correct?”), or alternative questions (“Do you mean to Paris or from Paris?”). However, we have chosen negative feedback (“I don’t understand”) rather than *y/n*-questions to indicate lack of understanding. Clarification questions are not used in GODIS2; however, they will be introduced in Chapter 3. This leaves us with *y/n* understanding questions, which concern one specific interpretation of a previous utterance. These are represented in GODIS2 as **?und(DP*C)** where *DP* is a DP and *C* is a proposition, and can be paraphrased as “Did *DP* mean *C*?” or “Is *C* a correct understanding of *DP*’s utterance?”. In the case where the understanding question concerns a question (raised by an **ask**-move), the proposition *C* is **issue(Q)** where *Q* is a question. In this case, the paraphrase can be further specified as “Did *DP* mean to raise *Q*?”.

To allow this, we have extended the GODIS semantics presented in Larsson *et al.* (2002) to include two new kinds of propositions.

- **und(DP, P)** : Proposition where *P* : Proposition and *DP* : Participant¹¹
- **issue(Q)** : Proposition where *Q* : Question¹²

Implicit understanding-questions Actually, the use of the term “implicit” in the context of grounding (or verification) can be used to describe not the grounding behaviour itself but, rather,

¹¹Note that this definition allows *P* to itself be a proposition of the form **und(DP, P)**; however, we allow this to pass in the interest of brevity.

¹²This definition allows **issue(Q)** as a proposition even when it is not embedded in a proposition **und(DP, issue(Q))**. In GODIS, the proposition **issue(Q)** only appears inside understanding questions or as an argument to an ICM move (see Section 2.6.5). However, in Chapter 3 we will use the corresponding question **?issue(Q)**. A suitable paraphrase of this question would be “Should *Q* become an issue?” or “Should *Q* be opened for discussion”; this is similar to Ginzburg’s MAX-QUD question.

the status of the grounding issue. What is often referred to as implicit verification can arguably be seen as implicitly raising a grounding question, which may or may not be responded to.

This idea will not be implemented until Chapter 3, since it requires some additional mechanisms which will be needed anyway for the kind of behaviours we introduce there. Specifically, we need a distinction between a global and a local QUD (see also Cooper *et al.* (2000) and Cooper and Larsson (2002)), where the former contains explicitly raised or addressed (but as yet unresolved) issues, and the latter contains questions which can be used for resolving short answers.

To give a short preview, the basic idea is that explicit positive feedback *implicitly* raises an understanding-issue, i.e. when the implicit feedback is integrated, the understanding question is pushed on local QUD. This allows the user to discard the system's interpretation simply by providing a negative answer to the grounding question, or confirm it by giving a positive answer. Since the question is added only to the local QUD, and not to the global one, it will eventually disappear if it is not answered. This allows dialogues to proceed more efficiently, since the user does not have to give explicit confirmations all the time. Again, this will be explained in detail in Chapter 3.

Acceptance questions

In Ginzburg's protocol, a DP who has perceived and interpreted an utterance is faced with the acceptance-question; whether to accept the content of the utterance or not. If the content is not accepted, the DP should push the integrate question (push it on QUD) and address it.

One way of dealing with acceptance would be to follow Ginzburg's account and explicitly represent an acceptance-question which is pushed on QUD in cases where a user utterance is understood but cannot be integrated, and subsequently produce an utterance addressing the acceptance question. We regard feedback-moves on the reaction level (compliance and declination moves, in Clark's terminology) as addressing acceptance questions. Above, we have argued against pushing anything on QUD in this case since it is a shared structure, and the user in this case has no chance of doing the corresponding update on her own QUD until the utterance has been produced. So an alternative strategy would be to first produce the utterance addressing the acceptance question and subsequently assume that the user will accommodate it and push it on QUD; at this time, the system can do the same.

However, it appears that it is only useful to represent the acceptance question explicitly on QUD in cases where it can give rise to a discussion where DPs argue for and against the acceptance of a question as a topic for discussion, or for a proposition as a fact or as a topic for discussion. For a dialogue system unable to perform such argumentation dialogues, it appears pointless to represent the acceptance question explicitly. Since an acceptance or rejection move cannot be challenged, the move will provide a definite answer to the integration question which would thus

be immediately removed from QUD once the rejection had been grounded.

For these reasons, we will not represent acceptance issues explicitly in GODIS. In a system capable of negotiation and/or argumentation, however, it would be necessary to do so, and to regard feedback-moves on the acceptance level as relevant answers to this question (see Section 4.8.2 for further discussion).

Temporary storage

To enable cautious grounding we need some way of revising the dialogue gameboard when optimistic grounding assumptions turn out to be premature, without having to deal with the problems of generalized belief revision Gärdenfors (1988). A straightforward way of doing this is to keep around relevant parts of previous dialogue gameboard states, and copy the contents of these back to the DGB when necessary. This strategy will be used for system utterances in GODIS2, and also for user utterances in GODIS3.

2.6.4 Enhancing the information state to handle feedback

In this section, we show how the GODIS information state needs to be modified to handle grounding and feedback. The new information state type is shown in Figure 2.1.

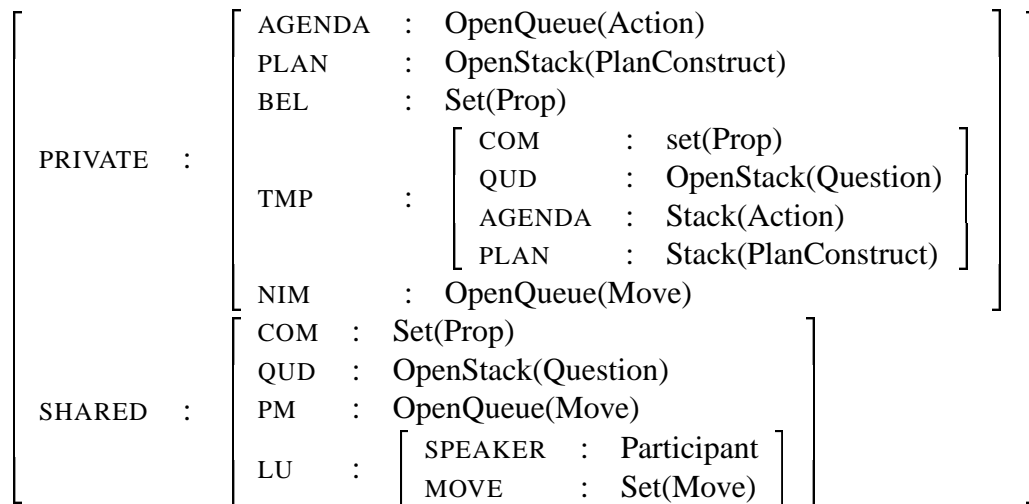


Figure 2.1: GODIS2 Information State type

Temporary store

To enable the system to backtrack if an optimistic assumption turns out to be mistaken, relevant parts of the information state is kept in `/PRIVATE/TMP`. The QUD and COM fields may change when integrating an `ask` or `answer` move, respectively. The plan may also be modified, e.g. if a `raise` action is selected. Finally, if any actions are on the agenda when selection starts (which means they were put there during by the update module), these may have been removed during the move selection process.

Non-integrated moves

Since several moves can be performed per turn, GODiS needs some way of keeping track of which moves have been interpreted. This is done by putting all moves in `LATEST_MOVES` in a queue structure called `NIM`, for Non-Integrated Moves. This structure is private, since it is an internal matter for the system how many moves have been integrated so far. Once a move is assumed to be grounded on the understanding level the move is added to the `/SHARED/LU/MOVES` set. Since the move has now been understood on the pragmatic level, the content of the move will be a question or a full proposition (for short answers, the proposition resulting from combining it with a question on QUD).

Previous moves

To be able to detect irrelevant followups, GODiS needs to know what moves were performed (and grounded) in the previous utterance. These are stored in the `/SHARED/PM` field.

Timeout

To be able to decide when the user has given up her turn, we have added a TIS variable `TIMEOUT` of type `Real`, whose value is the time (in seconds) after which the system will assume that the turn has been given up if no speech has been detected. This variable will be further discussed in Section 2.6.6.

2.6.5 Feedback and sequencing dialogue moves

In this section, we first show how feedback dialogue moves in GODIS2 are represented. We then review the full range of feedback moves, starting with system-generated feedback and then moving on to user feedback.

The general notation for ICM dialogue moves used in GODIS is the following:

$$(14) \text{ icm:}L^*P\{:\textit{Args}\}$$

where L is an action level, P is a polarity, and \textit{Args} are arguments.

- L : action level
 - con: contact (“Are you there?”)
 - per: perception (“I didn’t hear anything from you”, “I heard you say ’to Paris’”)
 - sem: semantic understanding (“I don’t understand”, “To Paris.”)
 - und: pragmatic understanding (“I don’t quite understand”, “You want to know about price.”)
 - acc: acceptance/reaction (“Sorry, I can’t answer questions about connecting flights”, “Okay.”)
- P : polarity
 - neg: negative
 - int: interrogative
 - pos: positive
- \textit{Args} : arguments

Note that the “neutral” polarity has been replaced by the label “int”; we have made a simplifying assumption that neutral feedback is always eliciting and interrogative.¹³

The arguments are different aspects of the utterance or move which is being grounded, depending on action level:

¹³Note, however, that if we had included feedback forms like “What did you say?”, this would still be regarded as negative feedback. The ‘int’ label only refers to check-questions, which are usually *y/n*-questions. This is arguably not an optimal labelling convention.

- for per-level: *String*, the recognized string
- for sem-level: *Move*, a move interpreted from the utterance
- for und-level: $DP * C$, where
 - DP : Participant is the DP who performed the utterance
 - C : Proposition is the propositional content of the utterance
- for acc-level: C : Proposition, the content of the utterance

For example, the ICM move `icm:und*pos:usr*dest.city(paris)` provides positive feedback regarding a user utterance that has been understood as meaning that the user wants to go to Paris.

In addition, sequencing ICM moves for indicating reraising of issues and loading a plan are included:

- `icm:reraise`: indicate reraising implicitly (“So, ...”)
- `icm:reraise:Q`: reraising an issue Q explicitly (“Returning to the issue of Price.”)
- `icm:loadplan` (“Let’s see.”)

System feedback to user utterances in GODIS2

In this section and the following section, we review surface forms related to feedback and other ICM behaviour that will be implemented in GODIS2.

For user utterances, GODIS2 will be able to produce e.g. the following kinds of feedback utterances (for the examples, assume that the user just said “I want to go to Paris”):

- contact
 - negative; `icm:con*neg` (“I didn’t hear anything from you”)
- perception
 - negative; `icm:per*neg` realized as conventional feedback phrase or a declarative sentence (“Pardon?”, “I didn’t hear what you said.”)
 - positive; `icm:per*pos:String` realized as metalevel verbatim repetition (“I heard ‘to paris’ ”)

- understanding (semantic)
 - negative; *icm:sem*neg* realized as fb-phrase (“I don’t understand.”)
 - positive; *icm:sem*pos:Content* realized as repetition/reformulation of content (object-level) (“Paris.”)
- understanding (pragmatic)
 - negative; *icm:und*neg* realized as fb-phrase (“I don’t quite understand.”)
 - positive; *icm:und*pos:DP*Content* realized as repetition/reformulation of content (object-level) (“To Paris.”)
 - interrogative; *icm:und*int:DP*Content* realized as ask about interpretation (“To Paris, is that correct?”)
- integration
 - negative
 - * proposition-rejection; *icm:acc*neg:Content* realized as explanation (“Sorry, Paris is not a valid destination city”)
 - positive; *icm:acc*pos* realized as fb-word (“Okay”)

In addition, GODIS2 will be able to perform issue-rejection using the move *icm:acc*neg:issue(Q)*, where *Q* : Question as illustrated in (DIALOGUE 2.2).

(DIALOGUE 2.2)

U> What about connecting flights?

S> Sorry, I cannot answer questions about connecting flights.

We are not claiming that humans always make these distinctions between action explicitly or even consciously, nor that the link between surface form and feedback type is a simple one-to-one correspondence; for example, “mm” may be used as positive feedback on the perception, understanding, and acceptance levels. Feedback is, simply, often ambiguous. However, since GODIS is making all these distinctions internally we might as well try to produce feedback which is not so ambiguous. Of course, there is also a tradeoff in relation to brevity; extremely explicit feedback (e.g. “I understood that you referred to Paris, but I don’t see how that is relevant right now.”) could be irritating and might decrease the efficiency of the dialogue. We feel that the current choices of surface forms are fairly reasonable, but testing and evaluation on real users would be needed to find the best ways to formulate feedback on different levels. This is an area for future research.

A general strategy used by GODIS in ICM selection is that if negative or interrogative feedback on some level is provided, the system should also provide positive feedback on the level below. For example, if the system produces negative feedback on the pragmatic understanding level, it should also produce positive feedback on the semantic understanding level.

(15) S> Paris. I don't quite understand.

In some systems, positive or interrogative feedback to user utterances is not given immediately; instead, the system repeats all the information it has received just before making a database query and asks the user if it is correct. It is also possible to combine feedback after each utterance with a final confirmation. In GODIS2, we have not implemented final confirmations. It can be argued that final confirmations are more important in action-oriented dialogue (see Chapter 4), whereas they are not so important in inquiry-oriented dialogue since they never result in any actions other than database searches.

User feedback to system utterances in GODIS2

For system utterances, GODIS2 will react appropriately to the following types of user feedback:

- perception level
 - negative; fb-phrase (“Pardon?”, “Excuse me?”, “Sorry, I didn’t hear you”) interpreted as icm:per*neg
- reaction/acceptance level
 - positive; fb-phrase (“Okay.”) interpreted as icm:acc*pos
 - negative; issue rejection fb-phrase (“I don’t know”, “Never mind”, “It doesn’t matter”) interpreted as icm:acc*neg:issue

In addition, irrelevant followups to system **ask**-moves are regarded as implicit issue-rejections.

The coverage of user feedback behaviour is thus more limited than the coverage for system behaviour. The main motivation for this is that system utterances are less likely to be problematic for the user to interpret than vice versa.

Still, the available coverage allows some useful feedback-phrases, including negative perception feedback which is useful if the output from the system’s speech synthesizer is of poor quality.

Ideally, this would provide a slight reformulation by the system, but since generation is not a main topic here, this has not been implemented.

Understanding-level feedback has not been included but may be useful in cases where the user hears the system but cannot understand the meaning of the words uttered by the system. In this case, a reformulation by the system may again be useful.

2.6.6 Grounding of user utterances in GODIS2

In this section we show how optimistic and pessimistic grounding of user utterances has been integrated in GODIS2. First we show how grounding strategies are dynamically selected depending on recognition score, in the case where a move has been fully understood and accepted. Next, we show how to deal with system responses to interrogative feedback associated with pessimistic grounding. Finally, we show how the system deals with failure to perceive, understand, and integrate user utterances by giving negative feedback on the appropriate action level.

Dynamic selection of grounding strategies for user moves

For user utterances, GODIS2 uses optimistic or pessimistic grounding strategies based on the recognition score and the dialogue move type. This makes the corresponding integration rules more complex than the ones in GODIS1. For user “core” moves (in GODIS2, **ask** and **answer**), the integration strategy depends on the recognition score for the utterance in question. This choice is determined by two recognition thresholds, T_1 and T_2 , where $T_1 > T_2$. If the recognition score is higher than T_2 , an optimistic strategy is chosen; positive acceptance feedback (“OK”) is selected, and if the score is lower than T_1 positive understanding feedback (“To Paris.”) is also selected.

If the score is lower than T_2 , the move is not integrated and in the selection phase a pessimistic strategy involving interrogative understanding feedback (e.g. “To Paris, is that correct?”) is selected (see Section 2.6.6).

Of course, the idea of using recognition score for determining whether and how to confirm user utterances is not new (see e.g. San-Segundo *et al.*, 2001), and more sophisticated decision procedures are certainly possible. We use it here to show how GODIS2 enables flexible choice both of feedback type and of grounding update strategy.

In addition to being checked for relevance, contentful moves are checked for integratability (acceptability) and if these conditions are not fulfilled the move will not be integrated; instead, it will give rise to negative acceptance feedback as explained in Section 2.6.6.

Integration of user ask move The integration rule for user **ask** move implementing the optimistic grounding strategy is shown in (RULE 2.1).

(RULE 2.1) RULE: **integrateUsrAsk**
 CLASS: **integrate**
 PRE: {
 \$/SHARED/LU/SPEAKER==usr
 fst(\$/PRIVATE/NIM, ask(*Q*))
 \$SCORE=*Score*
 $Score > 0.7$
 \$DOMAIN :: plan(*Q*, *Plan*)
 EFF: {
 1 pop(/PRIVATE/NIM)
 2 push(/PRIVATE/AGENDA, icm:acc*pos)
 3 add(/SHARED/LU/MOVES, ask(*Q*))
 4 if_do($Score \leq 0.9$,
 push(/PRIVATE/AGENDA, icm:und*pos:usr*issue(*Q*)))
 5 if_do(in(\$/SHARED/QUD, *Q*) and not fst(\$/SHARED/QUD, *Q*),
 push(/PRIVATE/AGENDA, icm:reraise:*Q*))
 6 push(/SHARED/QUD, *Q*)
 7 push(/PRIVATE/AGENDA, respond(*Q*))

The first two conditions pick out a user **ask** move on NIM. The third and fourth conditions check the recognition score of the utterance and if it is higher than 0.7 (T_2), the rule proceeds to check for acceptability. If the score is too low, the move should not be optimistically integrated; instead, a pessimistic grounding strategy should be applied and interrogative feedback selected (see below).

The fifth condition checks for acceptability, i.e. that the system is able to deal with this question, i.e. that there is a corresponding plan in the domain resource. If not, the integration rule will not trigger and the **ask** move will remain on NIM until the selection phase, where it will give rise to an issue rejection (see Section 2.6.6).

The first update pops the integrated move off NIM. In update 2, positive integration feedback is added to the agenda, to indicate that the system can integrate the **ask**-move. Update 3 adds the move to /SHARED/LU/MOVES, thereby reflecting the optimistic grounding assumption on the understanding level. In update 4, positive understanding feedback is selected unless the score is higher than 0.9 (T_1).

Update 5 checks if this question is already on QUD; if so, the system selects sequencing feedback to show that it has understood that the user is reraising an open issue. (If the question is already on top of QUD, however, it is not seen as a case of reraising.) See Section 2.6.9 for more cases of reraising. Update 6 pushes *Q* on QUD; note that if *Q* was already on QUD but not topmost, pushing it will be equivalent to raising it to the topmost position. This is a property of the

OpenStack datatype (see SIRIDUS (2002)).

Update 7 pushes the action to respond to Q on the agenda. This can be regarded as a shortcut for reasoning about obligations and intentions; when accepting a user question, thus accepting the obligation to try to respond to it, the system will automatically intend to respond to it.

Default ICM move selection rule The role of the ICM move selection rules is to add moves to be generated to the NEXT_MOVES TIS variable based on the contents of the agenda. ICM which is added to the agenda by the update module will be moved to NEXT_MOVES by the default ICM selection rule (RULE 2.2).

```
(RULE 2.2)  RULE: selectIcmOther
            CLASS: select_icm
            PRE: { in(/PRIVATE/AGENDA, icm:A)
                  { not in($NEXT_MOVES, B) and B=ask(C)
            EFF: { push(NEXT_MOVES, icm:A)
                  { del(/PRIVATE/AGENDA, icm:A)
```

Dialogue example: integrating user ask-move The dialogue below shows how a user ask move with a score of 0.76 is successfully integrated, and positive understanding and acceptance feedback is produced.

(DIALOGUE 2.3)

S> Welcome to the travel agency!

U> price information please [0.76]

getLatestMoves

```
{ set(/PRIVATE/NIM, oqueue([ask(?A.price(A))]))
  set(/SHARED/LU/SPEAKER, usr)
  clear(/SHARED/LU/MOVES)
  set(/SHARED/PM, set([greet]))
```

integrateUsrAsk

```
{ pop(/PRIVATE/NIM)
  push(/PRIVATE/AGENDA, icm:acc*pos)
  add(/SHARED/LU/MOVES, ask(?A.price(A)))
  if_do( $0.76 \leq 0.9$ , push(/PRIVATE/AGENDA, icm:und*pos:usr*issue(?A.price(A))))
  if_do(in(/SHARED/QUD, ?A.price(A)) and not fst(/SHARED/QUD, ?A.price(A)),
    push(/PRIVATE/AGENDA, icm:reraise:?A.price(A)))
  push(/SHARED/QUD, ?A.price(A))
  push(/PRIVATE/AGENDA, respond(?A.price(A)))
```

findPlan

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \left\langle \left\langle \begin{array}{l} \text{icm:acc*pos} \\ \text{icm:und*pos:usr*issue(?A.price(A))} \\ \text{icm:loadplan} \end{array} \right\rangle \right\rangle \\ \text{NIM} = \langle \rangle \\ \text{COM} = \{ \} \\ \text{QUD} = \{ ?A.price(A) \} \\ \text{PM} = \{ \text{greet} \} \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{usr} \\ \text{MOVES} = \{ \text{ask(?A.price(A))} \} \end{array} \right] \end{array} \right] \right]$$

backupShared

selectFromPlan

selectIcmOther

```
{ push(NEXT_MOVES, icm:acc*pos)
  del(/PRIVATE/AGENDA, icm:acc*pos)
```

selectIcmOther

```
{ push(NEXT_MOVES, icm:und*pos:usr*issue(?A.price(A)))
  del(/PRIVATE/AGENDA, icm:und*pos:usr*issue(?A.price(A)))
```

selectIcmOther

selectAsk

S> Okay. You want to know about price. Lets see. How do you want to travel?

Interrogative understanding feedback for user ask move If a user ask move cannot be assumed to be understood because of a low recognition score, interrogative feedback on the understanding level is selected by (RULE 2.3).

(RULE 2.3) RULE: **selectIcmUndIntAsk**

CLASS: **select_icm**

PRE: $\left\{ \begin{array}{l} \$/\text{SHARED}/\text{LU}/\text{SPEAKER} == \text{usr} \\ \text{fst}(\$/\text{PRIVATE}/\text{NIM}, \text{ask}(Q)) \\ \text{\$SCORE} \leq 0.7 \end{array} \right.$

EFF: $\left\{ \begin{array}{l} \text{pop}(/ \text{PRIVATE} / \text{NIM}) \\ \text{push}(\text{NEXT_MOVES}, \text{icm:und*int:usr*issue}(Q)) \end{array} \right.$

The conditions are straightforward. The first update removes the move from NIM, even though it has not been integrated. An alternative approach would be to keep this move on NIM and explicitly represent the grounding as concerning this move. However, this would require labelling all moves with unique move IDs; instead, we follow the general philosophy of GODiS of trying to keep our representation as simple as possible as long as it works. The interrogative feedback selected in the second update will, in a sense, take over the function of the original move; if the

feedback is answered positively, the end result will be the same as if the **ask** move had been integrated immediately (see Section 2.6.6 for further explanation).

Integration of user answer move The integration rule for user **answer** moves, shown in (RULE 2.4) is similar to that for **ask** moves, except that answers are checked for relevance as well as reliability and acceptability.

(RULE 2.4) RULE: **integrateUsrAnswer**
 CLASS: **integrate**

PRE:	{	1 fst(/PRIVATE/NIM, answer(<i>A</i>)) 2 \$/SHARED/LU/SPEAKER==usr 3 ! \$SCORE= <i>Score</i> 4 <i>Score</i> > 0.7 5 fst(/SHARED/QUD, <i>Q</i>) 6 \$DOMAIN :: relevant(<i>A</i> , <i>Q</i>) 7 \$DOMAIN :: combine(<i>Q</i> , <i>A</i> , <i>P</i>) 8 \$DATABASE :: validDBparameter(<i>P</i>) or <i>P</i> =not(<i>X</i>)
EFF:	{	1 pop(/PRIVATE/NIM) 2 add(/SHARED/LU/MOVES, answer(<i>P</i>)) 3 push(/PRIVATE/AGENDA, icm:acc*pos) 4 if_do(<i>Score</i> ≤ 0.9 and <i>A</i> ≠ yes and <i>A</i> ≠ no, push(/PRIVATE/AGENDA, icm:und*pos:usr* <i>P</i>)) 5 add(/SHARED/COM, <i>P</i>)

Conditions 1-4 are similar to those for the **integrateUsrAsk** rule. The relevance of the content of the answer to a question on QUD is checked in condition 6.

The acceptability condition in condition 8 makes sure that the propositional content resulting from combining the question topmost on QUD with the content of the **answer**-move is either

- a valid database parameter, or
- a negated proposition

Negated propositions can always be integrated (as long as they are relevant); for example, it is okay to say that you do not want to go to Paris, even if Paris is not in the database.

Updates 1-3 again correspond closely to those in **integrateUsrAsk**. Update 4 checks if the score was lower than or equal to 0.9; if so, a positive understanding feedback move is selected. If the

score is higher than 0.9 or if the answer is **yes** or **no**, no understanding feedback is produced. The special special status of “yes” and “no” builds on the assumption that these are easily recognized; if this is not the case, their special status should be removed. Finally, update 5 adds the proposition resulting from combining the question on QUD with the content of the **answer** move to the shared commitments.

Interrogative understanding feedback for user ask move If a user **ask** move receives a low score (lower than T_2 , which is here set to 0.7) and the question raised by the move is acceptable to the system, interrogative understanding feedback is selected by (RULE 2.5). (If the question is not acceptable it will instead be rejected; see Section 2.6.6).

```
(RULE 2.5)  RULE: selectIcmUndIntAnswer
            CLASS: select_icm
            PRE: {
                fst(/PRIVATE/NIM, answer(A))
                $/SHARED/LU/SPEAKER==usr
                $SCORE ≤ 0.7
                fst(/SHARED/QUD, B)
                $DOMAIN :: relevant(A, B)
                $DOMAIN :: combine(B, A, C)
            }
            EFF: {
                pop(/PRIVATE/NIM)
                push(NEXT_MOVES, icm:und*int:usr*C)
            }
```

The conditions check that there is a user **answer** move on NIM whose content is **relevant** to and combines with a question on QUD, and that the recognition score was less than or equal to 0.7. If these conditions are true, the move is popped off NIM and interrogative understanding feedback is selected.

Integrating and responding to interrogative feedback

Integrating interrogative understanding feedback As explained in Section 2.6.3, Interrogative feedback raises understanding questions. This is reflected in (RULE 2.6).

```
(RULE 2.6)  RULE: integrateUndIntICM
            CLASS: integrate
            PRE: {
                fst(/PRIVATE/NIM, icm:und*int:DP*C)
            }
            EFF: {
                pop(/PRIVATE/NIM)
                add(/SHARED/LU/MOVES, icm:und*int:DP*C)
                push(/SHARED/QUD, und(DP*C))
            }
```

The condition simply checks that there is an `icm:und*int:DP*C` move on NIM, which is then popped off by the first update and added to `/SHARED/LU/MOVES` by the second update. The third update pushes the understanding question `?und(DP*C)` on QUD.

Integrating positive answer to understanding-question When the system raises an understanding question (e.g. by saying “To Paris, is that correct?”), the user can either say “yes” or “no”. (The case where the user does not give a **relevant** answer to the interrogative feedback is treated in Section 2.6.8). In GODIS2, we do not represent propositions related to the understanding of utterances in the same way as other propositions (which are stored in `/SHARED/COM`). Therefore, special rules are needed for dealing with answers to understanding-questions.

The rule for integrating a negative answer to an understanding-question is shown in (RULE 2.7).

(RULE 2.7) **RULE: integrateNegIcmAnswer**
 CLASS: integrate
 PRE: $\left\{ \begin{array}{l} \text{fst}(\$/\text{PRIVATE}/\text{NIM}, \text{answer}(\text{no})) \\ \text{fst}(\$/\text{SHARED}/\text{QUD}, \text{und}(\text{DP}^*C)) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{pop}(\$/\text{PRIVATE}/\text{NIM}) \\ \text{add}(\$/\text{SHARED}/\text{LU}/\text{MOVES}, \text{answer}(\text{und}(\text{DP}^*C))) \\ \text{pop}(\$/\text{SHARED}/\text{QUD}) \\ \text{push}(\$/\text{PRIVATE}/\text{AGENDA}, \text{icm:und*pos:DP*not}(C)) \end{array} \right.$

The conditions check that there’s an `answer(yes)` move on NIM and an understanding-question on QUD. The first three updates establish the move as shared and pop the understanding-question off QUD. Finally, positive feedback is selected to indicate that the system has understood that the assumed interpretation *C* was incorrect.

Integrating positive answer to understanding question The rule for integrating a positive answer to an understanding-question is shown in (RULE 2.8).

(RULE 2.8) **RULE: integratePosIcmAnswer**
CLASS: integrate
PRE: $\left\{ \begin{array}{l} \text{fst}(\$/\text{PRIVATE}/\text{NIM}, \text{answer}(\text{yes})) \\ \text{fst}(\$/\text{SHARED}/\text{QUD}, \text{und}(DP*Content)) \end{array} \right.$
EFF: $\left\{ \begin{array}{l} \text{pop}(/ \text{PRIVATE}/\text{NIM}) \\ \text{add}(/ \text{SHARED}/\text{LU}/\text{MOVES}, \text{answer}(\text{und}(DP*Content))) \\ \text{pop}(/ \text{SHARED}/\text{QUD}) \\ \text{if_then_else}(Content=\text{issue}(Q), [\\ \quad \text{push}(/ \text{SHARED}/\text{QUD}, Q) \\ \quad \text{push}(/ \text{PRIVATE}/\text{AGENDA}, \text{respond}(Q))], \\ \text{add}(/ \text{SHARED}/\text{COM}, Content)) \end{array} \right.$

The conditions and the first three updates are similar to those in the **integrateNegIcmAnswer** rule. The final (conditionalized) update integrates the content C . If the “original” move (the move which caused the interrogative feedback to be produced in the first place) was **ask**, C will be a proposition **issue**(Q). Consequently, integrating this propositions has the same effects as integrating an **ask**-move: pushing the question Q on QUD and pushing the action **respond**(Q) on the agenda. If the proposition is not of this type, it is simply added to /SHARED/COM.

Dialogue example: positive and negative response to interrogative feedback In the following dialogue, the system produces interrogative understanding feedback for two user utterances, one containing an **ask** move and the other containing an **answer** move. The first interrogative feedback is answered positively and the second negatively.

(DIALOGUE 2.4)

U> price information please [0.65]

getLatestMoves

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \langle \rangle \\ \text{PLAN} = \langle \rangle \\ \text{NIM} = \langle \langle \text{ask}(?A.\text{price}(A)) \rangle \rangle \\ \text{COM} = \{ \} \\ \text{QUD} = \langle \rangle \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{usr} \\ \text{MOVES} = \{ \} \end{array} \right] \end{array} \right] \right]$$

backupShared

selectIcmUndIntAsk

$$\left\{ \begin{array}{l} \text{pop}(/ \text{PRIVATE}/\text{NIM}) \\ \text{push}(/ \text{PRIVATE}/\text{AGENDA}, \text{icm:und*int:usr*issue}(\text{?}A.\text{price}(A))) \end{array} \right.$$

selectIcmOther

```

{
  push(NEXT_MOVES, icm:und*int:usr*issue(?A.price(A)))
  del(/PRIVATE/AGENDA, icm:und*int:usr*issue(?A.price(A)))
}

```

S> You want to know about price , is that correct?

getLatestMoves

integrateUndIntICM

```

{
  pop(/PRIVATE/NIM)
  add(/SHARED/LU/MOVES, icm:und*int:usr*issue(?A.price(A)))
  push(/SHARED/QUD, und(usr*issue(?A.price(A))))
}

```

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \langle \rangle \\ \text{PLAN} = \langle \rangle \\ \text{NIM} = \langle \rangle \\ \text{COM} = \{ \} \\ \text{QUD} = \left\langle \text{und(usr*issue(?A.price(A)))} \right\rangle \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{sys} \\ \text{MOVES} = \{ \text{icm:und*int:usr*issue(?A.price(A))} \} \end{array} \right] \end{array} \right] \right]$$

U> yes

getLatestMoves

integratePosIcmAnswer

```

{
  pop(/PRIVATE/NIM)
  add(/SHARED/LU/MOVES, answer(und(usr*issue(?A.price(A)))))
  pop(/SHARED/QUD)
  if_then_else(issue(?A.price(A))=issue(B), [
    push(/SHARED/QUD, B)
    push(/PRIVATE/AGENDA, respond(B)) ],
  add(/SHARED/COM, issue(?A.price(A))))
}

```

findPlan

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \langle \langle \text{icm:loadplan} \rangle \rangle \\ \text{PLAN} = \left\langle \begin{array}{l} \text{findout(?A.how(A))} \\ \text{findout(?B.dest_city(B))} \\ \text{findout(?C.dept_city(C))} \\ \text{findout(?D.month(D))} \\ \text{findout(?E.dept_day(E))} \\ \text{findout(?F.class(F))} \\ \text{consultDB(?G.price(G))} \end{array} \right\rangle \\ \text{NIM} = \langle \rangle \\ \text{COM} = \{ \} \\ \text{QUD} = \langle ?H.price(H) \rangle \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{usr} \\ \text{MOVES} = \{ \text{answer(und(usr*issue(?H.price(H))))} \} \end{array} \right] \end{array} \right] \right]$$

backupShared

selectFromPlan

selectIcmOther

selectAsk

S> Lets see. How do you want to travel?

getLatestMoves
integrateOtherICM
integrateSysAsk

U> by plane [0.56] (*user actually said "by train"*)

getLatestMoves
backupShared
selectIcmUndIntAnswer
{ pop(/PRIVATE/NIM)
 push(/PRIVATE/AGENDA, icm:und*int:usr*how(plane))
selectIcmOther

S> by flight , is that correct?

getLatestMoves
integrateUndIntICM
{ pop(/PRIVATE/NIM)
 add(/SHARED/LU/MOVES, icm:und*int:usr*how(plane))
 push(/SHARED/QUD, und(usr*how(plane)))

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \langle \rangle \\ \text{PLAN} = \left\langle \begin{array}{l} \text{findout}(?A.\text{how}(A)) \\ \text{findout}(?B.\text{dest_city}(B)) \\ \text{findout}(?C.\text{dept_city}(C)) \\ \text{findout}(?D.\text{month}(D)) \\ \text{findout}(?E.\text{dept_day}(E)) \\ \text{findout}(?F.\text{class}(F)) \\ \text{consultDB}(?G.\text{price}(G)) \end{array} \right\rangle \\ \text{NIM} = \langle \rangle \\ \text{COM} = \{ \} \\ \text{QUD} = \left\langle \begin{array}{l} \text{und}(\text{usr}*\text{how}(\text{plane})) \\ ?H.\text{how}(H) \\ ?I.\text{price}(I) \end{array} \right\rangle \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{sys} \\ \text{MOVES} = \{ \text{icm:und*int:usr*how(plane)} \} \end{array} \right] \end{array} \right]$$

U> no

getLatestMoves
integrateNegIcmAnswer


```

{
  pop(/PRIVATE/NIM)
  add(/SHARED/LU/MOVES, answer(und(usr*how(plane))))
  pop(/SHARED/QUD)
  push(/PRIVATE/AGENDA, icm:und*pos:usr*not(how(plane)))
}

```

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \langle \langle \text{icm:und*pos:usr*not(how(plane))} \rangle \rangle \\ \text{PLAN} = \left\langle \begin{array}{l} \text{findout(?A.how(A))} \\ \text{findout(?B.dest_city(B))} \\ \text{findout(?C.dept_city(C))} \\ \text{findout(?D.month(D))} \\ \text{findout(?E.dept_day(E))} \\ \text{findout(?F.class(F))} \\ \text{consultDB(?G.price(G))} \end{array} \right\rangle \\ \text{NIM} = \langle \rangle \\ \text{COM} = \{ \} \\ \text{QUD} = \left\langle \begin{array}{l} ?H.how(H) \\ ?I.price(I) \end{array} \right\rangle \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{usr} \\ \text{MOVES} = \{ \text{answer(und(usr*how(plane)))} \} \end{array} \right] \end{array} \right]$$

backupShared

reraiseIssue

selectIcmOther

```

{
  push(NEXT_MOVES, icm:und*pos:usr*not(how(plane)))
  del(/PRIVATE/AGENDA, icm:und*pos:usr*not(how(plane)))
}

```

selectIcmOther

selectAsk

S> not by flight. So, How do you want to travel?

Negative contact and perception level feedback

What happens if no system utterance is detected, or if the speech recognizer fails? Most speech recognizers can tell the difference between not hearing anything at all, and hearing something but not being able to come up with any hypothesis regarding what was said. We will use this distinction to enable GODIS to produce feedback on the contact and perception levels.

If GODIS does not receive any input within a certain time-frame (specified by the TIMEOUT TIS variable), it will produce feedback indicating that nothing was perceived, e.g. “I didn’t hear anything from you.”. We classify this as negative feedback on the contact level. It could perhaps be argued that the distinction between contact and perception level feedback is not very sharp, and that this kind of feedback actually concerns the perception level. However, it is possible that the reason that nothing was registered by the recognizer was a failure to establish a channel of communication from the user to the system, e.g. if a microphone is broken or not plugged in properly.

If something is detected by the speech recognizer but it was not able to come up with a good enough guess about what was said, the system will produce negative feedback on the perception level, e.g. “I didn’t hear what you said.”.

We have configured the input module to set the INPUT variable to ‘TIMED_OUT’ if nothing is detected, and to ‘FAIL’ if something unrecognizable was detected.

Negative system contact feedback If the speech recognizer does not get any input within a certain time frame (specified by the TIMEOUT TIS variable), the INPUT variable will be set to ‘TIMED_OUT’ by the input module. The rule for selection of negative contact feedback is shown in (RULE 2.9).

```
(RULE 2.9)  RULE: selectIcmConNeg
            CLASS: select_icm
            PRE: { $INPUT= 'TIMED_OUT'
                  { is_empty($NEXT_MOVES)
                  { is_empty($/PRIVATE/AGENDA)
            EFF: { push(NEXT_MOVES, icm:con*neg)
```

Unless the system has something else to do, this will trigger negative contact ICM by the system, realised e.g. as “I didn’t hear anything from you.”. The purpose of this is primarily to indicate to the user that nothing was heard, but perhaps also to elicit some response from the user to show that she is still there. Admittedly, this is a rather undeveloped aspect of ICM in the current GODIS implementation, and alternative strategies could be explored. For example, the system could increase the timeout span successively instead of repeating negative contact ICM every five seconds. Other formulations with more focus on the eliciting function could also be considered, e.g. “Are you there?” or simply “Hello?”.

The second and third condition check that nothing is on the agenda or in NEXT_MOVES. The motivation for this is that there is no reason to address contact explicitly in this case, since any utterance from the system implicitly tries to establish contact.

Default ICM integration rule Since contact is not explicitly represented in the information state proper, integration of negative system contact ICM moves have no specific effect on the information state, and are therefore integrated by the default ICM integration rule shown in (RULE 2.10). Unless an ICM move has a specific integration rule defined for it, it will be integrated by this rule.

```

(RULE 2.10)  RULE: integrateOtherICM
              CLASS: integrate
              PRE: { fst($/PRIVATE/NIM, icm:A)
              EFF: { pop(/PRIVATE/NIM)
                   add(/SHARED/LU/MOVES, icm:A)

```

The condition and updates in this rule are straightforward.

Negative system perception feedback If the speech recognizer gets some input from the user but is not able to reliably figure out what was said (the recognition score may be too low), the INPUT variable gets set to 'FAIL'. This will trigger negative perception ICM, e.g. "I didn't hear what you said".

```

(RULE 2.11)  RULE: selectIcmPerNeg
              CLASS: select_icm
              PRE: { $INPUT='FAIL'
                   not in($NEXT_MOVES, icm:per*neg)
              EFF: { push(NEXT_MOVES, icm:per*neg)

```

The purpose of the second condition is to prevent selecting negative perception feedback more than once in the selection phase. As with negative system contact feedback, negative system perception feedback is integrated by the **integrateOtherICM** rule.

Negative understanding level feedback

Negative feedback can concern either of the two sublevels of the understanding level: semantic and pragmatic understanding.

Negative system semantic understanding feedback If some input is recognized by the recognition module, the interpretation module will try to find an interpretation of the input. If this fails, the LATEST_MOVES gets set to **failed** which triggers selection of negative semantic understanding feedback (e.g. "I don't understand"). In addition, positive perception feedback (e.g. "I heard 'perish' ") is produced to indicate to the user what the system thought she said.

This will only occur if the recognition lexicon covers sentences not covered by the interpretation lexicon.

```

(RULE 2.12)  RULE: selectIcmSemNeg
              CLASS: select_icm
              PRE: { $LATEST_MOVES=failed
                    { $INPUT=String
                    { not in($NEXT_MOVES, icm:sem*neg)
              EFF: { push(NEXT_MOVES, icm:per*pos:String)
                    { push(NEXT_MOVES, icm:sem*neg)

```

The purpose of the third condition is to prevent negative semantic understanding feedback from being selected more than one time. Since only one string is recognized per turn, there is never any reason to apply the rule more than once; and if anything at all can be interpreted, the rule will not trigger at all even if some material was not used in interpretation. In a system with a wide-coverage recognizer and a more sophisticated interpretation module, one may consider producing negative semantic understanding feedback for any material which cannot be interpreted (e.g. “I understand that you want to go to Paris, but I don’t understand what you mean by ‘Londres’.”).

The first update in this rule selects positive perception ICM to show the user what the system heard. The second update selects negative semantic understanding ICM.

Negative system pragmatic understanding feedback The system will try to integrate the moves according to the rules above in Section 2.6.7. If this fails (if there are still moves which have not been integrated), the rule in (RULE 2.13) will be triggered and a *icm:und*neg*-move will be selected by the system. However, if the reason that the move was not integrated is that it had a low score or was not acceptable to the system, interrogative understanding feedback (Section 2.6.6) or negative acceptance feedback (Section 2.6.6), respectively, will instead be selected and the move will be popped off NIM before the rule in (RULE 2.13) is tried.

In GODIS, only *ask*-moves can be irrelevant. Other moves, including *ask*, do not have any relevance requirements. This means that *answer* moves are the only moves that can fail to be understood on the pragmatic level, given that they have been understood on the semantic level. Also, for an utterance to be completely irrelevant, no part of it must have been integrated. For these reasons, the rule in (RULE 2.13) will trigger only if no move in the latest utterance was integrated, and the utterance was interpreted as containing at least one *answer*-move.

(RULE 2.13) RULE: **selectIcmUndNeg**
 CLASS: **select_icm**
 PRE: $\left\{ \begin{array}{l} \text{not in}(\$NEXT_MOVES, \text{icm:und*neg}) \\ \text{in}(\$LATEST_MOVES, \text{answer}(A)) \\ \text{forall}(\$LATEST_MOVES/ELEM=Move, \\ \quad \$/PRIVATE/NIM/ELEM=Move) \\ \text{forall}(\$LATEST_MOVES/ELEM=\text{answer}(A'), \\ \quad \text{not fst}(\$/SHARED/QUD, D) \text{ and } \$DOMAIN :: \text{relevant}(A', Q)) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{forall_do}(\$LATEST_MOVES/ELEM=Move, \\ \quad \text{push}(NEXT_MOVES, \text{icm:sem*pos:Move})) \\ \text{push}(NEXT_MOVES, \text{icm:und*neg}) \end{array} \right.$

The first rule checks that negative pragmatic understanding feedback has not already been selected. The second condition checks that the latest utterance contained an **answer** move, and the third checks that none of the moves performed in the latest utterance has been integrated; all moves in `LATEST_MOVES` are still on `NIM`. Finally, the fourth condition checks that no answer is **relevant** to any question on `QUD`.

The first update selects positive feedback on the semantic understanding level for each move performed in the latest utterance, to show that the utterance was at least understood to some extent. The second update selects negative feedback and pushes it on `NEXT_MOVES`.

The system is thus able to make a distinction between utterances it cannot interpret (and thus not ground), and utterances that it can interpret and ground but not integrate. The rule in (15) triggers when integration fails because the system cannot see the relevance of the user utterance in the current dialogue context. Negative pragmatic understanding feedback is currently realized as “I don’t quite understand”; the idea is to indicate that the utterance was almost fully understood, but not quite. Again, it can be argued what the best realization is.

Dialogue example: negative system contact, perception, and understanding feedback In this dialogue, we see examples of negative system feedback on the contact, perception, and understanding (both semantic and pragmatic) levels. (Since this dialogue uses the text-based input module, we have simulated timeout and recognition failure.)

(DIALOGUE 2.5)

S> Welcome to the travel agency!

U> 'TIMED_OUT'

```
getLatestMoves
integrateNoMove
{ pop(/PRIVATE/NIM)
selectIcmConNeg
{ push(NEXT_MOVES, icm:con*neg)
```

S> I didnt hear anything from you.

```
getLatestMoves
integrateOtherICM
{ pop(/PRIVATE/NIM)
```

U> 'FAIL'

```
backupShared
selectIcmPerNeg
{ push(NEXT_MOVES, icm:per*neg)
```

S> Sorry, I didnt hear what you said.

```
getLatestMoves
integrateOtherICM
{ pop(/PRIVATE/NIM)
```

U> jfdbhajhdgarbledfdasd

```
backupShared
selectIcmSemNeg
```

S> I heard you say jfdbhajhdgarbledfdasd. Sorry, I dont understand.

```
getLatestMoves
integrateOtherICM
{ pop(/PRIVATE/NIM)
integrateOtherICM
{ pop(/PRIVATE/NIM)
```

U> paris

```
getLatestMoves
backupShared
selectIcmUndNeg
```

$$\left\{ \begin{array}{l} \text{forall_do}(\$/\text{SHARED}/\text{LU}/\text{MOVES}/\text{ELEM}=A, \text{push}(\text{NEXT_MOVES}, \text{icm:sem*pos:}A)) \\ \text{push}(\text{NEXT_MOVES}, \text{icm:und*neg}) \\ \text{forall_do}(\text{in}(\$/\text{SHARED}/\text{LU}/\text{MOVES}, E) \text{ and } E=\text{answer}(C) \text{ and } \$\text{LEXICON} :: \text{yn_answer}(C) \text{ and} \\ \quad \text{in}(\$/\text{PRIVATE}/\text{NIM}, E), \\ \quad \text{del}(\$/\text{PRIVATE}/\text{NIM}, E)) \end{array} \right.$$

S> paris. I dont quite understand.

```
getLatestMoves
integrateOtherICM
{ pop(/PRIVATE/NIM)
integrateOtherICM
{ pop(/PRIVATE/NIM)
```

Negative reaction level feedback

To be able to distinguish **relevant** but non-integratable utterances from utterances which are both **relevant** and integratable, a dialogue system in general needs to be able to distinguish between relevance and integratability (acceptability), i.e. it needs to understand the relevance of utterances that it cannot integrate.

System proposition-rejection In addition to issue-rejection, proposition-rejection is also arguably relevant to a dialogue system. A case in point is when the user supplies information which results in an invalid database query, i.e. a query which would yield no results. An example is given in (DIALOGUE 2.6) (understanding-feedback has been removed for readability).

(DIALOGUE 2.6)

```
U(1)> Price information please
S(1)> OK. Where do you want to travel?
U(2)> to Paris
S(2)> OK. What city you want to travel from?
U(3)> Oslo
S(3)> Oslo. Sorry, there are no flights matching your
specification.
```

However, this case is a bit more problematic - is S(3) really a rejection of U(3), or should it be regarded as a negative answer to the user's query in U(1)? We believe it makes more sense to do the latter. On this view, the issue of price will be regarded as (negatively) resolved after S(3).

(Note that we are here assuming that Oslo is in fact a valid departure city, but there happen to be no flights from Oslo to Paris in the database.)

A variant of the dialogue in (DIALOGUE 2.6) that is perhaps a better case of rejection is where the user supplies a destination which is not available in the database. In this case, it seems to make sense to say that it is indeed the utterance containing the information about the destination that is rejected.

(DIALOGUE 2.7)

U(1)> Price information please
S(1)> OK. Where do you want to travel?
U(2)> to Paris
S(2)> OK. What city you want to travel from?
U(3)> Kuala Lumpur
S(3)> Sorry, Kuala Lumpur is not in the database. So, What city do you want to travel from?¹⁴

In this case, the issue of price is still unresolved, as is the issue of destination city. To handle a dialogue like that in (DIALOGUE 2.7), a system again needs to be able to recognize relevant information that it cannot deal with, and distinguish it from such information that it can deal with. One way of doing this is to encode relevant information in the domain knowledge resource that is not necessarily in the database. If a user utterance that contains a relevant answer or assertion is perceived and understood, the system should perform a database search to check if it is able to deal with that information; if not, the user's utterance should be rejected.

Of course, it is a well-known problem that bigger vocabularies make speech recognition harder, and consequently there's a tradeoff between recognizing and dealing correctly with non-acceptable information, and getting the acceptable information right. Possibly, one could use collected dialogues in a domain to decide how much non-acceptable information the system should be able to recognize and understand.

In GODIS, we have implemented the ability to reject user answers by checking whether they provide valid database parameters. This requires an additional database resource condition "validDBparameter(P)" which is true if P is a valid parameter in the database. For example, if a travel agency database contains flights within Europe, any destination outside Europe is an invalid database parameter and should be rejected by the system.

¹⁴Optionally, one might want a system to be more helpful and offer a suitable alternative destination.


```

(RULE 2.14)  RULE: rejectProp
              CLASS: select_action
              PRE: {
                    in(/PRIVATE/NIM, answer(A))
                    $/SHARED/LU/SPEAKER=usr
                    fst(/SHARED/QUD, Q)
                    $DOMAIN :: relevant(A, Q)
                    $DOMAIN :: combine(Q, A, P)
                    not $DATABASE :: validDBparameter(P)
              }
              EFF: {
                    del(/PRIVATE/NIM, answer(A))
                    push(/PRIVATE/AGENDA, icm:und*pos:usr*P)
                    push(/PRIVATE/AGENDA, icm:acc*neg:P)
              }

```

The first five conditions are identical to those for the rule for integrating user answers, **integrateUsrAnswer** (Section 2.6.6). The final condition checks that the proposition *P*, resulting from combining a question on QUD with the content of the **answer** move, is not a valid database parameter. The updates remove the move from NIM and selects positive understanding feedback to show what the system understood, and negative acceptance feedback.

Of course, it is not optimally efficient that the same sequence of conditions is checked by several different rules; an alternative approach would be to let one rule determine e.g. how an **answer** move is **relevant**, **combine** it with a question on QUD, and store the result in a datastructure containing pragmatically interpreted material. This datastructure could then be inspected by both integration and rejection rules. (See also 9.3.1.)

Dialogue example: system proposition rejection In the following dialogue, we illustrate system rejection of the proposition that the means of transport to search for will be train. A motivation is also given by the system, i.e. that “train” is not available as a means of transport in the database.

(DIALOGUE 2.8)

S> Okay. I need some information. How do you want to travel?

```

getLatestMoves
integrateOtherICM
integrateOtherICM
integrateSysAsk

```

U> train please

```

getLatestMoves
backupShared
rejectProp
{
  del(/PRIVATE/NIM, answer(train))
  push(/PRIVATE/AGENDA, icm:und*pos:usr*how(train))
  push(/PRIVATE/AGENDA, icm:acc*neg:how(train))
}
selectIcmOther
{
  push(NEXT_MOVES, icm:und*pos:usr*how(train))
  del(/PRIVATE/AGENDA, icm:und*pos:usr*how(train))
}
selectIcmOther
{
  push(NEXT_MOVES, icm:acc*neg:how(train))
  del(/PRIVATE/AGENDA, icm:acc*neg:how(train))
}

```

S> by train. Sorry, by train is not in the database.

```

getLatestMoves
integrateOtherICM
integrateOtherICM

```

System issue-rejection For example, the system might know some questions which are relevant in a certain activity, but not be able to answer them. This is not usually the case with existing dialogue systems. For example, the Swedish railway information system (based on the Philips dialog system (Aust *et al.*, 1994) cannot answer questions about the availability of a cafeteria on a train. If this question is asked, the system will try to interpret it as an answer to something it just asked about (as illustrated in the made-up dialogue (16)). But one could imagine a system that would have a store of potentially relevant questions which it cannot handle, enabling it to respond to such questions in a more appropriate way, e.g. by saying “Sorry, I cannot answer that question”. This would constitute a rejection (an issue-rejection, to be precise) of a question whose meaning has been understood. An (made-up) example is shown in (17).

- (16) U : Is there a cafeteria on the train?
 S : You want to travel to Siberia, is that correct?
- (17) U : Is there a cafeteria on the train?
 S : Sorry, I cannot answer questions about cafeteria availability.

Issue rejection has been implemented in GODIS2 for the travel agency domain; in the travel agency domain, the system will recognize and understand, but reject, questions about connecting flights. A possible extension of this would be to make the system more helpful and make it explain why it cannot answer the question; this has not yet been done in GODIS.

In case the system has interpreted a user utterance as an **ask**-move with content q , but the system does not have a plan for dealing with q , the system must reject q and indicate this to the user using appropriate feedback. This rule allows the system to respond intelligently to user questions even if it cannot answer them (given that they can be recognized and interpreted).

```
(RULE 2.15)  RULE: rejectIssue
              CLASS: select_action
              PRE: {
                    { in($/PRIVATE/NIM, ask( $Q$ ))
                      $/SHARED/LU/SPEAKER=usr
                      not $DOMAIN :: plan( $Q$ ,  $\_Plan$ )
                    }
              }
              EFF: {
                    { del(/PRIVATE/NIM, ask( $Q$ ))
                      push(/PRIVATE/AGENDA, icm:und*pos:usr*issue( $Q$ ))
                      push(/PRIVATE/AGENDA, icm:acc*neg:issue( $Q$ ))
                    }
              }
```

The rule is similar to the **rejectProp** rule. The third condition checks that there is no plan for dealing with the question Q .

Dialogue example: system issue rejection In the following dialogue, the user's request for information about connecting flights is rejected on the grounds that the system does not know how to address that issue.

(DIALOGUE 2.9)

S> Okay. The price is 123 crowns.

U> what about connecting flights

```
getLatestMoves
backupShared
rejectIssue
{
  del(/PRIVATE/NIM, ask(?A.con_flight(A)))
  push(/PRIVATE/AGENDA, )
  push(/PRIVATE/AGENDA, icm:acc*neg:issue(?A.con_flight(A)))
}
selectIcmOther
{
  push(NEXT_MOVES, icm:und*pos:usr*issue(?A.con_flight(A)))
  del(/PRIVATE/AGENDA, icm:und*pos:usr*issue(?A.con_flight(A)))
}
selectIcmOther
{
  push(NEXT_MOVES, icm:acc*neg:issue(?A.con_flight(A)))
  del(/PRIVATE/AGENDA, icm:acc*neg:issue(?A.con_flight(A)))
}
```

S> You asked about connecting flights. Sorry, I cannot answer questions about connecting flights.

```

getLatestMoves
integrateOtherICM
integrateOtherICM

```

2.6.7 Grounding of system utterances in GODIS2

In this section, we show how a cautiously optimistic grounding strategy for system utterances has been implemented in GODIS2. We first present basic update rules reflecting the cautious strategy. We then present integration rules for the “core” system dialogue moves (**ask** and **answer**), and describe the rules for integrating user feedback to system moves.

Enabling cautious updates

GODIS2 uses a mix of various grounding strategies. For system utterances, a cautiously optimistic strategy is used.

Moving latest moves to NIM The GODIS2 version of the update rule **getLatestMoves** is shown in (RULE 2.16).

```

(RULE 2.16)  RULE: getLatestMoves
              CLASS: grounding
              PRE: { $LATEST_MOVES=Moves
                    { $LATEST_SPEAKER=DP
                    { $/SHARED/LU/MOVES=PrevMoves
              EFF: { set(/PRIVATE/NIM, Moves)
                    { set(/SHARED/LU/SPEAKER, DP)
                    { clear(/SHARED/LU/MOVES)
                    { set(/SHARED/PM, PrevMoves)

```

The rule loads information regarding the latest utterance performed into NIM and copies the previously grounded moves (in /SHARED/LU/MOVES) to the /SHARED/PM field. Note that this rule has changed significantly compared to GODIS1; no optimistic assumption about understanding of the latest utterance is made here. Instead of putting the latest moves in /SHARED/LU/MOVES, which would be to assume that they have been mutually understood, GODIS2 clears /SHARED/LU/MOVES so that moves can be added when they are actually integrated; only then are they assumed to be understood.

Saving previous state before integration Before selecting, producing, and integrating a new system utterance, the rule in (RULE 2.17) copies relevant parts of the IS to the TMP field. This makes it possible to backtrack to a previous state, should the optimistic grounding assumptions concerning a system move turn out to be mistaken. This means that any optimistic updates associated with integration of system moves are now cautiously optimistic.

```
(RULE 2.17)  RULE: backupShared
              CLASS: none
              PRE: {
EFF: {
    /PRIVATE/TMP/QUD := $/SHARED/QUD
    /PRIVATE/TMP/COM := $/SHARED/COM
    /PRIVATE/TMP/AGENDA := $/PRIVATE/AGENDA
    /PRIVATE/TMP/PLAN := $/PRIVATE/PLAN
  }
```

There are no conditions on this rule. It is executed at the start of the selection algorithm described in Section 2.7, and is thus only called before system utterances.

Cautiously optimistic integration of system moves

For system **ask** and **answer** moves, the integration rules are similar to those in GoDIS1; however, rather than picking out moves from /SHARED/LU/MOVES, GoDIS2 picks moves from /PRIVATE/NIM and adds them to /SHARED/LU/MOVES, thereby assuming grounding on the understanding level, only in connection with integration. Since optimistic grounding is assumed for system moves, it would be okay to handle them the same way we did in GoDIS1; however, user moves are no longer (always) optimistically grounded, and we have chosen to give a uniform treatment to all moves. Since in GoDIS system moves are always successfully integrated, however, there is no real difference between the way they are handled in GoDIS1 and GoDIS2.

```
(RULE 2.18)  RULE: integrateSysAsk
              CLASS: integrate
              PRE: {
                $/SHARED/LU/SPEAKER==sys
                fst($/PRIVATE/NIM, ask(A))
EFF: {
    pop(/PRIVATE/NIM)
    add(/SHARED/LU/MOVES, ask(A))
    push(/SHARED/QUD, A)
  }
```

(RULE 2.19) **RULE: integrateSysAnswer**
 CLASS: integrate
 PRE: $\left\{ \begin{array}{l} \text{fst}(\$/\text{PRIVATE}/\text{NIM}, \text{answer}(A)) \\ \$/\text{SHARED}/\text{LU}/\text{SPEAKER} == \text{sys} \\ \$\text{DOMAIN} :: \text{proposition}(A) \\ \text{fst}(\$/\text{SHARED}/\text{QUD}, B) \\ \$\text{DOMAIN} :: \text{relevant}(A, B) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{pop}(\$/\text{PRIVATE}/\text{NIM}) \\ \text{add}(\$/\text{SHARED}/\text{LU}/\text{MOVES}, \text{answer}(A)) \\ \text{add}(\$/\text{SHARED}/\text{COM}, A) \end{array} \right.$

One complication is that in GODIS2, several moves may be performed in a single utterance. To keep track of which utterances have been integrated, the `/PRIVATE/NIM` stack of non-integrated moves is popped for each move that gets integrated. Note also that each integrated (and thus understood) move is added to `/SHARED/LU/MOVES` (whereas in GODIS1 this was done at the start of the update cycle).

The cautiously optimistic acceptance assumptions built into these rules can be retracted on integration of negative user perception feedback, as explained in Section 2.6.6, or on negative user integration feedback, as show in Section 2.6.7. Dialogue examples involving the rules shown above will be given in these sections.

User feedback to system utterances

In this section we review user feedback to system utterances and how these affect the optimistic grounding assumptions.

Negative user perception feedback If the system makes an utterance, it will assume it is grounded and accepted. If the user indicates that she did not understand the utterance, the rule in (RULE 2.20) makes it possible to retract the effects of the system’s latest move, thus cancelling the assumptions of grounding and acceptance.

(RULE 2.20) RULE: **integrateUsrPerNegICM**
 CLASS: **integrate**
 PRE: { $\begin{cases} \$/\text{SHARED}/\text{LU}/\text{SPEAKER} == \text{usr} \\ \text{fst}(\$/\text{PRIVATE}/\text{NIM}, \text{icm:per*neg}) \end{cases}$
 EFF: { $\begin{cases} \text{pop}(/ \text{PRIVATE}/\text{NIM}) \\ / \text{SHARED}/\text{QUD} := \$/\text{PRIVATE}/\text{TMP}/\text{QUD} \\ / \text{SHARED}/\text{COM} := \$/\text{PRIVATE}/\text{TMP}/\text{COM} \\ / \text{PRIVATE}/\text{AGENDA} := \$/\text{PRIVATE}/\text{TMP}/\text{AGENDA} \\ / \text{PRIVATE}/\text{PLAN} := \$/\text{PRIVATE}/\text{TMP}/\text{PLAN} \end{cases}$

The four last updates revert the COM, QUD, PLAN and AGENDA fields to the values stored in /PRIVATE/TMP.

Dialogue example: negative user perception feedback This dialogue shows how GODIS2 is able to react to negative user perception feedback (e.g. “pardon”) by retracting the optimistic grounding assumption by backtracking relevant parts of SHARED to the state in /PRIVATE/TMP/SYS, stored before the system utterance was generated. Also, the plan and agenda are backtracked to enable the system to continue the dialogue properly.

(DIALOGUE 2.10)

S> Okay. You asked about price. I need some information. How do you want to travel?

getLatestMoves
 integrateOtherICM
 integrateOtherICM
 integrateOtherICM
 integrateSysAsk

$$\left[\begin{array}{l} \text{PR.} \\ \text{SH.} \end{array} \right] = \left[\begin{array}{l} \text{AGENDA} = \langle \rangle \\ \text{PLAN} = \left\langle \begin{array}{l} \text{findout}(?A.\text{how}(A)) \\ \text{findout}(?B.\text{dest_city}(B)) \\ \text{findout}(?C.\text{dept_city}(C)) \\ \text{findout}(?D.\text{month}(D)) \\ \text{findout}(?E.\text{dept_day}(E)) \\ \text{findout}(?F.\text{class}(F)) \\ \text{consultDB}(?G.\text{price}(G)) \end{array} \right\rangle \\ \text{BEL} = \{ \} \\ \text{COM} = \{ \} \\ \text{QUD} = \langle ?H.\text{price}(H) \rangle \\ \text{AGENDA} = \left\langle \left\langle \begin{array}{l} \text{icm:acc*pos} \\ \text{icm:und*pos:usr*issue}(?H.\text{price}(H)) \\ \text{icm:loadplan} \end{array} \right\rangle \right\rangle \\ \text{TMP} = \left\langle \begin{array}{l} \text{findout}(?A.\text{how}(A)) \\ \text{findout}(?B.\text{dest_city}(B)) \\ \text{findout}(?C.\text{dept_city}(C)) \\ \text{findout}(?D.\text{month}(D)) \\ \text{findout}(?E.\text{dept_day}(E)) \\ \text{findout}(?F.\text{class}(F)) \\ \text{consultDB}(?G.\text{price}(G)) \end{array} \right\rangle \\ \text{PLAN} = \left\langle \begin{array}{l} \text{findout}(?A.\text{how}(A)) \\ \text{findout}(?B.\text{dest_city}(B)) \\ \text{findout}(?C.\text{dept_city}(C)) \\ \text{findout}(?D.\text{month}(D)) \\ \text{findout}(?E.\text{dept_day}(E)) \\ \text{findout}(?F.\text{class}(F)) \\ \text{consultDB}(?G.\text{price}(G)) \end{array} \right\rangle \\ \text{NIM} = \langle \rangle \\ \text{COM} = \{ \} \\ \text{QUD} = \left\langle \begin{array}{l} ?I.\text{how}(I) \\ ?H.\text{price}(H) \end{array} \right\rangle \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{sys} \\ \text{MOVES} = \left\langle \left\langle \text{icm:acc*pos}, \dots \right\rangle \right\rangle \end{array} \right] \\ \text{PM} = \left\langle \left\langle \text{ask}(?H.\text{price}(H)) \right\rangle \right\rangle \end{array} \right]$$

U> pardon

getLatestMoves

integrateUsrPerNegICM

$$\left\{ \begin{array}{l} \text{pop}(/PRIVATE/NIM) \\ /SHARED/QUD := \$/PRIVATE/TMP/QUD \\ /SHARED/COM := \$/PRIVATE/TMP/COM \\ /PRIVATE/AGENDA := \$/PRIVATE/TMP/AGENDA \\ /PRIVATE/PLAN := \$/PRIVATE/TMP/PLAN \end{array} \right.$$

PRIVATE	=	AGENDA	=	$\left\langle \left\langle \begin{array}{l} \text{icm:acc*pos} \\ \text{icm:und*pos:usr*issue(?A.price(A))} \\ \text{icm:loadplan} \end{array} \right\rangle \right\rangle$	$\left. \begin{array}{l} \text{findout(?B.how(B))} \\ \text{findout(?C.dest_city(C))} \\ \text{findout(?D.dept_city(D))} \\ \text{findout(?E.month(E))} \\ \text{findout(?F.dept_day(F))} \\ \text{findout(?G.class(G))} \\ \text{consultDB(?H.price(H))} \end{array} \right\rangle$
		PLAN	=	$\left\langle \begin{array}{l} \text{findout(?D.dept_city(D))} \\ \text{findout(?E.month(E))} \\ \text{findout(?F.dept_day(F))} \\ \text{findout(?G.class(G))} \\ \text{consultDB(?H.price(H))} \end{array} \right\rangle$	
		BEL	=	{ }	
		TMP	=	...	
		NIM	=	$\langle \langle \rangle \rangle$	
SHARED	=	COM	=	{ }	$\left. \begin{array}{l} \text{?A.price(A)} \end{array} \right\rangle$
		QUD	=		
		LU	=	$\left[\begin{array}{l} \text{SPEAKER} = \text{usr} \\ \text{MOVES} = \text{oqueue([icm:per*neg])} \end{array} \right]$	
		PM	=	...	

backupShared
 selectFromPlan
 selectIcmOther
 selectIcmOther
 selectIcmOther
 selectAsk

S> Okay. You asked about price. I need some information. How do you want to travel?

Explicit user issue rejection The rule in (RULE 2.21) allows the user to reject a system question (by indicating inability to answer, i.e. by uttering “I don’t know” or similar). If this is done, the optimistic grounding update is retracted by restoring the shared parts stored in NIM, i.e. QUD and COM, to their previous states.

(RULE 2.21) **RULE: integrateUsrAccNegICM**
 CLASS: integrate
 PRE: $\left\{ \begin{array}{l} \$/\text{SHARED}/\text{LU}/\text{SPEAKER} == \text{usr} \\ \text{fst}(\$/\text{PRIVATE}/\text{NIM}, \text{icm:acc*neg:issue}) \\ \text{in}(\$/\text{SHARED}/\text{PM}, \text{ask}(Q)) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{pop}(/ \text{PRIVATE}/\text{NIM}) \\ \text{add}(/ \text{SHARED}/\text{LU}/\text{MOVES}, \text{icm:acc*neg:issue}) \\ / \text{SHARED}/\text{QUD} := \$/\text{PRIVATE}/\text{TMP}/\text{QUD} \\ / \text{SHARED}/\text{COM} := \$/\text{PRIVATE}/\text{TMP}/\text{COM} \end{array} \right.$

The third condition checks that the previous utterance contained an **ask** move. The final two updates retract the optimistic grounding assumption on the integration / acceptance / reaction level.

Of course, if a question is rejected by the user this may result in a failed database query (unless the alternative database access method described in Larsson *et al.* (2002) is used). But how should a system react if the user rejects a system question? In some frame-based dialogue systems for database search (e.g. Chu-Carroll, 2000), fields in the frame can be labelled as obligatory or optional. In GODIS, this corresponds roughly to the distinction between the **raise** and **findout** actions; the former has succeeded as soon as the system asks the question, whereas the latter requires the question to be resolved. So if a question which was raised by a **raise** action was rejected, it will not be asked again. Questions raised by **findout** actions, however, will currently be raised again by GODIS2 immediately after a user rejection, since the action is still on top of the plan. This is perhaps not very cooperative, and alternative strategies need to be explored. For example, the **findout** action could be moved further down in the plan so that it will not be asked immediately again, or it may be raised again only if the database search fails.

Dialogue example: explicit user issue rejection In the following dialogue example, the user rejects the system question regarding how to travel. In this example, the plan has been altered so that **findout(?x.class(x))** has been replaced by **raise(?x.class(x))**, thereby making the class-question optional. Also, the alternative database access method described in Larsson *et al.* (2002) is used.

(DIALOGUE 2.11)

S> What class did you have in mind?

```
getLatestMoves
integrateSysAsk
{ pop(/PRIVATE/NIM)
  push(/SHARED/QUD, ?A.class(A))
```

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \begin{array}{l} \text{AGENDA} = \langle \rangle \\ \text{PLAN} = \left\langle \begin{array}{l} \text{raise}(\text{?}A.\text{class}(A)) \\ \text{consultDB}(\text{?}B.\text{price}(B)) \end{array} \right\rangle \\ \text{BEL} = \{ \} \\ \begin{array}{l} \text{COM} = \left\{ \begin{array}{l} \text{month}(\text{april}) \\ \text{dept_city}(\text{london}) \\ \text{dest_city}(\text{paris}) \\ \text{how}(\text{plane}) \end{array} \right\} \\ \text{QUD} = \langle \text{?}C.\text{price}(C) \rangle \\ \text{AGENDA} = \langle \rangle \\ \text{PLAN} = \left\langle \begin{array}{l} \text{raise}(\text{?}A.\text{class}(A)) \\ \text{consultDB}(\text{?}B.\text{price}(B)) \end{array} \right\rangle \end{array} \\ \text{NIM} = \langle \rangle \\ \begin{array}{l} \text{COM} = \left\{ \begin{array}{l} \text{month}(\text{april}) \\ \text{dept_city}(\text{london}) \\ \text{dest_city}(\text{paris}) \\ \text{how}(\text{plane}) \end{array} \right\} \\ \text{QUD} = \left\langle \begin{array}{l} \text{?}D.\text{class}(D) \\ \text{?}C.\text{price}(C) \end{array} \right\rangle \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{sys} \\ \text{MOVES} = \langle \langle \text{ask}(\text{?}D.\text{class}(D)) \rangle \rangle \end{array} \right] \\ \text{PM} = \langle \langle \text{icm:acc*neg:issue} \rangle \rangle \end{array} \end{array} \right] \end{array} \right]$$

U> it doesnt matter

getLatestMoves

integrateUsrAccNegICM

$\left\{ \begin{array}{l} \text{pop}(/PRIV\text{ATE}/\text{NIM}) \\ \text{add}(/SH\text{ARED}/\text{LU}/\text{MOVES}, \text{icm:acc*neg:issue}) \\ /SH\text{ARED}/\text{QUD} := \$/PRIV\text{ATE}/\text{TMP}/\text{QUD} \\ /SH\text{ARED}/\text{COM} := \$/PRIV\text{ATE}/\text{TMP}/\text{COM} \end{array} \right.$

exec_consultDB

$$\left[\begin{array}{l} \text{PR.} \\ \text{SH.} \end{array} = \left[\begin{array}{l} \text{BEL} \\ \text{TMP} \\ \text{NIM} \\ \text{COM} \\ \text{QUD} \\ \text{LU} \\ \text{PM} \end{array} = \left[\begin{array}{l} \text{db_entry} \left(\left\{ \begin{array}{l} \text{month(april)} \\ \text{dept_city(london)} \\ \text{dest_city(paris)} \\ \text{how(plane)} \end{array} \right\}, \{ \text{class(economy)}, \text{price(123)} \} \right) \\ \text{db_entry} \left(\left\{ \begin{array}{l} \text{month(april)} \\ \text{dept_city(london)} \\ \text{dest_city(paris)} \\ \text{how(plane)} \end{array} \right\}, \{ \text{class(business)}, \text{price(1234)} \} \right) \\ \text{COM} = \left\{ \begin{array}{l} \text{month(april)} \\ \text{dept_city(london)} \\ \text{dest_city(paris)} \\ \text{how(plane)} \end{array} \right\} \\ \text{QUD} = \langle ?B.\text{price}(B) \rangle \\ \text{AGENDA} = \langle \rangle \\ \text{PLAN} = \langle \text{raise}(?C.\text{class}(C)) \\ \text{consultDB}(?D.\text{price}(D)) \rangle \\ \text{NIM} = \langle \rangle \\ \text{COM} = \left\{ \begin{array}{l} \text{month(april)} \\ \text{dept_city(london)} \\ \text{dest_city(paris)} \\ \text{how(plane)} \end{array} \right\} \\ \text{QUD} = \langle ?B.\text{price}(B) \rangle \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{usr} \\ \text{MOVES} = \langle \langle \text{icm:acc*neg:issue} \rangle \rangle \end{array} \right] \\ \text{PM} = \langle \langle \text{ask}(?C.\text{class}(C)) \rangle \rangle \end{array} \right] \end{array} \right]$$

backupShared
 selectRespond
 selectAnswer

S> The price is 123 crowns. cheap. The price is 1234 crowns.
 business class.

2.6.8 Evidence requirements and implicit grounding

In this section, we discuss evidence requirements for grounding and how these have been implemented in the form of update rules for implicit grounding.

In GoDiS2 we use a cautiously optimistic grounding strategy for system utterances. This assumption can be retracted if negative evidence concerning grounding is found. So, what counts as negative and positive evidence? Recall Clark's ranking of different forms of positive evidence, ranging from weakest to strongest:

- Continued attention
- Relevant next contribution
- Acknowledgement: “uh-huh”, nodding, etc.
- Demonstration: reformulation, collaborative completion
- Display: verbatim display of presentation

Regarding the attention level, we will not have much to say¹⁵. The levels of acknowledgement, demonstration and display are presumably what we would regard as explicit feedback, although we have been mainly concerned with the acknowledgement level.

Evidence and relevance

The remaining level in Clark’s typology of evidence of understanding is “relevant next contribution”. Two questions arise here. First, what counts as a relevant followup? Second, if no relevant followup is produced, should this count as negative evidence of grounding, and if so, on what action level?

A property of dialogue systems sometimes discussed in the literature (The DISC consortium, 1999, Bohlin *et al.*, 1999) is the ability of a system to understand and integrate information different from that which was requested by the system. How does this relate to relevance and grounding? One way to formulate the problem is this: if the system just asked q , and the user’s response u did not contain an answer **relevant** to q or feedback concerning the system’s utterance, what should be assumed about the grounding status of q ? This is, of course, also a problem that human DPs must resolve; however, Clark does not (to our knowledge) directly discuss this case.

- (18) a. A: What city do you want to go to? [ask q]
 B: I’d like to travel in April [answer other question]
- b. A: What city do you want to go to? [ask q]
 B: Do you have a student discount? [ask other question]

¹⁵Clark includes “continued attention” as the weakest form of positive evidence of grounding. However, in principle continued attention from an addressee A after an utterance u is consistent with a complete lack of perception on A ’s side; A may not even have perceived u but is still waiting for the next utterance. While this example may not be very relevant for human-human communication, it is not a completely unlikely scenario if A is a dialogue system. Also, contact level feedback appears related to this.

Regarding cases where a question is ignored (i.e. neither addressed by a relevant answer, explicitly accepted, nor explicitly rejected), it is not obvious whether the question was accepted or not. The reason is that there are several possible explanations for this behaviour: one complies silently with the question but thinks that other information is more important right now (in which case the question was integrated by the hearer, and will be answered eventually), or one misheard or did not hear the question at all (in which case it was not understood, and thus neither accepted or rejected), or one does not think that the question is appropriate (in which case the question was implicitly rejected).

One possible strategy for finding negative evidence is to look for signs of misunderstanding, and to try to come up with a plausible explanation for how this misunderstanding came about. This is, however, a fairly difficult task even for humans and not one we intend to explore here.

Cases where a question is not followed by a relevant answer or relevant ICM, can be regarded as implicit rejections of that question. However, if the followup is relevant in some other way to the question asked, this should not be regarded as rejection. One type of relevant followup can be defined using Ginzburg's notion of question dependence:

- (19) An **ask**-move with content q is a relevant followup to an **ask**-move with content q' if q' depends on q .

In Larsson *et al.* (2002), we defined a domain-dependent notion of question dependence related to terms of plans, where q' depends on q if there is a plan for dealing with q' which includes an action `findout(q)`.

Consequently, in GODIS2 we have chosen the following requirements on an utterance u to count as an irrelevant followup to an utterance raising a question q :

- u contains no ICM
- the previous move raised a question q
- u contains no answer to q
- u contains no **ask**-move raising a question q' such that q depends on q'

Concerning our second question, are irrelevant followups to be regarded as negative grounding evidence? Or could it be the case that a *DP* understood and accepted an utterance u but opted to change the subject temporarily, planning to respond to u eventually?

If the irrelevant followup is interpreted as negative grounding evidence, how do we know what action level is concerned? Did the user implicitly reject the issue by ignoring it, or did she simply

not perceive or understand it? We suspect that the choice between these two interpretations might depend on quite subtle issues concerning timing. For example, if the user's followup overlaps with the system's question it is possible that the user has not even heard the system's question.

In GoDis2 we have chosen to consider irrelevant followups to system **ask** moves as implicit rejections. However, this choice is not obvious and is a further topic for future research.

Implicit user rejection of issue

If an irrelevant followup is detected, this is interpreted as an implicit issue rejection and consequently the optimistic assumption that the question q' was integrated by the user is assumed to be mistaken. Therefore, the optimistic assumption is retracted by reverting the previous shared state for the relevant parts of SHARED.

(RULE 2.22) **RULE: irrelevantFollowup**
 CLASS: none
 PRE: $\left\{ \begin{array}{l} 1 \text{ } \$\text{/PRIVATE/NIM}=\textit{Moves} \\ 2 \text{ } \$\text{/SHARED/LU/SPEAKER}==\textit{usr} \\ 3 \text{ not } A/\text{ELEM}=\textit{icm}:_ \\ 4 \text{ in}(\$ \text{/SHARED/PM}, \textit{PrevMove}) \\ 5 \text{ } \textit{PrevMove}=\textit{ask}(Q) \text{ or} \\ \quad (\textit{PrevMove}=\textit{icm}:\textit{und}*\textit{int}:DP*C \text{ and } Q=\textit{und}(DP*C)) \\ 6 \text{ not } \textit{Moves}/\text{ELEM}=\textit{ask}(Q') \text{ and } \$\text{DOMAIN} :: \textit{depends}(Q, Q') \\ 7 \text{ not } A/\text{ELEM}=\textit{answer}(A) \text{ and } \$\text{DOMAIN} :: \textit{relevant}(A, Q) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{/SHARED/QUD} := \$\text{/PRIVATE/TMP/QUD} \\ \text{/SHARED/COM} := \$\text{/PRIVATE/TMP/COM} \end{array} \right.$

(Since this rule is called “by name” from the update algorithm, there is no need for including it in a rule class.) Condition 3 checks that no ICM was included in the latest move. Condition 4 and 5 tries to find a question Q raised by the previous move, possibly an understanding-question. Note here that we do not check QUD; in GoDis2, questions remain on QUD only for one turn but it may be the case that we want questions to remain on QUD over several turns. What we are interested here is thus not which questions are on QUD but which questions were raised by the previous utterance, and this is the reason for looking in PM rather than QUD. Conditions 6 and 7 check that no move performed in the latest utterance is relevant to Q , neither by answering it nor by asking a question on which Q depends. The updates are similar to those for integration of negative acceptance feedback (Section 2.6.7).

As is the case for explicit rejections, questions raised by **findout** actions will be asked again, but questions raised by **raise** actions will not. ICM-related questions (interrogative understanding

feedback) are not repeated since they are not in the plan but only on the agenda.

A dialogue involving implicit user rejection of an issue will be shown later in (DIALOGUE 3.12).

2.6.9 Sequencing ICM: reraising issues and loading plans

In this section, we review sequencing-related ICM and show how this has been implemented in GoDIS2.

We believe it is good practice to try to keep the user informed about what's going on inside the system, at least to a degree that facilitates a natural dialogue where system utterances "feel natural". One way of doing this is to produce ICM phrases indicating significant updates to the information state which are not directly related to specific user utterances. Using Allwood's (1995) terminology, we refer to these instances of ICM as "sequencing ICM".

For GoDIS2, we will implement two types of sequencing ICM. First, when loading a plan GoDIS2 will indicate this. Second, GoDIS2 will produce ICM to indicate when an issue is being reraised (in contrast to being raised for the first time).

Loading plans

GoDIS2 will indicate when a plan is being loaded, thus preparing the user to answer questions. This is currently generated as "Let's see."

The rule for finding an appropriate plan to deal with a respond-action on the agenda is similar to that in GoDIS1. The difference is that the GoDIS2 rule produces ICM to indicate that it has loaded a plan, formalized as `icm:loadplan` and generated e.g. as "Let's see". Again, the choice of output form is provisory.

```
(RULE 2.23)  RULE: findPlan
              CLASS: load_plan
              PRE: { in($/PRIVATE/AGENDA, respond(Q))
                    $DOMAIN :: plan(Q, Plan)
                    not in($/PRIVATE/BEL, P) and $DOMAIN :: resolves(P, Q)
              }
              EFF: { del(/PRIVATE/AGENDA, respond(Q))
                    set(/PRIVATE/PLAN, Plan)
                    push(/PRIVATE/AGENDA, icm:loadplan))
              }
```


This rule is identical to that in GODIS1 (Larsson *et al.* (2002)), except for the final update which pushes the `icm:loadplan` move on the agenda.

Reraising issues

System reraising of issue associated with plan If the user raises a question Q and then raises Q' before Q has been resolved, the system should return to dealing with Q once Q' is resolved; this was described in Section 2.6.9. The **recoverPlan** rule in GODIS2, shown in (20), is almost identical to the one in GODIS1, except that ICM is produced to indicate that an issue ($q1$) is being reraised. This ICM is formalized as `icm:reraise:q` where q is the question being reraised, and expressed e.g. as “Returning to the issue of price”.

(RULE 2.24) RULE: **recoverPlan**
 CLASS: `load_plan`
 PRE: $\left\{ \begin{array}{l} \text{in}(\$/\text{SHARED}/\text{QUD}, Q) \\ \text{is_empty}(\$/\text{PRIVATE}/\text{AGENDA}) \\ \text{is_empty}(\$/\text{PRIVATE}/\text{PLAN}) \\ \$\text{DOMAIN} :: \text{plan}(Q, Plan) \\ \text{not in}(\$/\text{PRIVATE}/\text{BEL}, P) \text{ and } \$\text{DOMAIN} :: \text{resolves}(P, Q) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{set}(\$/\text{PRIVATE}/\text{PLAN}, Plan) \\ \text{push}(\$/\text{PRIVATE}/\text{AGENDA}, \text{icm:reraise:}Q) \\ \text{push}(\$/\text{PRIVATE}/\text{AGENDA}, \text{icm:loadplan}) \end{array} \right.$

Issue reraising by user In the case where the user reraises an open issue, an `icm:reraise:Q` move is selected by the **integrateUsrAsk** described in Section 2.6.6.

System reraising of issue not associated with plan The GODIS1 **reraiseIssue** rule described in Larsson *et al.* (2002) reraises any questions on QUD which are not associated with any plan (i.e. which have been raised previously by the system). In this case it is again helpful to indicate that the system is aware that the issue is being reraised. However, since the issue will be reraised, the sequencing ICM does not need to indicate which question is being reraised.

(RULE 2.25) RULE: **reraiseIssue**
 CLASS: `select_action`
 PRE: $\left\{ \begin{array}{l} \text{fst}(\$/\text{SHARED}/\text{ISSUES}, Q) \\ \text{not } \$\text{DOMAIN} :: \text{plan}(Q, P) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{push}(\$/\text{PRIVATE}/\text{AGENDA}, \text{icm:reraise}) \\ \text{push}(\$/\text{PRIVATE}/\text{AGENDA}, \text{raise}(Q)) \end{array} \right.$

The conditions of this rule checks that there is a question Q on ISSUES for which the system has no plan, i.e. one that the system needs to ask the user.

The first update adds an `icm:reraise` (without an argument) to signal that it is reraising a question; this is currently generated as prefixing “so” to the next `ask` move, which is an ordinary raising of the question (placed on the agenda by the second update in the rule). In a more sophisticated implementation one could consider abbreviating the original raising of the question to make an appropriate reraising, e.g. “So, from what city?” But we do not have a general method for doing that yet, even in these simple dialogues. To do it properly would probably require more detailed syntactic and semantic analysis of contributions than we currently have in GODIS.

Dialogue example: user and system reraising (DIALOGUE 2.12) is an example of a dialogue where the user reraises an issue which was previously raised but not resolved.

Also subsequently the system reraises an issue which is relevant for addressing the issue that the user reraised. In order to address the issue of price the system needs to get information from the user concerning where she wishes to travel from.

(DIALOGUE 2.12)

```
S> Welcome to the travel agency!
U> price information please
S> Okay. Lets see. How do you want to travel?
U> a flight
S> Okay. What city do you want to go to?
U> paris
S> Okay. What city do you want to go from?
U> right... actually, do i need a visa
user accepts departure city issue, then raises visa issue
S> Okay. Lets see. What country are you from?
U> um... lets talk about price again
user implicitly rejects (avoids) citizenship issue, reraises price issue
S> Okay. Returning to the issue of price. Lets see. So, What
city do you want to go from?
system indicates returning to price issue, then reraises departure city issue
U> london
S> Okay. What month do you want to leave?
U> april
S> Okay. What day do you want to leave?
U> fifth
S> Okay. What class did you have in mind?
```

U> cheap

S> Okay. The price is 123 crowns. Returning to the issue of visa. Let's see. What country are you from?

system answers price issue, thereby resolving it, then reraises the visa issue, indicates plan loading, and reraises citizenship issue

The first case of reraising here, where it is the user who reraises the price issue, is handled by the conditional update of the **integrateUsrAnswer** (Section 2.6.6) which selects an `icm:reraise:Q` move to signal that the system is aware that issue was already open by saying “You reraised the issue of price”. In the same utterance, the system reraises the issue of where the user wants to travel from, requires a selection rule for the system. When reraising an issue, GODIS2 produces ICM to indicate awareness that the issue has been raised before. This ICM is formalized as `icm:reraise` and can be realized e.g. by the discourse particle “So”. Note that this would not have happened if the user had not accepted this question (by saying “right”) when it was first raised. Since the system does not regard the departure city question as dependent on the visa issue, raising the visa issue in response to asking for departure city would have been regarded as an implicit rejection (Section 2.6.8).

Once the price issue has been resolved, the system reraises the visa issue which is still unresolved; this is done by the **recoverPlan** rule as described in Section 2.6.9.

2.7 Further implementation issues

In this section we describe parts of the implementation of GODIS2 which have not been discussed earlier in this chapter, and which are not directly reused from GODIS1.

2.7.1 Update module

The GODIS2 update algorithm is shown in (20).

```

(20) 1 if not LATEST_MOVES == failed
      2 then { getLatestMove,
      3         try irrelevantFollowup,
      4         repeat integrate,
      5         try load_plan,
      6         repeat manage_plan
      7         try downdate_qud }
      8 else try unclearFollowup

```

Line 1 checks that the interpretation of the latest utterance was successful (of course, in the case of system utterances this is always true). If not, the **unclearFollowup** rule described in Section 2.6.8 is tried. If interpretation was successful, the latest moves are incorporated in the information state proper by the **getLatestMoves** rule (see Section 2.6.7). Before integration starts, the **irrelevantFollowup** rule described in Section 2.6.8 is tried to catch cases where a system question has been ignored by the user. After this, the integration rule class is repeatedly applied until the system has tried to integrate all moves in /PRIVATE/NIM. If the user asked a question, the appropriate plan will be loaded by line 5. Any loaded plan is executed by applying the **exec_plan** rule class until no more execution is possible at the current stage of the dialogue.

2.7.2 Selection module

As in GODIS1, action selection rules add actions to the agenda. However, while in GODIS1 only one action was selected per turn, in GODIS2 several actions may be selected per turn. For example, the **selectRespond** in GODIS2, shown in (RULE 2.26), does not require the agenda to be empty, but only that the **respond** action has not already been selected, and thus it allows several moves to be selected per turn.

```

(RULE 2.26)  RULE: selectRespond
              CLASS: select_action
              PRE: {
                    { is_empty($/PRIVATE/PLAN)
                      fst($/SHARED/QUD, A)
                      in($/PRIVATE/BEL, B)
                      not in($/SHARED/COM, B)
                      $DOMAIN :: resolves(B, A)
                      not in($/PRIVATE/AGENDA, respond(A))
                    }
              }
              EFF: { push(/PRIVATE/AGENDA, respond(A))

```

Similarly, the move selection rules in GODIS2 are repeatedly applied, popping actions off the AGENDA queue and pushing the corresponding moves on NEXT_MOVES. As an example, the

selectAnswer rule is shown in (RULE 2.27).

```
(RULE 2.27)  RULE: selectAnswer
              CLASS: select_move
              PRE: { fst($/PRIVATE/AGENDA, respond(A))
                    in($/PRIVATE/BEL, B)
                    not in($/SHARED/COM, B)
                    $DOMAIN :: resolves(B, A)
              EFF: { push(NEXT_MOVES, answer(B))
                    pop(/PRIVATE/AGENDA)
```

The selection algorithm for GODIS2 is shown in (21).

```
(21)  { backupShared,
        if not in($/PRIVATE/AGENDA, A) and q_raising_action(A)
        then try select_action,
        repeat ( select_icm or else select_move ) }
```

The **select_action** rule class selects actions and places them on the AGENDA, whereas the **select_move** and **select_icm** rule classes select AGENDA items and places them on NEXT_MOVES. Before selection, the **backupShared** (Section 2.6.7) is applied to copy relevant parts of the information state to /PRIVATE/NIM.

The basic strategy for selection in GODIS is that only one question should be raised by the system in each utterance. The GODIS2 selection algorithm first checks if some question-raising action is already on the agenda; if not, it tries to select a new action. Then, it selects moves and ICM repeatedly until nothing more can be selected.

The “q_raising_action(*A*)” condition uses a macro condition (see SIRIDUS (2002)) whose definition is shown in (22). What this says is, basically, that interrogative ICM, **raise** and, **findout** actions raise questions.

```
(22)  q_raising_action(Move) if
        Move = icm:und*int:X or Move = raise(X) or Move = findout(X)
```

2.8 Discussion

2.8.1 Some grounding-related phenomena not handled by GODIS2

In this section we mention some areas which have not been accounted for in the issue-based approach presented here. We do not by any means claim that this list is complete.

Perhaps the most significant omission in GODIS2 is a treatment of semantic ambiguity, e.g. ambiguous words. A possible direction of research in this area is to handle semantic ambiguity on a pragmatic level. Specifically, the relevance of an ambiguous move in the current dialogue context may be sufficient to resolve the semantic ambiguity, or at least reduce the number of possible semantic interpretations. In any case, we see no reason that mechanisms similar to those for dealing with pragmatic ambiguity could be used for semantic ambiguity.

Another area that remains unexplored from the point of view of issue-based dialogue management is semantic vagueness. For instance, one might want a system to understand vague answers (e.g. “I want to go to southern France”, “I want to travel around the 10th of April”), and perhaps also to ask less specific questions which leave more room for the user to choose how to specify e.g. parameters for database search (e.g. “Where do you want to travel?” rather than “What city do you want to go to?”).

On the pragmatic understanding level, we have concentrated on ellipsis resolution and relevance, however we are still lacking a treatment of referent resolution. One reason for this is of course that GODIS2 does not represent referents. This is a fairly well-researched area, and we hope to be able to include some existing account of referent resolution when this becomes necessary.

Overlapping user feedback and and barge-in

Most dialogue systems do not handle feedback from the user in any form, and most (if not all) existing systems which handle barge-in will stop talking if they perceive any sound from the speaker. This means that even positive feedback (e.g. “uhuh”) from the user will cause the system to stop speaking. This problem is aggravated in noisy environments, where noises may be misinterpreted as speech from the user and cause a system to stop speaking. What is needed is clearly that the system makes a distinction between different kinds of feedback from the user; positive feedback should usually not cause the system to stop speaking.

Mechanisms for handling overlapping user feedback has been explored within the GoDiS framework (Berman, 2001), but are not included here. However, the inclusion of positive user feedback in GODIS provides a basis for further explorations in this area.

2.8.2 Towards an issue-based account of grounding and action levels

We have hinted that a full-coverage account of grounding should include grounding on all four action-levels. Ginzburg's content- and acceptance-questions indicate how this could be accomplished in an issue-based theory of dialogue. For each action level, grounding issues can be raised and addressed; feedback moves on level L are regarded as addressing grounding issues on level L.

This would allow grounding to be handled by the same basic update mechanisms as questions and answers. A distinction can be made between short (elliptical, underspecified) answers (feedback utterances whose action level is not explicit) and full answers (feedback utterances whose action level is clear from the form and content of the utterance).

In GODiS, we strive for simplicity at the cost of completeness; however, the account given here can be seen as a first step towards a more complete issue-based account of grounding an action levels in dialogue. A sketch of a more complete account can be found in Section 9.3.1.

2.8.3 Comparison to Traum's computational theory of grounding

Traum (1994) provides a computational account of grounding based on a combination of finite automata and cognitive modelling. This model builds on Clark and Schaefer (1989b) but attempts to solve some computational problems inherent in that account.

Traum argues that Clark's account of the presentation and acceptance phases is problematic from a computational point of view. Firstly, it may be hard to tell if a speech signal is part of the presentation or acceptance phase. Second, it is hard to know when a presentation or acceptance is finished; often, this is only possible in hindsight, which may cause problems for a dialogue system engaged in real-time spoken dialogue. Third, it is unclear whether grounding acts (in our terminology, ICM dialogue moves) themselves need to be grounded.

Regarding the last point, we follow Traum in assuming that ICM moves do not need to be grounded. In fact, on our view this amounts to an optimistic grounding strategy where ICM moves are concerned.

We agree that in general the problem of deciding when a contribution ends is one that should be handled as a part of dialogue management, and that something like Traum's atomic grounding acts are needed for this. However, for the time being we make the simplifying assumption that contributions are already segmented before dialogue management starts; in the implementation, we rely on the speech recognizer's built-in algorithms for deciding when an utterance is finished.

Our account does not address the first point, i.e. the problem of jointly produced contributions, where DPs e.g. can repair each other's utterances. Traum proposes a recursive transition network (RTN) model of the grounding process which includes repairs, requests for repairs, acknowledgements and requests for acknowledgements (a simpler finite state model is also provided). Our account does not include repairs or requests for acknowledgements; however, Traum's acknowledgements correspond roughly to positive feedback and requests for repairs correspond (very) roughly to negative feedback.

It is important to note here that Traum (1994) uses the term "grounding" to refer exclusively to what we call "understanding-level grounding". It is notable that Traum focuses almost exclusively on positive feedback, whereas negative feedback is given a less detailed treatment. The grounding act most closely corresponding to negative feedback is request for repair; however, it is doubtful whether all negative feedback can be regarded as requests (e.g. "I don't understand"). To use Allwood's terminology, feedback has both an expressive dimension (expressing lack of perception, understanding, acceptance) and an evocative dimension (requesting a "repair" or repetition/reformulation). It appears that Traum has focused more on the evocative dimension whereas we have been more concerned with the expressive dimension. (We do feel that the evocative aspect of feedback is something that perhaps deserves more attention than we have given it so far; this is yet another area for future research.)

The dialogue acts "accept" and "reject" are regarded by Traum as "Core Speech Acts" on the same level as assertions, asking questions, giving instructions etc. The **accept** act is defined as "agreeing to a proposal" (p.58), which gives the impression that an acceptance act is a natural followup to some proposal act.

However, Traum's acceptance act also has similarities to what we refer to as positive reaction-level feedback. For example, an acceptance move may follow an assertion or an instruction, and the effects of the accept act is to change the status of the content of the assertion or instruction from being merely proposed to actually being shared. Regarding questions, it is unclear whether they need to be accepted before being shared. According to Traum, asking a question imposes an obligation on the addressee to address the question, i.e. to either answer it or to reject it. This seems (although it is far from clear) to indicate that questions on Traum's account are optimistically assumed to be grounded whereas assertions and instructions are not.

In Chapter 4, we extend the issue-based account of dialogue to negotiative dialogue, and argue that two kinds of acceptances need to be distinguished: acceptance as positive feedback on the reaction level, and acceptance of a proposed alternative solution to some problem (e.g. a certain domain plan as one among several ways to reach some goal).

2.9 Summary

After providing some dialogue examples where various kinds of feedback are used, we reviewed some relevant background, and discussed general types and features of feedback as it appears in human-human dialogue. Next, we discussed the concept of grounding from an information update point of view, and introduced the concepts of optimistic, cautious and pessimistic grounding strategies. We then related grounding and feedback to dialogue systems, and discussed the implementation of a partial-coverage model of feedback related to grounding in GODIS2. This allows the system to produce and respond to feedback concerning issues dealing with the grounding of utterances.

Chapter 3

Addressing unraised issues

3.1 Introduction

In the previous chapter, we discussed various mechanisms for handling grounding¹. One of the action levels to which grounding applies is that of pragmatic understanding, i.e. making sense of the meaning of an utterance in the current dialogue context. Some basic mechanisms for grounding on the understanding level were implemented in GODIS2. However, the kinds of dialogues handled by this system are still rather rigid and system-controlled.

The aim of the current chapter is to enable more flexible dialogue. After reviewing some shortcomings of GODIS2, we take a closer look at the notions underlying the QUD data structure, which results in dividing QUD into two substructures, one global and one local. Next, the notion of question accommodation is introduced to allow the system to be more flexible in the way utterances are interpreted relative to the dialogue context. Among other things, question accommodation allows the system to understand answers to questions which have not yet been asked, and to understand such answers even before any issue has been explicitly raised. In cases of ambiguity, clarification dialogues may be needed. Question accommodation combined with (very basic) belief revision abilities also allows GODIS to reaccommodate questions which have previously been resolved. Finally, a version of reaccommodation, where reaccommodation of one issue requires reaccommodation of a dependent issue as well, allows for successive modifications of database queries.

The division of QUD into a global and a local structure also enables a simple accommodation mechanism allowing the user to correct the system in cases where explicit positive feedback shows that the system has misunderstood a user utterance.

¹This chapter is a slightly altered version of Chapter 4 in Larsson (2002a).

Apart from the initial and final sections, this chapter is structured around the various question accommodation mechanisms. For each type of accommodation, there is an informal description, a formalization consisting of one or more update rules, and dialogue examples.

3.2 Some limitations of GODIS2

Handling answers to unasked questions

The dialogue structure allowed by the GODIS2 system is rather rigid and system-controlled. The main part of the dialogue consists of the system asking questions which the user has to answer. The user is not allowed to give more information, or different information, than what the system has just asked for.

In general, we require that the content of each **answer**-move must match a question on QUD. In GODIS2, the only way questions can end up on QUD is by being explicitly asked. This forces a simple tree structure on dialogue. In real dialogue, however, people often perform utterances which can be seen as answers to questions, or addressing issues, which have not yet been raised.

Revising information

Once the user has supplied some information to GODIS2, this information cannot be changed. This is clearly undesirable, and solving this problem would provide several advantages:

- The user may change his mind during the specification of the database query
- After the user has been given e.g. price information for a specified trip, he can modify some of the information to produce a new query, without having to enter all information again

Correcting explicit positive feedback

An important factor influencing the choice of feedback and grounding strategies in a dialogue system is usability (including efficiency of dialogue interaction). A disadvantage of the confirmation-question approach is that the dialogue becomes slow and tiring for the user, which decreases the efficiency and usability of the system.

For this reason, in the previous chapter we added the capability of producing feedback on the understanding level in non-eliciting form, i.e. as a declarative or elliptical sentence (without question intonation). However, this solution is unsatisfactory since the system may be mistaken and there is no way to correct it. A very natural response to positive explicit feedback which indicates a misunderstanding is to protest, e.g. by saying “no!”, possibly followed by a correction.

In this chapter, we use a special case of question accommodation to allow this, thus extending the issue-based account of grounding. If the user is satisfied with the system’s interpretation, she does not have to do anything; the system will eventually continue (possibly after a short pause) with the next step in the dialogue plan. The user’s silence is regarded as an implicit compliance with the system’s feedback. There is also the option of giving an explicit positive response to the feedback (e.g. “yes” or “right”). Finally, if the user responds negatively to the system’s feedback (e.g. by saying “no”), the system will understand that it misunderstood the user and act accordingly.

3.3 The nature(s) of QUD

Before extending the capabilities of GODIS, we will investigate the nature of QUD and make some distinctions between the different tasks that QUD can be used for. We will draw the conclusion that QUD needs to be divided into two substructures, one global and one local.

In this section, we present and compare some alternative notions of QUD.

3.3.1 Ginzburg’s definition of QUD

In Ginzburg (1997), Ginzburg provides the following definition of QUD:

QUD (‘questions under discussion’): a set that specifies the currently discussable questions, partially ordered by \prec (‘takes conversational precedence’). If q is maximal in QUD, it is permissible to provide any information specific to q using (optionally) a short-answer. (Ginzburg, 1997, p. 63)

While the definition above merely states that QUD is a partially ordered set, the operations performed on QUD in Ginzburg’s protocols suggest that in fact it is more like a partially ordered stack². In GODIS1 and GODIS2 we made the simplification that QUD is simply a stack.

²A partially ordered stack would be a structure where elements can be pushed and popped, but which only has a partial ordering. For example, more than one element can be topmost on the stack.

Ginzburg thus uses a single structure to do two jobs: (1) specifying the questions that are currently available for discussion (“open” questions), and (2) specifying the questions that can be addressed by a short answer (namely, those that are QUD-maximal).

Based on Ginzburg’s QUD querying protocol (see Larsson *et al.* (2002)), Ginzburg’s QUD can also be said to (3) represent questions which have been explicitly raised in the dialogue. While this is not explicitly stated, it appears that the only way a task-level question can enter QUD on Ginzburg’s account is by being explicitly asked. (However, grounding-related questions may enter QUD without being raised as part of the internal reasoning of a DP; see Section 2.2.2).

Similarly, the QUD downdate protocol (see Larsson *et al.* (2002)) suggests that QUD also fulfills a further property (4) of containing as-yet unresolved questions. Openness and unresolvedness may not be identical properties; arguably, resolved questions may still to some extent be open for discussion, and a question could be discarded from the open issues without being resolved, e.g. if it becomes irrelevant (Larsson, 1998).

To summarize, given this basic characterization of QUD, we can say that questions on QUD are

1. open for discussion,
2. available for ellipsis resolution,
3. explicitly raised, and
4. not yet resolved.

In GODIS2, the implemented QUD essentially fits with Ginzburg’s definition, except for the simplification that it is a plain stack rather than an open, partially ordered stack. This is sufficient for the relatively system-controlled, rigid dialogue handled by GODIS2. When the dialogue structure becomes more flexible, however, these various properties of the QUD listed above no longer appear to co-occur in all situations.

3.3.2 Open questions not available for ellipsis resolution

Regarding QUD as a stack (or stack-like) structure suggests that when the topmost element (or set of elements) is popped off the stack, the element (or set of elements) that was previously next-to-maximal becomes maximal. This implies that questions can be answered elliptically at an arbitrary distance from when they were raised. However, it can be argued that in many cases a question which has been raised a few turns back is no longer available for ellipsis resolution (or at least significantly less available than it was right after the question was raised). For example,

B's final utterance in the made-up dialogue in (1) is unlikely to occur and it would be rather confusing if it did, simply because it is not clear which question B is answering.

- (1) A : Who's coming to the party?
B : That depends, is Jill coming
A : Jill Jennings?
B : Yes

A : By the way, did you hear about her brother? What's his name anyway?

B : Umm.. I'm not sure. Anyway, I'd rather not talk about it.
A : OK. So, No
B : So, Jim

If this argument is accepted, we see that a question may satisfy requirements (1) and (3) above, to be a currently open for discussion, explicitly raised question, while not satisfying property (2) of being available for ellipsis resolution.

3.3.3 Open but not explicitly raised questions

Studying recorded travel agency dialogues in light of the QUD approach indicates that it may be the case that a question which has not been (explicitly) raised is in fact discussable, and even available for ellipsis resolution, as the dialogue in (2) shows³.

- (2) A : When do you want to travel?
B : April, as cheap as possible

Thus, we can observe that a question may satisfy requirements (1) of being open for discussion and (2) of being available for ellipsis resolution, without satisfying property (3) of having been explicitly raised.

3.3.4 Global and local QUD

The observations above suggest that it is not ideal to model QUD using a single structure satisfying properties (1) to (4). The solution we propose is to divide QUD into a global and a local

³This is a simplified version of the dialogue in example 4.6.

structure; the former satisfying property (1) of being open for discussion and (4) of being unresolved, and the latter satisfying property (2) of being available for ellipsis resolution. Property (3) of being explicitly raised is not satisfied by either structure. This enables more flexible ways of introducing questions into a dialogue. This division of labour also appears to allow the use of simpler data structures than partially ordered sets.

Definition of local QUD

For the local QUD, a set seems appropriate for modelling the questions currently available for ellipsis resolution. A stack-like structure would suggest e.g. the (made-up) dialogues (3) should be easily processed by DPs, but in fact it is very unclear what B means.

- (3) A : Where are you going? Where is your wife going?
B : Paris. London.

Also, consider example (4):

- (4) A : When are you leaving? When are you coming back?
B : ten thirty and eleven thirty

A simple stack structure also suggests a very unintuitive interpretation of B's answer, where 10:30 is the time when B is coming back and 11:30 is the time when B is leaving. It appears that among the constraints guiding ellipsis resolution in cases where multiple questions are available, the order in which the questions were asked is not very significant. Of course, Ginzburg realizes this and this appears to be the main reason for letting QUD be a partially ordered set where several internally unordered elements may be topmost on QUD, and thus available for ellipsis resolution.

In GODIS3 we define QUD to be an open stack of questions that can be addressed using short answers. The reason for using an open stack is that it has the set-like properties we want, but also retains a stack structure in case it should be useful for ellipsis resolution.

Definition of global QUD, or 'Live Issues'

The global QUD contains all questions which have been raised in a dialogue (explicitly or implicitly) but not yet resolved. It thus contains a collection of current, or "live" issues. A suitable data structure appears to be an open stack, i.e. a stack where non-topmost elements can be accessed. This allows a non-rigid modelling of current issues and task-related dialogue structure.

3.3.5 Some other notions of what a QUD might be

In fact, there are some additional notions of what QUD might be, all of which in some sense contain questions that are under discussion, and all of which have potential uses in a theory of dialogue management and in a dialogue system.

- closed issues: questions that have been raised and resolved (see Section 2.6.9)
- raisable domain issues: all issues potentially relevant in regard to the domain
- potential grounding issues: all issues pertaining to grounding of (a) recent utterance(s)
- resolvable issues (for a DP): all issues that a DP knows some way of dealing with, either by answering directly or by entering a subdialogue

However, while all these may be useful, it may not be necessary to model them explicitly as separate structures in a dialogue system. For example, “raisable domain issues” and “resolvable issues” may be derived from the (static) domain knowledge.

Regarding “closed issues”, we can to some extent derive them from the shared commitments by checking which issues are resolved by propositional information, as in (5).

- (5) Q is a closed issue iff there is some $P \in \text{/SHARED/COM}$ such that P resolves Q and P does not resolve any other question (in the domain)

However, this only works as long as each proposition **resolves** a unique issue. If this is not true, a separate store of closed issues is needed e.g. for detecting reraisings of previously discussed issues.

3.4 Question Accommodation

In this section, we introduce the concept of accommodation and show how it can be extended to handle accommodation of questions in various ways. We also show how question accommodation can be implemented in GODIS.

3.4.1 Background: Accommodation

Lewis' notion of accommodation

David Lewis, in Lewis (1979), in discussing the concept of a conversational scoreboard, compares conversation to a baseball game:

...conversational score does tend to evolve in such a way as is required in order to make whatever occurs count as correct play (Lewis, 1979, p. 347)

He also provides a general scheme for rules of *accommodation* for conversational score:

If at time t something is said that requires component s_n of conversational score to have a value in the range r if what is said is to be true, or otherwise acceptable; and if s_n does not have a value in the range r just before t ; and if such-and-such further conditions hold; then at t the score-component s_n takes some value in the range r .
(Lewis, 1979, p. 347)

This very general schema can be used for dealing with definite descriptions, presupposition projection (see e.g. Van Der Sandt, 1992), anaphora resolution, and many other pragmatic and semantic problems.

One motivation for thinking in terms of accommodation has to do with generality. We could associate expressions which introduce a presupposition as being ambiguous between a presuppositional reading and a similar reading where what is the presupposition is part of what is asserted. For example, an utterance of "The king of France is bald" can be understood either as an assertion of the proposition that there is a king of France and he is bald, or as an assertion of the proposition that he is bald with the presupposition that there is a king of France and that "he" refers to that individual. However, if we assume that accommodation takes place before the integration of the information expressed by the utterance then we can say that the utterance always has the same interpretation.

3.4.2 Accommodation, interpretation, and tacit moves

In an information update framework, accommodation is naturally implemented as an update rule which modifies the information state to include the information presupposed by an utterance, in

such a way as to make the utterance felicitous, i.e. to make it possible to understand the relevance of and possibly integrate the move(s) associated with the utterance. The accommodation update acts as a replacement for a dialogue move, which would have prepared the (common) ground for the utterance actually performed. For this reason, accommodation updates may be referred to as a kind of *tacit move*. For example, the silent accommodation move which adds “there is a king of France” to allow the integration of “The king of France is bald” corresponds to a dialogue move asserting this proposition.

Thus, we can simplify our dialogue move analysis so that the updates to the information state normally associated with a dialogue move are actually carried out by tacit accommodation moves.

This fits well with the fact that very few (if any) effects of a dialogue move are guaranteed as a consequence of performing the move; rather, the actual resulting updates depend on reasoning by the addressed participant. Accommodation is one type of reasoning involved in understanding and integrating the effects of dialogue moves.

3.4.3 Extending the notion of accommodation

In this section, we extend the notion of accommodation introduced by Lewis to cover accommodation of Questions Under Discussion.

As defined by Lewis, accommodation is not limited to only propositions⁴. It states that any component of the scoreboard can be modified by accommodation. If we carry this over to the issue-based approach to dialogue management, it follows that in addition to the accommodation of propositions to the set of jointly committed propositions, questions can be accommodated to QUD.

Thus, question accommodation can be exploited to provide an explanation of the fact that questions can be addressed (even elliptically) without having been explicitly raised. This is very relevant for a dialogue system, since it allows the user more freedom regarding when and how to provide information to the system. In addition, the related concept of question *reaccommodation* can be used to enable addressing resolved issues, which among other things provides a way of handling revision of jointly committed propositions in a principled manner.

Before proceeding to explore the exact formulation and formalization of question accommodation, we provide a rough characterization of the notions used:

⁴Of course, all information on the DGB is, in the end, propositional in nature; a DGB containing a question Q at the top of QUD could in principle be described by a set of propositions including ‘Q is topmost on QUD’. This, however, would be impractical and inefficient compared to maintaining a proper stack-like structure.

- question/issue accommodation: adjustments of common ground required to understand an utterance addressing an issue which has not been raised, but which is
 - relevant to the current dialogue plan
 - relevant to some issue in the domain
- question/issue reaccommodation: adjustments of common ground required to understand an utterance addressing an issue which has been resolved and
 - does not influence any other resolved issue
 - influences another resolved issue
 - concerns grounding of a previous utterance

Utterances which are relevant to the current dialogue plan can also be regarded as being *indirectly relevant* to the goal of that plan. In inquiry-oriented dialogue we model goals as issues which allows an alternative formulation of accommodation as “adjustments of common ground required for understanding an utterance addressing an issue which has not been raised, but which is (directly or indirectly) relevant to some issue in the domain.” In action-oriented dialogue (Chapter 4), utterances may also be indirectly relevant to some goal action.

3.5 Formalizing question accommodation

In this section we discuss the various types of question accommodation and show how they are formalized in GODIS3. We start by explaining and motivating some modifications of the information state type required to handle dialogues involving question accommodation.

3.5.1 Information state in GODIS3

The information state used in GODIS3 is shown in Figure 3.1.

The first change compared to the GODIS2 information state is the addition of the open stack /SHARED/ISSUES, which contains the open issues. The /SHARED/QUD field has not been modified in terms of data type, but is now used for modelling the local QUD.

The second change is the division of /SHARED/TMP into two subfields. The SYS subfield corresponds to the TMP field in GODIS2, and contains parts of the information state copied right before integrating the latest system utterance. As in GODIS2, system utterances are optimistically assumed to be grounded, and if the user gives negative feedback the TMP/SYS field is used

$$\begin{array}{l}
\left[\begin{array}{l} \text{PRIVATE} : \\ \\ \text{SHARED} : \end{array} \left[\begin{array}{l} \text{AGENDA} : \text{OpenQueue}(\text{Action}) \\ \text{PLAN} : \text{OpenStack}(\text{PlanConstruct}) \\ \text{BEL} : \text{Set}(\text{Prop}) \\ \text{TMP} : \left[\begin{array}{l} \text{USR} : \text{Tmp} \\ \text{SYS} : \text{Tmp} \end{array} \right] \\ \text{NIM} : \text{OpenQueue}(\text{Pair}(\text{DP}, \text{Move})) \\ \\ \text{COM} : \text{Set}(\text{Prop}) \\ \text{ISSUES} : \text{OpenStack}(\text{Question}) \\ \text{QUD} : \text{OpenStack}(\text{Question}) \\ \text{PM} : \text{OpenQueue}(\text{Move}) \\ \text{LU} : \left[\begin{array}{l} \text{SPEAKER} : \text{Participant} \\ \text{MOVES} : \text{Set}(\text{Move}) \end{array} \right] \end{array} \right] \right] \\
\\
\text{Tmp} = \left[\begin{array}{l} \text{COM} : \text{Set}(\text{Prop}) \\ \text{ISSUES} : \text{OpenStack}(\text{Question}) \\ \text{QUD} : \text{OpenStack}(\text{Question}) \\ \text{AGENDA} : \text{OpenQueue}(\text{Action}) \\ \text{PLAN} : \text{OpenStack}(\text{PlanConstruct}) \end{array} \right]
\end{array}$$

Figure 3.1: GoDIS3 Information State type

to retract the optimistic assumption. In addition, the system sometimes makes an optimistic assumption regarding the grounding and understanding of a user utterance, and produces positive feedback (e.g. “OK. To Paris.”). In GoDIS3, we will use a type of question accommodation to enable retraction of the optimistic grounding assumption regarding user utterances, in cases where the user rejects the system’s reported interpretation. For this, we also need to keep a copy of relevant parts of the information state as they were right before the user’s utterance was interpreted and integrated; this is what the TMP/USR field contains.

Finally, the items on /PRIVATE/NIM are now pairs, where the first element is the DP who made the move, and the second is the move itself. In GoDIS2, it can be assumed that all non-integrated moves were performed in the latest utterance. In GoDIS3, question accommodation mechanisms allow less restricted dialogues, and there is no longer any guarantee that all non-integrated moves were made in the latest utterance. Moves may be stored in NIM for several turns before being integrated.

3.6 Varieties of question accommodation and reaccommodation

As shown by the dialogue in (6)⁵, questions can be answered (even elliptically) without previously having been raised.

- (6) J : vicken månad ska du åka
 what month do you want to go
 B : ja: typ den: ä: tredje fjärde april / nån gång där
 well around 3rd 4th april / some time there
 P : så billit som möjlit
 as cheap as possible

But where does the accommodated question come from? In principle, we could imagine a huge number of possible questions associated with any answer, especially if it is elliptical or semantically underspecified. How is this search space constrained? The answer lies in the activity which is being performed; the question must be available as part of the knowledge associated with the activity - either static knowledge describing how the activity is typically performed, or dynamic knowledge of the current state of the activity.

In this section we first describe the three basic question accommodation mechanisms: global question accommodation (issue accommodation), local question accommodation (QUD accommodation) and dependent issue accommodation. We then discuss the need for clarification questions in cases where it is not clear which question is being addressed, before moving on to describing reaccommodation and dependent reaccommodation. For each type of accommodation we also describe the implementation and provide dialogue examples from the implemented system.

In general, accommodation is tried only after “normal” integration has failed. The coordination of the accommodation rules in relation to grounding (including integration) rules is handled by the update algorithm described in Section 3.7.2.

⁵This dialogue has been collected by the University of Lund as part of the SDS project. We quote the transcription done in Göteborg as part of the same project.

3.6.1 Issue accommodation: from dialogue plan to ISSUES

This type of accommodation occurs when a DP addresses an issue which is not yet open but which is part of the current plan⁶. In the dialogue in example 6, P's second utterance ("as cheap as possible") addresses the issue of which price class P is interested in. At this stage of the dialogue, this issue has not been raised, but presumably J was planning to raise it eventually.

Before GODIS can integrate an answer, it needs to find an open issue to which the answer is relevant (see the definition of the **integrateUtrAnswer** rule in Section 2.6.6). Thus, to handle a dialogue like that in example 6 some mechanism is needed for finding an appropriate issue in the current dialogue plan and moving it to the ISSUES stack. A schematic representation of issue accommodation is shown in Figure 3.2.

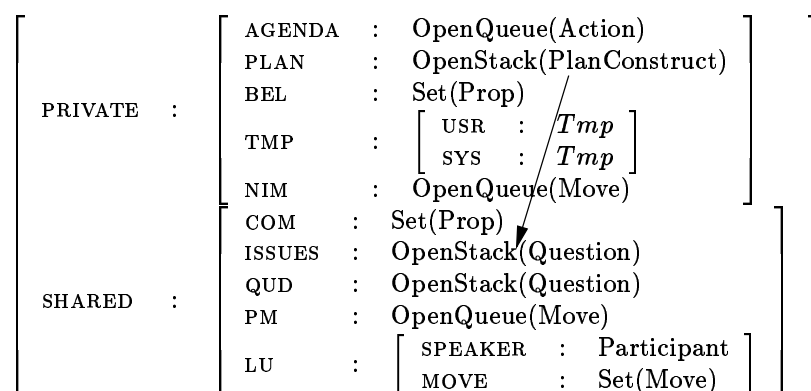


Figure 3.2: Issue accommodation

The issue accommodation update rule in (RULE 3.1) first checks whether a question which matches the answer occurs in the current dialogue plan (provided there is one). A question matches an answer if the answer is **relevant** to, or (in Ginzburg's terminology) **about** the question. If such a question can be found, it can be assumed that this is now an open issue. Accommodating this amounts to pushing the question on the ISSUES stack.

⁶Since the current plan is presumably being carried out in order to deal with some open issue, we may regard the utterance as indirectly relevant to some open issue (via the plan).

(RULE 3.1) RULE: **accommodatePlan2Issues**
 CLASS: **accommodate**
 PRE: {
 \$/PRIVATE/NIM/ELEM/SND = **answer**(*A*)
 not \$LEXICON :: yn_answer(*A*)
 in(\$/PRIVATE/PLAN, **findout**(*Q*))
 \$DOMAIN :: relevant(*A*, *Q*)
 \$DOMAIN :: default_question(*Q*) or
 not (in(\$/PRIVATE/PLAN, **findout**(*Q'*))
 and $Q \neq C$
 and \$DOMAIN :: relevant(*A*, *Q'*))
 EFF: { push(/SHARED/ISSUES, *B*)

The first condition picks out a non-integrated **answer** move with content *A*. The second condition checks that *A* is not a *y/n* answer (e.g. **yes**, **no**, **maybe** etc.), and thus implements an assumption that such answers cannot trigger question accommodation, since they are too ambiguous⁷. The third and fourth conditions check if there is a **findout** action with content *Q* in the currently loaded plan, such that *A* is **relevant** to *Q*. The final condition checks that there is no other question in the plan that the answer is **relevant** to, or alternatively that *Q* has the status of a default question. If this condition does not hold, a clarification question should be raised by the system; this is described in Section 3.6.3. The “default question” option allows encoding of the fact that one issue may be significantly more salient in a certain domain. For example, in a travel agency setting the destination city may be regarded as more salient than the departure city question. If this is encoded as a default question, then if the user says simply “Paris” it is interpreted as answering the destination city question; no clarification is triggered⁸.

Example dialogue: issue accommodation The dialogue in (DIALOGUE 3.1) illustrates accommodation of the question ?*C.class*(*C*) from the plan to the stack of open issues.

(DIALOGUE 3.1)

S> What month do you want to leave?

⁷However, in general one cannot rule out the possibility that *y/n* answers can trigger accommodation in severely restricted domains. The assumption that this cannot happen can be regarded as a very simplified version of a constraint on the number of questions which an answer may be relevant without making question accommodation infeasible.

⁸The normal grounding mechanisms should of course enable correction of this assumption. In GODIS3 the choice of grounding strategy depends solely on the recognition score which means that a high-scoring answer may be interpreted as an answer to a default question and not receive any explicit feedback. This is one case which indicates a need for taking more factors into account when choosing feedback and grounding strategy.

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \langle \rangle \\ \text{PLAN} = \left\langle \begin{array}{l} \text{findout}(?A.\text{month}(A)) \\ \text{findout}(?B.\text{dept_day}(B)) \\ \text{findout}(?C.\text{class}(C)) \\ \text{consultDB}(?D.\text{price}(D)) \end{array} \right\rangle \\ \text{BEL} = \{\} \\ \text{NIM} = \langle \rangle \\ \text{COM} = \left\{ \begin{array}{l} \text{dest_city}(\text{paris}) \\ \text{dept_city}(\text{london}) \\ \text{how}(\text{plane}) \end{array} \right\} \\ \text{ISSUES} = \left\langle \begin{array}{l} ?F.\text{month}(F) \\ ?E.\text{price}(E) \end{array} \right\rangle \\ \text{QUD} = \langle ?F.\text{month}(F) \rangle \\ \text{PM} = \dots \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{sys} \\ \text{MOVES} = \text{ask}(?F.\text{month}(F)) \\ \text{SCORE} = 1 \end{array} \right] \end{array} \right] \right]$$

U> april as cheap as possible

```

getLatestMoves
backupSharedUsr
integrateUsrShortAnswer
downdateISSUES
removeFindout
accommodatePlan2Issues
{ push(/SHARED/ISSUES, ?A.class(A))
integrateUsrFullAnswer
downdateISSUES
removeFindout
downdateQUD

```

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \langle \langle \text{icm:acc*pos} \rangle \rangle \\ \text{PLAN} = \left\langle \begin{array}{l} \text{findout}(?A.\text{dept_day}(A)) \\ \text{consultDB}(?B.\text{price}(B)) \end{array} \right\rangle \\ \text{BEL} = \{\} \\ \text{NIM} = \langle \rangle \\ \text{COM} = \left\{ \begin{array}{l} \text{class}(\text{economy}) \\ \text{month}(\text{april}) \\ \text{dest_city}(\text{paris}) \\ \text{dept_city}(\text{london}) \\ \text{how}(\text{plane}) \end{array} \right\} \\ \text{ISSUES} = \langle ?D.\text{price}(D) \rangle \\ \text{QUD} = \langle \rangle \\ \text{PM} = \langle \langle \text{icm:acc*pos}, \text{icm:loadplan}, \text{ask}(?C.\text{month}(C)) \rangle \rangle \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{usr} \\ \text{MOVES} = \langle \langle \text{answer}(\text{april}), \text{answer}(\text{class}(\text{economy})) \rangle \rangle \\ \text{SCORE} = 1 \end{array} \right] \end{array} \right] \right]$$

S> Okay. What day do you want to leave?

3.6.2 Local question accommodation: from ISSUES to QUD

If a move with underspecified content is made which does not match any question on the QUD, the closest place to look for such a question is ISSUES, and if it can be found there it should be pushed on the local QUD to enable ellipsis resolution. As a side-effect, the question has now been brought into focus and should, if it is not topmost on the open issues stack, be raised to the top of open issues. A schematic overview of local question accommodation is shown in Figure 3.3.

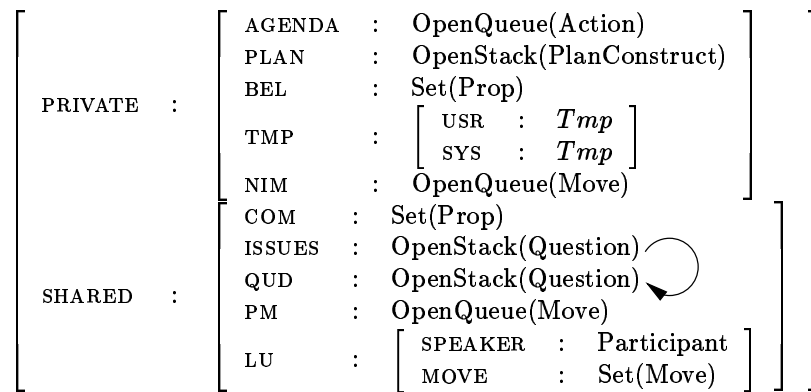


Figure 3.3: Local question accommodation

This type of accommodation can e.g. occur if a question which was raised previously has dropped off the local QUD but has not yet been resolved and remains on ISSUES. It should also be noted that several accommodation steps can be taken during the processing of a single utterance; for example, if an issue that is in the plan but has not yet been raised is answered elliptically.

(RULE 3.2) **RULE: accommodateIssues2QUD**
 CLASS: accommodate
 $\left\{ \begin{array}{l} \$/PRIVATE/NIM/ELEM=usr_answer(A) \\ \$DOMAIN :: short_answer(A) \\ not \$LEXICON :: yn_answer(A) \end{array} \right.$
 PRE: $\left\{ \begin{array}{l} in(\$/SHARED/ISSUES, Q) \\ not\ in(\$/SHARED/QUD, Q) \\ \$DOMAIN :: relevant(A, Q) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} push(\$/SHARED/QUD, Q) \\ raise(\$/SHARED/ISSUES, Q) \end{array} \right.$

The second condition in (RULE 3.2) checks that the content of the **answer** move picked out by condition 1 is semantically underspecified. The third condition imposes a constraint on local question accommodation, excluding short answers to *y/n*-questions (“yes”, “no”, “maybe” etc.). The remaining conditions check that the answer-content is **relevant** to an issue which is on ISSUES but not on QUD. The first operation pushes the accommodated question on QUD, and the final update raises the question to the top of the stack of open issues.

3.6.3 Issue clarification

In GoDIS2, user answers are either pragmatically relevant to the question topmost on QUD, or not relevant at all. When we add mechanisms of accommodation to allow for answers to unraised questions, it becomes necessary to deal with cases where an answer may be potentially relevant to several different questions.

Semantically underspecified answers may (but need not) be pragmatically ambiguous, i.e. it is not clear what question they provide an answer to. This can be resolved by asking the speaker what question she intended to answer (or equivalently, which proposition she wanted to convey).

In this case, we can use the same strategy as for negative grounding, i.e. when a pragmatically ambiguous utterance is to be interpreted the system raises a question whose answer will be integrated instead of the ambiguous answer. For example, “Paris” may be **relevant** to either the destination city question or the departure city question. When the clarification question “Do you mean from Paris or to Paris?” is raised it is expected that the user will answer this question, which means that the ambiguous answer no longer needs to be integrated and can be thrown away⁹.

In this way we see how question accommodation, amended with a mechanism for resolving which question to accommodate, can be used to resolve pragmatic ambiguities in user input. The accommodation mechanism can thus be regarded as a refinement of the account of grounding on the understanding level put forward in Chapter 2. The rule which selects the issue clarification issue is shown in (RULE 3.3).

⁹GoDIS3 only handles full answers to clarification questions, i.e. “To Paris.” or “From Paris.”. A slightly more advanced semantics would be required to handle cases where the user again gives an underspecified response which **resolves** the question, i.e. “To.” or “From.”.

(RULE 3.3) RULE: **clarifyIssue**
 CLASS: **select_action**
 PRE: $\left\{ \begin{array}{l} \text{in}(\$/PRIVATE/NIM, \text{usr-answer}(A)) \\ \text{setof}(C, \text{in}(\$/PRIVATE/PLAN, \text{findout}(Q)) \text{ and} \\ \quad \$DOMAIN :: \text{relevant}(A, Q), QSet) \\ \$\$arity(QSet) > 1 \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} ! \text{setof}(?P, \text{in}(QSet, Q) \text{ and } \$DOMAIN :: \text{combine}(Q, A, P), AltQ) \\ \text{push}(/PRIVATE/AGENDA, \text{findout}(AltQ)) \\ \text{del}(/PRIVATE/NIM, \text{usr-answer}(A)) \end{array} \right.$

The first condition picks out the **answer**-move from the NIM queue. The second and third conditions check that there is more than one question in the plan to which the answer is **relevant**, by constructing the set of such questions. The first operation constructs the alternative-question by applying each question in the set constructed in condition 2 to the answer to get a proposition and prefixing the question operator '?' to each proposition to get a *y/n*-question. The alternative question is this set of *y/n*-questions. The second operation pushes the action to raise the alternative question on the agenda, and the final operation removes the **answer** move from NIM; this is motivated above.

A sample dialogue with issue clarification is shown in (DIALOGUE 3.2).

(DIALOGUE 3.2)

```
S> Welcome to the travel agency!
U> price information please
S> Okay. I need some information. How do you want to travel?
U> flight um paris
S> OK, by flight. Do you mean from paris or to paris?
```

The user's utterance of "paris" is interpreted as **answer(paris)**, which is **relevant** to two questions in the plan: $?x.\text{dest_city}(x)$ and $?x.\text{dept_city}(x)$. Because of this, the issue accommodation rule in 1 will not fire and the answer is not integrated. This allows the **clarifyIssue** rule to fire in the selection phase. By combining each of these questions with the content of the answer (**paris**), and turning each resulting proposition into a *y/n*-question, the set $\{?\text{dest_city}(\text{paris}), ?\text{dest_city}(\text{paris})\}$ is obtained. This set also works as an alternative-question, which is used as the content of the clarification question in the system's final utterance in (6).

Note that these clarification questions are dynamically put together by the system and thus do not need to be pre-programmed. This means that the application designer does not even need to realize that an ambiguity exists.

3.6.4 Dependent issue accommodation: from domain resource to ISSUES

Issue accommodation, introduced above, presupposes that there is a current plan in which to look for an appropriate question; this, in turn, presupposes that there is some issue under discussion which the plan is meant to deal with. But what if there is currently no plan?

In this case, it may be necessary to look in at the set of stored domain-specific dialogue plans (or come up with a new plan) to try to figure out which issue the latest utterance was addressing; this can be regarded as a simple kind of plan recognition. An appropriate plan should contain a question matching some information provided in the latest utterance. If such a plan is found, it is possible that, in addition to the question answered by the latest utterance, a further issue should also be accommodated: the “goal-issue” which the plan in question is aimed at dealing with. Given our definition of dependence between questions in Larsson *et al.* (2002), the goal issue is dependent on the issue directly addressed, and hence we refer to this as dependent issue accommodation.

Dependent issue accommodation is thus the process of finding an appropriate background issue and a plan for dealing with that issue which makes the latest utterance relevant, given “normal” global issue accommodation. That is, dependent issue accommodation is always followed by global issue accommodation. Dependent issue accommodation applies when no issues are under discussion, and a previously unraised question is answered using a full or short answer (in the latter case, global issue accommodation must in turn be followed by local question accommodation). A schematic overview of dependent issue accommodation is shown in Figure 3.4, and the update rule is shown in (7).

(RULE 3.4) **RULE: accommodateDependentIssue**
 CLASS: accommodate
 PRE: {
 setof(A , $\$/PRIVATE/NIM/ELEM/SND=answer(A)$, $AnsSet$)
 $\$Sarity(AnsSet) > 0$
 is_empty($\$/PRIVATE/PLAN$)
 $\$DOMAIN :: plan(DepQ, Plan)$
 forall(in($AnsSet$, A), in($Plan$, findout(Q)) and
 $\$DOMAIN :: relevant(A, Q)$)
 not ($\$DOMAIN :: plan(DepQ', Plan')$ and $DepQ' \neq DepQ$ and
 forall(in($AnswerSet$, A), in($Plan'$, findout(Q)) and
 $\$DOMAIN :: relevant(A, Q)$))
 not in($\$/PRIVATE/AGENDA$, icm:und*int:usr*issue($DepQ$))
 EFF: {
 push($\$/SHARED/ISSUES$, $DepQ$)
 push($\$/PRIVATE/AGENDA$, icm:accommodate: $DepQ$)
 push($\$/PRIVATE/AGENDA$, icm:und*pos:usr*issue($DepQ$))
 set($\$/PRIVATE/PLAN$, $Plan$)
 push($\$/PRIVATE/AGENDA$, icm:loadplan)

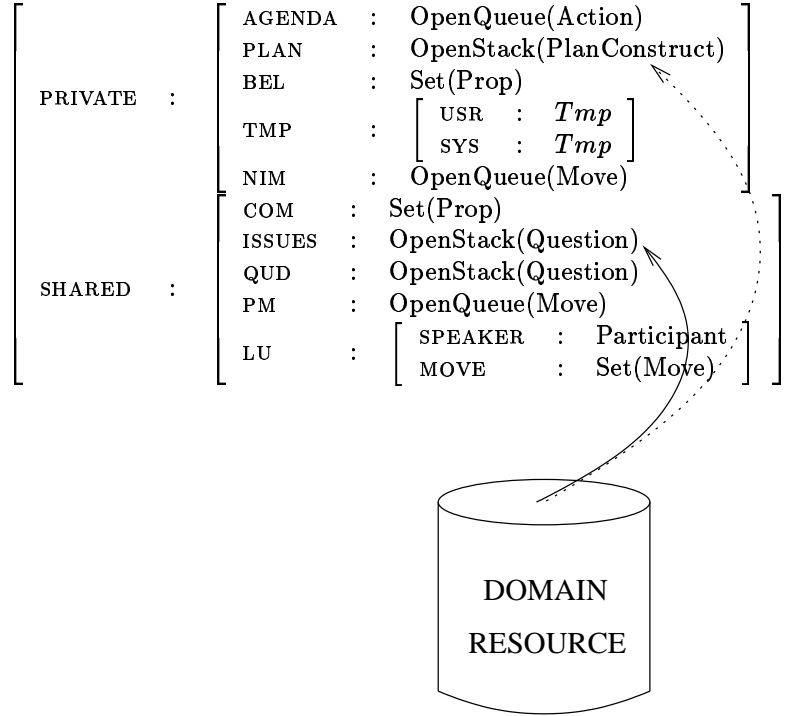


Figure 3.4: Dependent issue accommodation

The first two conditions construct a set of all non-integrated answers and check that the arity of this set is larger than zero, i.e. that there is at least one non-integrated answer. It should be noted that this formulation of the rule relies on the assumption that all unintegrated answers have been provided by the user. This is true for GODIS3, since all system answers are integrated immediately and never need accommodation. However, in a more complex system this may not always be true; in this case, the rule would need some slight modifications to only pick out user moves.

The third condition checks that the plan is empty. The consequence of this is that dependent issue accommodation is not available when some plan is being executed, so if an issue is being dealt with the only way to raise a new issue is to do so explicitly. We believe this is a reasonable restriction, but if desired it can be disabled by removing the condition. However, doing so may give problems in case speech recognition mistakenly recognizes an answer not matching the current plan; if this answer triggers dependent accommodation this may result in confusing utterances from the system.

The fourth and fifth conditions look for a plan in the domain resource to which all non-integrated answers are **relevant**. This can be regarded as a simple version of plan recognition: given an observed set of actions (user answers), try to find a plan and a goal (an issue) such that the actions fit the plan. Here, the user answers fit the plan by being **relevant** answers to questions in the plan (more precisely, questions such that the plan includes actions to resolve them).

The final condition checks that there is only one plan to which all the answers are **relevant**. If there are several such plans, the accommodation rule should not trigger; instead, a clarification question should be raised by the system (see Section 3.6.5).

The updates push the dependent issue on ISSUES, loads the plan, and pushes the appropriate ICM moves on the agenda: positive feedback concerning the accommodated issue (“You want to know about price.”) and feedback indicating that a new plan has been loaded (“I need some information”). In addition, ICM indicating accommodation is produced (see Section 3.7.1).

(DIALOGUE 3.3)

S> Welcome to the travel agency!

U> i want a flight

getLatestMoves

backupSharedUsr

accommodateDependentIssue

```
{
  push(/SHARED/ISSUES, ?C.price(C))
  push(/PRIVATE/AGENDA, icm:accommodate:?C.price(C))
  push(/PRIVATE/AGENDA, icm:und*pos:usr*issue(?C.price(C)))
  set(/PRIVATE/PLAN, stackset([findout(?D.how(D)), findout(?E.dest_city(E)), ... ]))
  push(/PRIVATE/AGENDA, icm:loadplan)
```

accommodatePlan2Issues

```
{ push(/SHARED/ISSUES, ?A.how(A))
```

accommodateIssues2QUD

```
{ push(/SHARED/QUD, ?A.how(A))
  raise(/SHARED/ISSUES, ?A.how(A))
```

integrateUsrAnswer

downdateISSUES

removeFindout

downdateQUD

backupSharedSys

selectIcmOther

selectIcmOther

PRIVATE	=	AGENDA	=	⟨⟨ icm:loadplan, icm:acc*pos ⟩⟩]
				findout(?A.dest_city(A))	
				findout(?B.dept_city(B))	
				findout(?C.month(C))	
				findout(?D.dept_day(D))	
		PLAN	=	⟨]
				findout(?E.class(E))	
				consultDB(?F.price(F))	
				⟩	
				⟩	
		BEL	=	...]
		NIM	=	...	
		COM	=	{ how(plane) }	
		ISSUES	=	{ ?G.price(G) }	
		QUD	=	⟨⟩	
SHARED	=	PM	=	⟨⟨ greet ⟩⟩]
				SPEAKER = usr	
				MOVES = ⟨⟨ answer(plane) ⟩⟩	
				SCORE = 1	

S> Alright. You want to know about price.

U>

S> I need some information. Okay. By flight. What city do you want to go to?

The current solution has an optimistic strategy for dependent accommodation: the issue is assumed to be under discussion and the system gives explicit positive feedback of this assumption. It may be argued that a pessimistic strategy is more appropriate for dependent accommodation; this can be achieved by replacing the list of updates in 4 with the update in (7).

(7) push(/PRIVATE/AGENDA, icm:und*int:usr*issue(D))

This will provide interrogative feedback from the system concerning whether the dependent issue should be opened, e.g. “You want to know about price, is that correct?”. If the user gives a positive response to this feedback, the system will use the same update rules as usual for integrating the user’s response to interrogative feedback.

(DIALOGUE 3.4)

S> Welcome to the travel agency!

U> i want a flight

getLatestMoves
backupSharedUsr


```

accommodateDependentIssue
{ push(/PRIVATE/AGENDA, icm:und*int:usr*issue(?C.price(C)))
downdateQUD
backupSharedSys
selectIcmUndNeg
selectIcmOther

```

S> flight. I dont quite understand. You want to know about price, is that correct?

```

getLatestMoves
integrateOtherICM
integrateOtherICM
integrateUndIntICM

```

U> yes

```

getLatestMoves
integratePosIcmAnswer
findPlan
accommodatePlan2Issues
accommodateIssues2QUD
integrateUsrAnswer
downdateQUD

```

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{ll} \text{AGENDA} & = \langle \langle \text{icm:loadplan, icm:und*int:usr*how(plane)} \rangle \rangle \\ & \text{findout(?A.how(A))} \\ & \text{findout(?B.dest_city(B))} \\ & \text{findout(?C.dept_city(C))} \\ \text{PLAN} & = \langle \begin{array}{l} \text{findout(?D.month(D))} \\ \text{findout(?E.dept_day(E))} \\ \text{findout(?F.class(F))} \\ \text{consultDB(?G.price(G))} \end{array} \rangle \\ \text{BEL} & = \{ \} \\ \text{NIM} & = \langle \langle \rangle \rangle \\ \text{COM} & = \{ \} \\ \text{ISSUES} & = \langle \begin{array}{l} ?A.how(A) \\ ?H.price(H) \end{array} \rangle \\ \text{QUD} & = \langle \rangle \\ \text{PM} & = \langle \langle \text{icm:sem*pos:answer(plane), ...} \rangle \rangle \\ \text{LU} & = \left[\begin{array}{ll} \text{SPEAKER} & = \text{usr} \\ \text{MOVES} & = \text{oqueueanswer(yes)} \\ \text{SCORE} & = 1 \end{array} \right] \end{array} \right]$$

```

backupSharedSys
selectIcmOther

```

selectIcmOther

S> I need some information. by flight , is that correct?

3.6.5 Dependent issue clarification

If no plan is loaded and one or several non-integrated answers are relevant to several plans, a clarification question should be raised by the system to find out which issue the user wants the system to deal with. This is done by the selection rule in (RULE 3.5).

(RULE 3.5) RULE: **clarifyDependentIssue**
 CLASS: **select_action**
 PRE: {
 in(\$/PRIVATE/NIM, pair(usr, answer(*A*)))
 setof(*Q'*, \$DOMAIN :: plan(*Q'*, *Plan*) and
 in(*Plan*, findout(*SomeQ*)) and
 \$DOMAIN :: relevant(*A*, *SomeQ*),
 $QSet'$)
 remove_unifiables($QSet'$, $QSet$)
 \$\$arity($QSet$) > 1
 EFF: {
 ! setof(*IssueQ*, in($QSet$, *I*) and *IssueQ*=?issue(*I*), *AltQ*)
 push(/PRIVATE/AGENDA, findout(*AltQ*))

The first condition checks if there is at least one non-integrated user answer left after the system has attempted to integrate the latest user utterance. The second and third conditions constructs the set $QSet$ of dependent issues that the non-integrated answer is indirectly relevant to (i.e. issues for which there is a plan containing an action to resolve a question to which the answer is relevant)¹⁰. The final condition checks that there is more than one such dependent issue.

The first update constructs an alternative-question by picking out each question *I* in $QSet$ and adding ?issue(*I*) to the set which constitutes the alternative-question. The final update pushes an action to resolve alternative-question on the agenda.

In the travel agency domain, an example of dependent issue clarification occurs if the user's first utterance is "to Paris", interpreted as answer(dest_city(paris)). This answer is relevant to the question ?*x*.dest_city(*x*) which occurs in both the plan for addressing the price issue and that for addressing the visa issue. This blocks the dependent issue accommodation rule. In the dialogue in (DIALOGUE 3.5), the system instead raises a clarification question. Note that GODIS3 here

¹⁰The remove_unifiables condition is used to remove multiple occurrences of the same issue. Note that these occurrences are not identical, since they may differ in the identity of variables. One may of course argue whether sets should have this property, but in the current TRINDIKIT implementation they do.

makes use of the fact that an **ask**-move can supply an answer to a question concerning which issue to pursue.

(DIALOGUE 3.5)

S> Welcome to the travel agency!

U> to paris

getLatestMoves
backupSharedUsr
downdateQUD

backupSharedSys
clarifyDependentIssue
 $\left\{ \begin{array}{l} \text{! setof}(E, \text{in}(\text{set}([\text{need_visa}, ?D.\text{price}(D)]), F) \text{ and } E=\text{issue}(F), G) \\ \text{push}(/PRIVATE/AGENDA, \text{findout}(G)) \end{array} \right.$
selectIcmUndNeg
selectAsk

S> to paris. I dont quite understand. Do you mean to ask about visa or to ask about price?

getLatestMoves
integrateOtherICM
integrateOtherICM
integrateSysAsk

U> visa

getLatestMoves
backupSharedUsr
integrateUsrAsk
downdateISSUES
findPlan
accommodatePlan2Issues
integrateUsrAnswer
downdateQUD

S> Okay. I need some information. to paris , is that correct?

3.6.6 Question reaccommodation

In GODiS1 and GODiS2 the user has a limited ability to reraise previously resolved issues; this will typically result in the system giving the same answer again. However, this kind of reraising is not very useful since the user is not able to modify her own answers to the system's previous questions.

Global question reaccommodation (Issue reaccommodation)

In general, if the user provides an alternative resolution of an issue which has been previously resolved, this triggers a reraising of that issue. If the previous answer is incompatible with the new one, the old answer is removed. This allows the user to change his/her mind during the dialogue. Here is an example dialogue with the system:

(DIALOGUE 3.6)

```
S> Welcome to the travel agency!
U> price information please
S> You asked about price. How do you want to travel?
U> a flight, april the fifth
S> by flight. in april. the fifth. Okay. What city do you
    want to go to?
U> london
S> Okay. to london.
U> actually, i want to go on the fourth
S> the fourth. What city do you want to go from?
```

Initially, integration of the answer using **integrateUsrAnswer** (Section 2.6.6) will fail since there is no matching question on ISSUES. The system will then try various accommodation strategies, including accommodation from /SHARED/COM formulated in (RULE 3.6).

```
(RULE 3.6)  RULE: accommodateCom2Issues
            CLASS: accommodate
            PRE: {
                  { $/PRIVATE/NIM/ELEM/SND=answer(A)
                    in($/SHARED/COM, P)
                  }
                  { $DOMAIN :: question(Q)
                    $DOMAIN :: relevant(A, Q)
                    $DOMAIN :: relevant(P, Q)
                  }
            }
            EFF: { push(/SHARED/ISSUES, Q)
```

This accommodation rule looks for an answer A among the moves which have not yet been integrated (first condition). It then looks for a proposition among the shared commitments established in the dialogue so far (second condition) which according to the system's domain resource is an appropriate answer to some question for which A is also an answer (third to fifth conditions). Given that in this simple system answers can only be **relevant** to a single question¹¹, this strategy will be successful in identifying cases where we have two answers to the same question. A system that deals with more complex dialogues where this is not the case would need to keep track of closed issues in a separate list of closed issues. Thus the conditions will succeed if there is a question such that both the user answer and a stored proposition are **relevant** answers to it; in the example dialogue above, "departure date is the fourth" and "departure date is the fifth" are both **relevant** answers to the question "which day do you want to travel?". If such a question is found it is accommodated to ISSUES, that is, it becomes an open issue again.

When **accommodateCom2Issues** has been successfully applied, the retract rule in (RULE 3.7) will remove the incompatible information from the system's view of shared commitments represented in $/\text{SHARED}/\text{COM}$.

(RULE 3.7) RULE: **retract**
 CLASS: **integrate**
 PRE: {
 $\$/\text{PRIVATE}/\text{NIM}/\text{ELEM}/\text{SND}=\text{answer}(A)$
 $\text{in}(\$/\text{SHARED}/\text{COM}, P')$
 $\text{fst}(\$/\text{SHARED}/\text{ISSUES}, Q)$
 $\$/\text{DOMAIN} :: \text{relevant}(P, Q)$
 $\$/\text{DOMAIN} :: \text{relevant}(A, Q)$
 $\$/\text{DOMAIN} :: \text{combine}(Q, A, P)$
 $\$/\text{DOMAIN} :: \text{incompatible}(P, P')$
 EFF: { $\text{del}(\$/\text{SHARED}/\text{COM}, P')$

The conditions here are similar to those in (RULE 4.6). We look for an unintegrated answer (first condition) which is **relevant** to a question at the head of the list of open issues (third and fifth conditions) and for which there is already a **relevant** answer in the shared commitments (second and fourth conditions). Finally, we determine that the result of combining the answer with the question (sixth condition) is incompatible with the answer already found (seventh condition). If all this is true we delete the answer which is currently in the shared commitments. This will finally allow the new answer to be integrated by a rule that integrates an answer from the user, and a further rule will remove the resolved issue from QUD. Note that this rule is of class **integrate**. It is tried before any other integration rule, to avoid integration of conflicting information.

¹¹That is, in the full form in which they appear in $/\text{SHARED}/\text{COM}$. "Chicago" can be an answer to "Which city do you want to go to?" and "Which city do you want to go from?" but when it has been combined with the questions the result will be "destination(Chicago)" and "from(Chicago)" respectively and it is this which is entered into the commitments.

Note also that the “incompatible” relation is defined as a part of the domain resource, and can thus be domain specific. The simple kind of revision that GODIS currently deals with is also handled by some form-based systems (although they usually do not give feedback indicating that information has been removed or replaced, as GODIS does). For example, Chu-Carroll (2000) achieves a similar result by extracting parameter values from the latest user utterance and subsequently (if possible) copying values from the previous form for any parameters not specified in the latest utterance. A similar mechanism is referred to as “overlay” by Alexandersson and Becker (2000). While we are dealing only with very simple revision here, the rule in (RULE 3.7) and the “incompatible” relation can be seen as placeholders for a more sophisticated mechanism of belief revision.

It is also possible to remove the old answer by denying it (asserting its negation) as in (DIALOGUE 3.7).

(DIALOGUE 3.7)

```
S> Welcome to the travel agency!
U> price information for a flight to paris on april the fifth
S> You asked about price.  by flight.  to paris.  in april.  the
fifth.  What city do you want to go from?
U> actually, not the fifth
S> not the fifth.  So, what day do you want to leave?
```

In this case, the system will explicitly reraise the issue to get a new response from the user. Again, the system will use the rule in 25 and signal reraising using “so, ”. All the rules will be applied as in the previous case, but the departure date question will not be removed since it is not resolved by the given answer. Eventually, this leads to the system reraising the question.

Reraising of dependent questions (dependent issue reaccommodation)

In some cases, an issue might be reraised which influences the answer to a further issue that has also been resolved. For example, the choice of price class for a flight influences the price of the flight. In this case, the influenced question also needs to be reaccommodated and answered again.

(DIALOGUE 3.8)

```
S> Welcome to the travel agency!
U> what's the price of a flight from london to paris april the
```

fifth?

S> You asked about price. by flight. from london. to paris. in april. the fifth. What class did you have in mind?

U> as cheap as possible

S> cheap. Okay. The price is 123 crowns.

U> actually, i might go for business class

S> first class. Okay. Concerning your question about price :
The price is 1234 crowns.

The rule that achieves the reraising of a dependent question COM-to-ISSUES accommodation is shown in (RULE 3.8).

(RULE 3.8) **RULE: accommodateCom2IssuesDependent**
 CLASS: accommodate
 PRE: {
 \$/PRIVATE/NIM/ELEM/SND=answer(A)
 in(\$/SHARED/COM, P)
 \$DOMAIN :: question(Q)
 \$DOMAIN :: relevant(A, Q)
 \$DOMAIN :: relevant(P, Q)
 is_empty(\$/SHARED/ISSUES)
 \$DOMAIN :: depends(Q', Q)
 in(\$/SHARED/COM, P')
 \$DOMAIN :: relevant(P', Q')
 EFF: {
 del(/PRIVATE/BEL, P')
 del(/SHARED/COM, P')
 push(/SHARED/ISSUES, Q')
 push(/SHARED/ISSUES, Q)
 push(/PRIVATE/AGENDA, respond(Q'))

This rule is similar to 6 except that it looks for a question which **depends** on the question it finds corresponding to the answer provided by the user. It puts both question onto the list of open issues and plans to respond to the dependent question. This rule, as currently implemented, is specific to the particular case treated in the system. There is, of course, a great deal more to say about what it means for one question to be dependent on another and how the system knows whether it should respond to dependent questions or raise them with the user.

3.6.7 Opening up implicit grounding issues

In Chapter 2 we outlined a general issue-based account of grounding, where issues of contact, perception, understanding and acceptance of utterances may be raised and addressed. Parts of

this account were implemented in GODIS2, allowing the system e.g. to raise understanding questions regarding the user's input (e.g. "To Paris, is that correct?"). This is a case of explicitly raising the understanding-question which results in this question being under discussion.

The system could also produce positive explicit feedback (e.g. "To Paris"); this kind of feedback does not explicitly raise the understanding question, and there is no obligation on the user to respond to it before the dialogue can proceed. However, it can be argued that even positive feedback raises grounding-related issues, although not explicitly. This is given some support from the fact that it is possible for the user to protest against the system's feedback in case the system got something wrong.

According to Ginzburg, an assertion can be followed by any utterance addressing the acceptance of this question as a fact, e.g. by saying "no!". This is then regarded as a short answer to the acceptance question; in effect, a rejection. In the case of an assertion addressing understanding (i.e. positive understanding feedback), the acceptance question can be paraphrased "Is it correct that you meant 'to Paris'?". That is, the acceptance-question regarding the system's understanding is exactly the same question which is raised explicitly by an interrogative feedback utterance.

In GODIS, we have chosen not to represent acceptance-questions explicitly; however, in the case of positive explicit grounding there are good reasons to do so. Positive feedback has the advantage of increased efficiency compared to interrogative feedback, but the disadvantage is that the user is not able to correct the system's interpretation. However, if the positive feedback move implicitly raises the question whether the system's interpretation was correct, we can use this to allow the user to reject faulty system interpretations. Besides, we already have mechanisms in place for representing and dealing with answers to the understanding-question.

To model the fact that the acceptance question regarding understanding is implicit rather than explicit, we push it onto the local QUD only. If the user addresses it (e.g. by saying "no"), the implicit issue is "opened up", i.e. it becomes an open issue; it is pushed on ISSUES.

(RULE 3.9) RULE: **accommodateQUD2Issues**
 CLASS: **accommodate**
 PRE: $\left\{ \begin{array}{l} \$/PRIVATE/NIM/ELEM/SND=answer(A) \\ in(\$/SHARED/QUD, Q) \\ \$DOMAIN :: relevant(A, Q) \\ not\ in(\$/SHARED/ISSUES, Q) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} push(/SHARED/ISSUES, Q) \end{array} \right.$

The rule in (RULE 3.9) picks out a non-integrated **answer**-move which is **relevant** to a question on QUD which is not currently an open issue, and pushes it on ISSUES.

To handle integration responses to positive understanding feedback, we also need to modify the **integrateNegIcmAnswer** rule described in Section 2.6.6. A significant difference between positive and interrogative feedback in GODIS is that the former is associated with cautiously optimistic grounding, while the latter is used in the pessimistic grounding strategy. This means that a negative response to feedback on the understanding level must be handled differently depending on whether the content in question has been added to the dialogue gameboard or not. Specifically, if the positive feedback is rejected the optimistic grounding assumption must be retracted.

(RULE 3.10) RULE: **integrateNegIcmAnswer**
 CLASS: **integrate**

PRE:	$\left\{ \begin{array}{l} \$/PRIVATE/NIM/FST/SND=answer(A) \\ fst(\$/SHARED/ISSUES, Q) \\ \$DOMAIN :: resolves(A, Q) \\ fst(\$/SHARED/QUD, Q) \\ \$DOMAIN :: combine(Q, A, P) \\ P=not(und(DP*C)) \end{array} \right.$
EFF:	$\left\{ \begin{array}{l} pop(/PRIVATE/NIM) \\ pop(/SHARED/ISSUES) \\ if_do(in(\$/SHARED/COM, C) \text{ or } \\ \quad C=issue(Q') \text{ and } in(\$/SHARED/ISSUES, Q'), [\\ \quad \quad /SHARED/QUD := \$/PRIVATE/TMP/DP/QUD \\ \quad \quad /SHARED/ISSUES := \$/PRIVATE/TMP/DP/ISSUES \\ \quad \quad /SHARED/COM := \$/PRIVATE/TMP/DP/COM \\ \quad \quad /PRIVATE/AGENDA := \$/PRIVATE/TMP/DP/AGENDA \\ \quad \quad /PRIVATE/PLAN := \$/PRIVATE/TMP/DP/PLAN]) \\ \quad push(/PRIVATE/AGENDA, icm:und*pos:DP*not(C)) \\ \quad clear(/PRIVATE/NIM) \\ \quad init_shift(/PRIVATE/NIM) \end{array} \right.$

The rule in (RULE 3.10) is similar to those for integrating “normal” user answers (see Section 2.6.6), because of the special nature of grounding issues, we include issue downdating in the rule rather than adding a further rule for downdating ISSUES for this special case. This means the rule has to check that the answer **resolves** the grounding issue, rather than merely checking that it is **relevant**; this is done in the third condition. The content resulting from combining the issue on QUD and the answer is computed in the fifth condition. Finally, the sixth condition checks that the content is **not(und(DP*C))** where *DP* is a DP and *C* is the content that is being grounded (or in this case, not grounded).

The second update removes the grounding question from ISSUES. The third update first checks if *C* has been optimistically grounded. In this case, the optimistic grounding assumption regarding the grounding of *C* is retracted. This is where the new TMP/USR field, containing relevant parts

of the information state as they were before the latest user utterance was optimistically assumed to be grounded, is used. If C has not been optimistically assumed to be grounded, nothing in particular needs to be done.

The fourth update adds positive feedback that the system has understood that C was false. Note that **not**(C) is not added to /SHARED/COM. The reason for this is that the negated proposition is not something that the user intended to add to the DGB - it was simply a result of a misunderstanding by the system.

Note also that this feedback will not raise a grounding issue according to the definition of question-raising ICM in Section 2.7.1. Since the content **not**(C) has not been added to the information state, there is no point in dealing with grounding.

The final two updates clear the NIM queue, which means that the system will disregard any moves which have not yet been integrated. One motivation for this is that if the system has misheard some part of the user's utterance, it is likely that it also misheard the rest. Clearing the NIM stack is also useful in dialogues such as that in (DIALOGUE 3.9).

(DIALOGUE 3.9)

```
S> Welcome to the travel agency!
U> flight to paris
$$> flight. to paris. I dont quite understand. You want to
know about price, is that correct?
U> no
S> You did not ask about price.
```

When integrating the user's "no" (a negative response to system ICM), the **answer**-moves realized in the utterance "flight to paris" are discarded;. The system has tried to make sense of it but the user rejected this attempt. At this point, the system simply cannot deal with them and rather than getting stuck in trying to figure out what the user meant, the moves are thrown out.

Note that the rule as implemented is actually more general than what is needed for (or used in) GoDIS3. Since the part of TMP that it backtracks to depends on the DP variable, in principle it could be used for cases where the user gives positive feedback and the system rejects this as mistaken.

A sample dialogue with a negative response to an implicit grounding question is shown in (DIALOGUE 3.10).

(DIALOGUE 3.10)

S> Welcome to the travel agency!

U> visa information please (0.78) (user actually said something else)

getLatestMoves
 backupSharedUsr
 integrateUsrAsk

findPlan
 downdateQUD
 backupSharedSys
 selectIcmOther
 selectIcmOther

S> Okay. You want to know about price.

getLatestMoves
 integrateOtherICM
 integrateUndPosICM

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \langle \langle \text{icm:loadplan} \rangle \rangle \\ \text{findout}(\text{?A.how}(A)) \\ \text{findout}(\text{?B.dest_city}(B)) \\ \text{findout}(\text{?C.dept_city}(C)) \\ \text{PLAN} = \langle \begin{array}{l} \text{findout}(\text{?D.month}(D)) \\ \text{findout}(\text{?E.dept_day}(E)) \\ \text{findout}(\text{?F.class}(F)) \\ \text{consultDB}(\text{?G.price}(G)) \end{array} \rangle \\ \text{BEL} = \{ \} \\ \text{TMP} = \left[\begin{array}{l} \text{USR} = \left[\begin{array}{l} \text{COM} = \{ \} \\ \text{QUD} = \langle \rangle \\ \text{ISSUES} = \langle \rangle \\ \text{AGENDA} = \langle \langle \rangle \rangle \\ \text{PLAN} = \langle \rangle \end{array} \right] \\ \text{NIM} = \langle \langle \rangle \rangle \\ \text{COM} = \{ \} \\ \text{ISSUES} = \langle \text{?H.price}(H) \rangle \\ \text{QUD} = \langle \text{und}(\text{usr*issue}(\text{?I.price}(I))) \rangle \\ \text{PM} = \{ \text{ask}(\text{?H.price}(H)) \} \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{sys} \\ \text{MOVES} = \left\{ \begin{array}{l} \text{icm:und*pos:usr*issue}(\text{?I.price}(I)) \\ \text{icm:acc*pos} \end{array} \right\} \end{array} \right] \end{array} \right] \right]$$

U> no

```

getLatestMoves
accommodateQUD2Issues
{ push(/SHARED/ISSUES, und(usr*issue(?A.price(A))))
integrateNegIcmAnswer
{
  pop(/PRIVATE/NIM)
  pop(/SHARED/ISSUES)
  if_do(in(/SHARED/COM, issue(?A.price(A))) or
    issue(?A.price(A))=issue(D) and in(/SHARED/ISSUES, D), [
    /SHARED/QUD := /PRIVATE/TMP/USR/QUD
    /SHARED/ISSUES := /PRIVATE/TMP/USR/ISSUES
    /SHARED/COM := /PRIVATE/TMP/USR/COM
    /PRIVATE/AGENDA := /PRIVATE/TMP/USR/AGENDA
    /PRIVATE/PLAN := /PRIVATE/TMP/USR/PLAN ])
  push(/PRIVATE/AGENDA, icm:und*pos:usr*not(issue(?A.price(A))))
  clear(/PRIVATE/NIM)
  init_shift(/PRIVATE/NIM)
}
downdateQUD

```

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \begin{array}{l} \text{AGENDA} = \langle\langle \text{icm:loadplan} \\ \text{icm:und*pos:usr*not(issue(?A.price(A)))} \rangle\rangle \\ \text{PLAN} = \langle \rangle \\ \text{BEL} = \{ \} \\ \text{TMP} = \left[\begin{array}{l} \text{USR} = \left[\begin{array}{l} \text{COM} = \{ \} \\ \text{QUD} = \langle \rangle \\ \text{ISSUES} = \langle \rangle \\ \text{AGENDA} = \langle\langle \rangle \rangle \\ \text{PLAN} = \langle \rangle \end{array} \right] \end{array} \right] \\ \text{NIM} = \langle\langle \rangle \rangle \\ \text{COM} = \{ \} \\ \text{ISSUES} = \langle \rangle \\ \text{QUD} = \langle \rangle \\ \text{PM} = \{ \text{icm:und*pos:usr*issue(?A.price(A)), icm:acc*pos} \} \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{usr} \\ \text{MOVES} = \{ \text{answer(not(und(usr*issue(?A.price(A))))} \} \end{array} \right] \end{array} \right] \right]$$

```

backupSharedSys
selectIcmOther

```

S> You did not ask about price.

Feedback for utterances realizing several moves

In the case where a user utterance provides answers to several questions (and perhaps also asks a question), the strategy we have chosen is to produce feedback for each move individually to give the user a chance to correct mishearings or misunderstandings. If positive feedback regard-

ing one piece of information is rejected, the optimistic assumption is retracted by modifying the information state according to the /PRIVATE/TMP/USR saved state. If feedback regarding one piece of information is accepted, this information should not disappear if a later feedback utterance from the system should be rejected.

(DIALOGUE 3.11)

```
S> Welcome to the travel agency!
U> price information please
S> Okay. I need some information. How do you want to travel?
U> a flight to ferris [0.78]
S> Okay. by flight.
U> yes
S> to paris.
U> no
S> not to paris. So, what city do you want to go to?
```

For example, in the dialogue in (DIALOGUE 3.11), the user accepts the system's feedback "by flight", but rejects "to paris"; however, the information that the user wants to travel by flight is retained.

To handle this, each time positive feedback is accepted, the parts of the /PRIVATE/TMP/USR structure corresponding to the SHARED field are modified. A further modification is thus needed for the **integratePosIcmAnswer** rule previously defined in Section 2.6.6.

(RULE 3.11) RULE: **integratePosIcmAnswer**
 CLASS: **integrate**

PRE:	{	$\$/PRIVATE/NIM/FST/SND=answer(A)$ $fst(\$/SHARED/ISSUES, Q)$ $\$DOMAIN :: resolves(A, Q)$ $fst(\$/SHARED/QUD, Q)$ $\$DOMAIN :: combine(Q, A, P)$ $P=und(DP*Content)$
EFF:	{	$pop(/PRIVATE/NIM)$ $pop(/SHARED/ISSUES)$ $if_then_else(Content=issue(Q'), [$ $\quad push(/PRIVATE/TMP/DP/QUD, Q')$ $\quad push(/PRIVATE/TMP/DP/ISSUES, Q')$ $\quad push(/PRIVATE/TMP/DP/AGENDA, respond(Q'))],$ $\quad add(/PRIVATE/TMP/DP/COM, Content))$ $if_do(not (in(\$/SHARED/COM, Content) or$ $\quad Content=issue(Q') and in(\$/SHARED/ISSUES, Q')),$ $\quad if_then_else(Content=issue(Q'), [$ $\quad \quad push(/SHARED/QUD, Q')$ $\quad \quad push(/SHARED/ISSUES, Q')$ $\quad \quad push(/PRIVATE/AGENDA, respond(Q'))],$ $\quad \quad add(/SHARED/COM, Content)))$

The conditions are similar to those of the previous version of the rule, except for inspecting ISSUES instead of QUD. The first two updates are also the same. The third update adds the content *Content* which is being grounded to TMP/USR (in case *DP* is *usr*, which it always is in GODIS3). This means that if future feedback (concerning the same utterance) from the system is rejected, the system will backtrack to a state where *Content* is integrated. The conditionals in the third and fourth updates reflect the fact that questions are integrated differently from propositions. The fourth update is similar to the third update in the previous version of the rule.

Implicit acceptance

Before we move on there is one more thing to consider. If the user does not reject the system's positive feedback concerning a piece of information, this is regarded as an implicit acceptance. Therefore, we also need to add a **noFollowup** rule, for cases where positive system feedback is not responded to at all (i.e. the user does not take the turn offered).

(RULE 3.12) RULE: **noFollowup**
 CLASS: (none)
 PRE: { \$INPUT, 'TIMED_OUT'
 in(\$/SHARED/PM, icm:und*pos:usr**Content*)
 if_then_else(*Content*=issue(*Q*), [
 push(/PRIVATE/TMP/USR/QUD, *Q*)
 push(/PRIVATE/TMP/USR/ISSUES, *Q*)
 push(/PRIVATE/TMP/USR/AGENDA, respond(*Q*))],
 EFF: {
 add(/PRIVATE/TMP/USR/COM, *Content*))

The first condition is true only if the user did not produce any utterance (that the system heard) during her latest turn¹². The second condition checks that the moves performed in the previous utterance includes positive understanding feedback regarding *Content*. The first updates are identical to the third update in the **integratePosIcmAnswer** rule in Section 3.6.7.

Below is a dialogue example involving positive, implicit positive, and negative followups to system feedback.

(DIALOGUE 3.12)

```
S> Welcome to the travel agency!
U> price information please
S> Okay. Lets see. How do you want to travel?
U> a flight to paris in april
S> Okay. by flight.
U> yes
S> to paris.
U>
S> in april.
U> no
S> not in april. What city do you want to go to?
```

Implicit questions and elliptical answers

In the case of implicit acceptance questions in English (and Swedish) it appears that they can be addressed by short answers; however, we cannot assume that all implicit issues can be addressed elliptically. The use of QUD for storing implicit issues relies on the fact that questions on QUD have not necessarily been raised explicitly; however, questions on QUD are also by definition

¹²See Section 2.6.6 for an explanation of 'TIMED_OUT'.

available for resolution of short answers. To represent implicit questions which cannot be addressed elliptically, a further local data structure for implicit questions under discussion would be needed.

3.7 Further implementation issues

In this section we describe parts of the implementation of GODIS3 which have not been discussed earlier in this chapter, and which are not directly reused from GODIS2.

3.7.1 Dialogue moves

For GODIS3, only one dialogue move has been added: ICM indicating accommodation of a dependent issue. In English, we have chosen “alright, you want to know about . . .” to indicate that some inference has been performed, and that it has been successful. This choice is based on the intuition that this indicates some process inference which has concluded successfully; this should be regarded as a preliminary and temporary solution awaiting further corpus and usability studies.

- `icm:accommodate:Q` : Move if `Q` : Question

3.7.2 GODIS3 update module

Update rules

The main additions to the update rule collection needed to handle accommodation and reaccommodation were described above in Section 3.6.

In this section, we describe changes applied to other rules from GODIS2 to fit with the modified information state used by GODIS3.

Backing up TMP/USR

The TMP/USR field contains copies of parts of the information state as they were before the latest user utterance was integrated. If the optimistic assumption should turn out to be wrong, the TMP/USR field is used to undo the optimistic grounding assumption without the need for complex revision processing (see Section 3.6.7).

The **backupSharedUsr** in (8) is called each time an utterance is to be integrated and stores the current QUD, ISSUES, COM, PLAN and AGENDA fields; these are all potentially affected by the integration of the moves in the latest utterance, and are also important for determining what to do next.

This backtracking mechanism only applies to domain-level communication; user ICM moves are always optimistically assumed to be correctly understood and integration always succeeds. Since ICM “subdialogues”, such as that in (DIALOGUE 3.13) are used to establish the fact that a previous user utterance was misunderstood by the system, it is important that TMP/USR is not overwritten during the subdialogue. For example, the **backupSharedUsr** rule should not trigger before integrating the user’s “pardon” or the user’s answer “no” to the system ICM “by boat”.

(DIALOGUE 3.13)

S> Okay. Lets see. How do you want to travel?
U> by boat [0.76] (user actually said something else)
S> Okay. by boat.
U> pardon ?
S> Okay. by boat.
U> no
S> not by boat. So, how do you want to travel?

(RULE 3.13) RULE: **backupSharedUsr**
 CLASS: (none)
 PRE: {
 \$LATEST_SPEAKER=*usr*
 \$LATEST_MOVES=*Moves*
 not in(*Moves*, icm:*X*)
 not in(*Moves*, no_move)
 not (fst(\$/SHARED/QUD, und(*usr***C*)) and
 in(*A*, answer(*D*)) and
 \$DOMAIN :: relevant(*D*, und(*usr***C*)))
 EFF: {
 /Private/TMP/USR/QUD := \$/SHARED/QUD
 /Private/TMP/USR/ISSUES := \$/SHARED/ISSUES
 /Private/TMP/USR/COM := \$/SHARED/COM
 /Private/TMP/USR/AGENDA := \$/PRIVATE/AGENDA
 /Private/TMP/USR/PLAN := \$/PRIVATE/PLAN

The first condition checks that the latest speaker was indeed the user; if not, the rule should of course not trigger. The next four conditions are used to prevent triggering in case of an ICM subdialog, i.e. if the user produced an ICM move or responded to one from the system. (Note that *no_move* may count as implicit ICM if the user does not respond to ICM from the system; see Section 3.6.7). The fifth condition checks if the user utterance contains an answer relevant to a grounding-question on QUD. The effects simply copy the contents of TMP/USR to the corresponding paths in the information state.

Integration rules and NIM

In GODIS2, the integration rules inspect NIM using the condition in(/PRIVATE/NIM, *Moves*). Since TRINDIKIT uses backtracking to find instantiations of variables in conditions (see SIRIDUS (2002)), this results in each integration rule looking through the whole queue of non-integrated moves. Thus, in GODIS2 the ordering of the integration rules determines which move is integrated first. This is okay for dialogues with a very simple structure, but when dialogues become more complex (e.g. because of accommodation), the ordering of the moves becomes more important.

Therefore, in GODIS3 all integration rules inspect only the first move on the NIM queue, using the condition fst(/PRIVATE/NIM, *Move*) or similar. In combination with the queue-shifting technique described in Section 3.7.2, this means that the algorithm tries to integrate moves in the order they were performed.

Update algorithm

Because of the more complex dialogues handled by GODIS3, the update algorithm is a bit more complex than that for GODIS2.

```
(8) 1 if not ($LATEST_MOVES == failed)
    2 then { getLatestMoves,
    3       try backupSharedUsr,
    4       try irrelevantFollowup,
    5       repeat {
    6           repeat( { integrate,
    7                     try downdate_issues,
    8                     try removeFindout,
    9                     try load_plan },
    10                  or else apply shift( /PRIVATE/NIM ) )
    11           until fully_shifted($/PRIVATE/NIM),
    12           apply shift(/PRIVATE/NIM),
    13           try select_action
    14           accommodate },
    15       apply cancel_shift( /PRIVATE/NIM ),
    16       repeat exec_plan,
    17       try downdate_qud }
    18 else { failedFollowup or else unclearFollowup }
```

Line 1 checks that the interpretation of the latest utterance was successful (of course, in the case of system utterances this is always true). If not, the **failedFollowup** and **unclearFollowup** rules in line 18, described in Section 2.6.8, are tried. If interpretation was successful, the latest moves are incorporated in the information state proper by the **getLatestMoves** rule (see Section 2.6.7). After this the **backupSharedUsr** rule is tried; its conditions are satisfied, the rule will trigger and store a copy of relevant parts of the information state in case the system makes an optimistic grounding assumption which turns out to be mistaken (see Section 3.7.2). Also, before integration starts, the **irrelevantFollowup** rule described in Section 2.6.8 is tried to catch cases where a system question has been ignored by the user.

After this, a loop involving integration and accommodation is executed until nothing more can be integrated (i.e. until the loop can no longer be executed). The basic idea is this: first try to integrate as many moves as possible by cycling through the NIM queue; then, if accommodation can be applied, do the same thing again. Repeat this until nothing can be integrated and no accommodation is possible.

The first part of this loop starts in line 6 and is itself a loop for cycling through all non-integrated moves and trying to integrate them. If integration succeeds, the algorithm tries to remove any resolved issues from ISSUES and PLAN, and if necessary load a new plan (e.g. if an `ask` move from the user was integrated). Then it tries integration again. If integration fails, the NIM queue is shifted one step, i.e. the topmost element is removed from the top and pushed to the end of the queue. Then, integration is tried again. This continues until the queue has been completely cycled through once, and all moves have had a shot at being integrated.

After this loop is finished, accommodation will attempt to adjust the `/SHARED` field so that any moves still not integrated may be understood on the pragmatic level, and integrated. However, we need to avoid a problem that arises as a consequence of having the integration rules handle pragmatic understanding, acceptance, and integration in a single step. The problem arises if some move is regarded as relevant (i.e. understood on the pragmatic level) but not acceptable, or if a relevant move has low reliability and should be verified before being integrated. In this case, accommodation should not be tried since the purpose of accommodation is to understand some utterance on the pragmatic level, and this has already been achieved. To solve this problem, some action selection rules (of class `select_action`) have been moved from the selection module to the update module. Before trying accommodation, line 13 of the update algorithm thus tries to select rejection moves and interrogative feedback moves to catch any moves which have already been understood.

Line 14 calls the accommodation rule class. If this succeeds, there is a chance that some moves that could not be integrated before can now be integrated, so the loop starting in line 6 is restarted. When nothing can be integrated and nothing can be accommodated, the sequence starting at line 6 and ending at line 14 cannot be executed, and consequently the loop started in line 5 will be finished. Line 15 cancels shifting of the NIM queue (see SIRIDUS (2002)).

Any loaded plan is executed in line 16 by repeatedly applying the `exec_plan` rule class until no more execution is possible at the current stage of the dialogue. Finally, QUD is downdated.

As an example of how integration and accommodation interact, in the dialogue in (DIALOGUE 3.14), “to paris” is integrated before accommodation is tried, so the only question available for ellipsis resolution of “paris” is the one concerning departure city.

(DIALOGUE 3.14)

```
S> Welcome to the travel agency!
U> price london to paris [0.78]
S> Okay. You want to know about price.
S> I need some information. to paris.
S> from london.
S> How do you want to travel?
```

Accommodation rule ordering For the **accommodate** rule class, the ordering in which the various accommodation rules are tried may be important in some cases. The ordering used in GoDIS3 is shown in (9).

- (9) **accommodate**
 1. **accommodateIssues2QUD**
 2. **accommodateQUD2Issues**
 3. **accommodatePlan2Issues**
 4. **accommodateCom2Issues**
 5. **accommodateCom2IssuesDependent**
 6. **accommodateDependentIssue**

This order in which to try the accommodation rules has been chosen based on intuitions about how accessible questions are depending on where they are retrieved. By experimenting with the ordering, different behaviours can be obtained. The current ordering should be regarded as provisional, and finding the “best” ordering is an object for future research. It may also sometimes be necessary to do clarification if an answer matches several questions whose accommodation rules have the same or nearly the same priority; this has not been implemented in GoDIS3.

Possible criteria for judging whether one ordering is better than another are (1) how reasonable the resulting behaviours are, (2) how efficient the overall processing becomes, and (3) how similar to human cognitive processes corresponding to accommodation the processing is (assuming question accommodation is cognitively plausible).

First, accommodation involving only ISSUES and QUD is tried, since these are the central structures for dealing with questions. If this fails, accommodation from the dialogue plan is tried; if this fails, reaccommodation from COM is attempted. First “normal” reaccommodation, then dependent reaccommodation. Finally, dependent issue accommodation is tried; this is tried last since it finds the question in the domain resource rather than the information state proper.

3.7.3 Selection module

The selection module is almost unchanged from GoDIS2. Some minor adjustments have been made to adapt the rules to the changes in the information state type: that objects in NIM are pairs of DPs and moves, and that TMP is divided into two substructures.

3.8 Discussion

In this section we discuss some variations on GODIS3, show some additional “emergent” features, and discuss various aspects of question accommodation.

3.8.1 Phrase spotting and syntax in flexible dialogue

As it turns out, GODIS3 sometimes runs into trouble if the interpreter recognizes several answers to the same question in an utterance. Whereas GODIS2 would simply integrate the first answer and ignore the second, GODIS3 will try to make sense of all the moves in an utterance, which may lead to problems if the accommodation rules are not designed to cover the case at hand.

For example, if the system recognizes “paris to london” as a first utterance in a dialogue, the system will try dependent issue accommodation (see Section 3.6.4) and note that the set of answers (`answer(paris)` and `answer(dest_city(london))`) is (indirectly) relevant to both the price issue and the visa issue. It might seem that this is wrong, since the two answers are in fact **relevant** to the same question (regarding destination city) in the “visa” plan, whereas it is **relevant** to two separate questions (destination and departure city) in the “price” plan, so it should be indirectly relevant only to the “price” issue. But in general, one cannot require that the two answers must be answers to different questions, since the second answer may be a correction of the first. This may of course be signalled more clearly, as in “to paris uh no to london”, but the correction signals may be left out, inaudible, or not recognized.

One way to solve this problem is to sometimes look for constructions which realize more than one move, and do some “cleaning up” in the interpretation phase so that the DME will not get into trouble. For example, we can add a lexical entry looking for phrases of the form “*X* to *Y*” and interpret this as “from *X* to *Y*”, i.e. `answer(dept_city(X))` and `answer(dest_city(Y))`.

A related problem occurs if the user first chooses Gothenburg as departure city and then says “not from gothenburg london”. Since plan-to-issues accommodation has precedence over com-to-issues, “london” will be integrated first by accommodating the destination city question, which is wrong. One solution is of course to give com-to-issues accommodation precedence, but then for “paris from london”, “from london” will first be integrated and then “paris” will be seen as a revision of the departure city, which is also wrong.

As mentioned before in Section 3.7.2, the exact precedence ordering between accommodation rules is a topic for future research, and it may sometimes be necessary to do clarification if an answer matches several questions whose accommodation rules have the same or nearly the same priority. However, an easier solution is to add a further interpretation rule saying that “not *P X*, *P Y*” should be interpreted as a paraphrase of “not *P X*, *P Y*”.

A slightly irritating but not very serious “bug” in GODIS occurs if a user utterance contains two answers to the same question (e.g. “to kuala lumpur to london”), and the first of these is an invalid database parameter. The first answer will be rejected, and appropriate feedback will be put on the agenda. The second answer will then (correctly) replace the first answer using retraction, but the rejection feedback concerning the now replaced first answer remains on the agenda. This means that the system will give some irrelevant information, namely that the first answer was rejected. This can be fixed to some extent by interpreting phrases of the form “ $PXPY$ ” as “ PY ”, i.e. the second part is regarded as a correction of the first part. Similarly, phrases of the form “ $PX \text{ no } (P)Y$ ”, where “no” is regarded as a correction indicator and the second P is optional, can also be interpreted as “ PY ”. In general, it is useful to detect corrections in the interpretation phase to avoid potentially expensive revisions in the integration phase.

Of course, these simple fixes will only get us so far, since they only capture the very simplest cases. For example, we would not be able to notice that an utterance contains two answers to the same question unless they are adjacent. What is really needed here is a proper parser (e.g. a HPSG-based parser) and grammar.

What this shows us, then, is that taking syntax into account in interpretation becomes increasingly important when the dialogue management becomes more flexible. This might not be very surprising, but by starting out with very simple keyword spotting and keeping it as long as possible, we find out exactly where it becomes insufficient. Had we used a sophisticated parser from the outset, we would not have had this insight.

3.8.2 Relaxing constraints using denial and dependent reaccommodation

In inquiry-oriented dialogue it is often useful to be able to relax constraints on the database search, especially in cases where no result is returned from the database. As it turns out, existing accommodation mechanisms in GODIS3 in combination with the slightly more complex database result format which was introduced as an extension of GODIS (see Larsson *et al.* (2002)) provides a way of doing exactly this.

The idea is this: if the system returns an empty or otherwise unsatisfactory database result, the user can modify the information given to the system by denying some part of it. This triggers reaccommodation of the dependent issue that the database search tried to find an answer to, and the corresponding plan is reloaded. Since some information is now missing, the system will again ask for it; the rest of the actions in the plan will be removed since the questions have already been answered. By rejecting the repeated question (e.g. by saying “It doesn’t matter” or “I don’t know”), the user makes the system search the database again, but this time with relaxed constraints.

(DIALOGUE 3.15)

S> Welcome to the travel agency!
U> a flight to paris from gothenburg in april
S> Alright. You want to know about price.
S> Lets see. Okay. What day do you want to leave?
U> the fifth
S> Okay. What class did you have in mind?
U> business class
\$\$> Okay. Sorry, there is nothing matching your request about price.
U> okay, not business class then
S> You want to know about price.
\$\$> Lets see. Okay. So, What class did you have in mind?
U> it doesnt matter
S> The price is 7654 crowns. economy class.

The example in (DIALOGUE 3.15) illustrates the principle. To make this feature more useful, one would need to be able to give disjunctive answers (“I want to leave on the twenty-first or the twenty-second”), specify a range of allowable “parameter values” (“I want to leave between the twenty-first and the twenty-fifth”), and give answers which are unspecific in relation to the question (“I want to go to France” in response to “What city do you want to go to”). Of course, this is a useful feature also when providing answers in the initial database query specification stage. This is an area for further implementation.

The nice thing about this approach to relaxing constraints is that it requires no special machinery, but it emerges as a possibility given mechanisms which are independently motivated.

3.8.3 “Smart” interpretation

The dialogues in (DIALOGUE 3.16) and (DIALOGUE 3.17) show how the integration strategy in GODIS2 is able to use all the information provided in an utterance as clues for how to interpret underspecified information. The interpretation of “paris” as destination or departure city depends on what other information is given in the utterance, specifically whether destination or departure city is specified elsewhere in the utterance.

(DIALOGUE 3.16)

U> price information please
S> Okay. Lets see. How do you want to travel?

U> paris from london [0.78]
S> Okay. from london.
S> to paris.

(DIALOGUE 3.17)

U> price information please
S> Okay. Lets see. How do you want to travel?
U> paris to london [0.78]
S> Okay. to london.
S> from paris.

The sample dialogue in (DIALOGUE 3.18) shows how the system can integrate successive hints from the user about what she wants to do.

(DIALOGUE 3.18)

S> Welcome to the travel agency!
U> um i would like to go to paris
S> to paris. I dont quite understand. Do you mean to ask about visa or to ask about price?
U> um i want to go by flight
S> Alright. You want to know about price.

3.8.4 Separating understanding, acceptance, and integration

The integration rules in GODIS3 perform several tasks: contextual interpretation (e.g. for underspecified answers), deciding whether to accept or reject a move and their contents, and (if acceptance is decided on) integration of the full effects of the move. While this was a good approach in GODIS1 and GODIS2, in GODIS3 this approach sometimes obscures the workings of the system and make rules rather complex.

An alternative approach would be to implement contextual interpretation, the acceptance/rejection decision, and integration as separate rule classes. The contextual interpretation rules would take moves off a queue of moves provided by the interpretation module (corresponding to the current NIM field); we could call this queue of possibly underspecified moves SUM (Semantically Understood Moves). The resulting fully specified moves could then be added to a PUM (Pragmatically Understood Moves) queue, which would serve as input for the acceptance/rejection decision rules. Rejected moves would be put on a RM (Rejected Moves) queue, which would

later be inspected in the selection phase to produce suitable feedback. Accepted moves would be added to an AM (Accepted Moves) stack, which in turn would serve as input to the integration rules. While this would probably require a larger number of rules and also some additional data structures in the information state, the complexity of the individual rules could be greatly reduced and the clarity of the overall processing would improve. It is also likely that this would lead to a less bug-prone and theoretically more satisfying implementation.

3.8.5 Accommodation and the speaker's own utterances

In this chapter we have been mainly concerned with issue accommodation as a way of interpreting utterances from the other DP (for a dialogue system, the user). But how does accommodation relate to the generation and integration of one's own utterances? This issues does not come up in GODIS since the system never produces utterances that can be expected to require accommodation on the part of the user (e.g. ending a long dialogue with "\$100" rather than "The price is \$100").

Ginzburg allows the speaker to update QUD with a question and then address it. This will (probably) require accommodation on the part of the hearer. The sequence of events here is roughly the following (*S* is the speaker, *H* the hearer):

- *S* pushes *Q* on QUD, then addresses *Q*
- *S* integrates *A*
- *H* accommodates *Q*, integrates *A*

However, we have noted above in Section 2.3.4 that this seems inconsistent with the view of QUD as something that is assumed to be shared. Possibly, one could have a "fuzzier" concept of QUD (and perhaps the DGB in general) that leaves some freedom of modifying it privately, as long as the hearer can be expected to accommodate these modifications.

The other alternative is to allow the speaker to generate utterances that do not exactly match the current information state, and then perform accommodation to integrate her own utterance. In this case, the sequence of events is instead:

- *S* addresses *Q*, believing that the information state can be adjusted (using accommodation) so as to make this utterance felicitous
- *S* and *H* accommodate *Q* and integrate *A*

Whether the choice between these two approaches make any real difference to the internal processing and/or external behaviour of the system remains a future research issue. For example, if QUD is updated with Q before A is produced, and the utterance realizing A is interrupted, should Q be removed from QUD?

3.8.6 Accommodation vs. normal integration

As we have seen, question accommodation allows a generalized account for how answers are integrated into the information state, regardless of the status of the corresponding question. The accommodation procedure may also have side-effects (e.g. loading a new dialogue plan) which serve to drive the dialogue forward.

Instead of giving rules for accommodation and integration separately, one could deny the existence of accommodation and just give more complex integration rules. The integration rule for short answers requires that there is a question on the QUD to which the latest move is an appropriate answer, and the accommodation rules are used if no such question can be found. The alternative is to skip the QUD requirement, thus incorporating the accommodation mechanisms into the integration rule, which would then split into several rules. For example, there would be one rule for integrating answers by matching them to questions in the plan directly.

Apart from the theoretical argument that question accommodation provides a generalization of the way answers are integrated, there are also practical motivations. In particular, the fact that several steps of accommodation may be necessary to integrate a single answer means that the total number of rules for integrating answers would be higher if accommodation was not used - one would need at least one integration rule for each possible combination of accommodation rules.

A further argument which is not explored in this thesis (but see Engdahl *et al.*, 1999) is that question presupposition and accommodation interact with intra-sentential information structure in interesting and useful ways.

3.8.7 Dependent issue accommodation in VoiceXML?

On a close reading of the VoiceXML specification (McGlashan *et al.*, 2001), it may appear that VoiceXML offers a mechanism similar to dependent issue accommodation¹³. In VoiceXML, a grammar can have scope over a single slot, over a form, or over a whole document (containing

¹³This discussion is based on the VoiceXML specification rather than hands-on experience of VoiceXML. This means that some unclarity remains about the capabilities of VoiceXML in general, and individual implementations of VoiceXML servers in particular. For both these reasons, the discussion should be regarded as tentative and open

several forms). Given a grammar with document scope (defining a set of sentences which the VoiceXML interpreter will listen for during the whole dialogue), if the user gives information which does not match the currently active form, VoiceXML will jump to a form matching the input¹⁴. This corresponds roughly to the dependent issue accommodation mechanism in GODIS. However, if input matches more than one task (e.g. “raise the volume” could match a task related to the TV or one related to the CD player), VoiceXML will not ask which of these tasks the user wants to perform but instead go to the one it finds first, regardless of what the user intended. Generally, it is hard to see how clarification questions could be handled in a general way in VoiceXML, since they do not belong to a particular form.

3.9 Summary

To enable more flexible dialogue behaviour, we made a distinction between a local and a global QUD (referring to the latter as “open issues”, or just “issues”). The notions of question and issue accommodation were then introduced to allow the system to be more flexible in the way utterances are interpreted relative to the dialogue context. Question accommodation allows the system to understand answers addressing issues which have not yet been raised. In cases of ambiguity, where an answer matches several possible questions, clarification dialogues may be needed.

for revision. However, it should also be pointed out that it is fairly clear what is supported in VoiceXML; most of the unclarities refer to what is possible, but not explicitly supported, in VoiceXML. In general, it is more important to know what is supported by a standard than what is possible, since almost anything is possible in any programming environment (given a sufficient number of hacks).

¹⁴Although the VoiceXML documentation does not provide any examples of this kind of behaviour, it appears to be possible, at least in principle.

Chapter 4

Action-oriented and negotiative dialogue

4.1 Introduction

In this chapter, we extend the issue-based approach to simple action-oriented and negotiative dialogue¹. First, we deal with action-oriented dialogue (AOD), which involves DPs performing non-communicative actions such as e.g. adding a program to a VCR or reserving tickets in a travel agency. We extend the GODIS system to handle a simple kind of AOD. In addition to issues and questions under discussion, this system also has to keep track of actions. Usually, it is useful for an AOD system to also handle IOD.

The concept of issue accommodation is extended to action accommodation. We also show how multiple simultaneous plans may be used to enable more complex dialogue structures, and how multiple plans interact with actions and issues. We show how dialogue plans may be constructed from menus, and illustrate menu-based AOD with examples from an implementation of a menu-based VCR interface.

Next, we turn to negotiative dialogue, and describe an issue-based account of a simple kind of collaborative negotiative dialogue. We also sketch a formalization of this account and discuss its implementation in GODIS.

¹This chapter is a slightly altered version of Chapter 5 in Larsson (2002a).

4.2 Issues and actions in action-oriented dialogue

In GODIS3, each dialogue plan was aimed at resolving a specific issue. In general, of course, not all dialogue is aimed at resolving issues; often it is aimed towards the performance of some (non-communicative) action. For example, turning on or off the lights in a room, adding a program to a VCR, calling somebody up, or making a reservation in a travel agency. Action oriented dialogue in general places obligations on DPs to perform actions, either during the dialogue or after. For example, booking a ticket involves an obligation on the clerk to send a ticket to the customer, and on the customer to pay for the ticket. Requesting a VCR manager to add a program puts an obligation on the manager to add the program to the VCR timer recording memory bank.

We will be dealing with a simple kind of AOD, where each action can only be performed by one of the DPs, similar to our assumptions regarding issues. This allows a simple representation of actions that does not take into account who has the obligation to perform each action. Since we are giving examples from a device control domain (VCR control), we will in fact only deal with the case where all actions are performed by the system².

Previous work with GoDiS, the predecessor of GODIS, has also addressed the case where the user performs all the actions (Larsson, 2000, Larsson and Zaenen, 2000).

4.3 Extending GODIS to handle action oriented dialogue

In this section, we describe additions to the information state, semantics, and dialogue moves. Update rules will be discussed in Section 4.6.

4.3.1 Enhancing the information state

In this section, we show how the GODIS information state needs to be modified to handle Action Oriented Dialogue. The new information state type is shown in Figure 4.1.

The only addition is the ACTIONS field which has been added to /SHARED and /PRIVATE/TMP. We assume the actions stack is an open stack, which is the same structure that we use for ISSUES.

²Of course, even in this simple domain it cannot really be assumed generally that the system performs all the actions; one could well imagine a VCR control dialogue system which, for example, requests the user to insert a tape into the VCR.

$$\begin{array}{l}
\left[\begin{array}{l}
\text{PRIVATE} : \left[\begin{array}{l}
\text{AGENDA} : \text{OpenQueue}(\text{Action}) \\
\text{PLAN} : \text{OpenStack}(\text{PlanConstruct}) \\
\text{BEL} : \text{Set}(\text{Prop}) \\
\text{TMP} : \left[\begin{array}{l} \text{USR} : \text{Tmp} \\ \text{SYS} : \text{Tmp} \end{array} \right] \\
\text{NIM} : \text{OpenQueue}(\text{Pair}(\text{DP}, \text{Move}))
\end{array} \right] \\
\text{SHARED} : \left[\begin{array}{l}
\text{COM} : \text{Set}(\text{Prop}) \\
\text{ISSUES} : \text{OpenStack}(\text{Question}) \\
\text{ACTIONS} : \text{OpenStack}(\text{Action}) \\
\text{QUD} : \text{OpenStack}(\text{Question}) \\
\text{PM} : \text{OpenQueue}(\text{Move}) \\
\text{LU} : \left[\begin{array}{l} \text{SPEAKER} : \text{Participant} \\ \text{MOVES} : \text{Set}(\text{Move}) \end{array} \right]
\end{array} \right]
\end{array} \right] \\
\text{Tmp} = \left[\begin{array}{l}
\text{COM} : \text{Set}(\text{Prop}) \\
\text{ISSUES} : \text{OpenStack}(\text{Question}) \\
\text{ACTIONS} : \text{OpenStack}(\text{Action}) \\
\text{QUD} : \text{OpenStack}(\text{Question}) \\
\text{AGENDA} : \text{OpenQueue}(\text{Action}) \\
\text{PLAN} : \text{OpenStack}(\text{PlanConstruct})
\end{array} \right]
\end{array}$$

Figure 4.1: GoDIS4 Information State type

Semantics

To handle action-oriented dialogue we need to extend our semantics. Given that $\alpha : \text{Action}$, we have

- **action**(α) : Proposition
- **done**(α) : Proposition

Rough paraphrases of these propositions are “action α should be performed (by any DP who can perform α)”, and “action α has been successfully performed”, respectively.

Actions and postconditions

The set of actions that can be requested depends on the domain; for example, in the travel booking domain one action would be `make_reservation`, and an example from the VCR control domain

is `vcr_add_program`. For dialogues where the user requests actions to be performed by the system, each such action (which we may refer to as a *goal-action*) is associated with a dialogue plan.

In device control dialogue, there is also an additional kind of actions, namely those that are specified by the device itself; we refer to these as *device actions*. We will also generalize over device actions using the UPnP protocol (“Universal Plug’n’Play”, Microsoft, 2000, Boye *et al.*, 2001, Lewin *et al.*, 2001); this requires a further type of *upnp action* whose arguments are a device and a device action. This allows us to access multiple devices defined using a common interface. This will be further clarified in Section 4.4.1.

Device actions and UPnP actions can be thought of as atomic actions, whereas goal actions are more complex; specifically, the execution of a single goal action (e.g. turning off all the lights in a room) may involve the execution of several device actions (e.g. turning off each individual light).

In addition to domain-specific goal actions and device actions, we still have the issue-related actions `findout`, `raise` and `respond` introduced in Larsson *et al.* (2002), and the set of dialogue moves.

For issues, the `resolves` relation provided a way to decide when an issue has been successfully performed and should be popped off the `/SHARED/ISSUES` stack. For actions, we instead need to define *postconditions* which are defined as relations between actions and propositions in the domain resource; these can then be used when to determine when an action can be removed from `/SHARED/ACTIONS`.

4.3.2 Dialogue moves

In addition to the dialogue moves introduced in Larsson *et al.* (2002) and Chapter 2, GODIS4 uses the following two moves:

- `request(α)`, where α : Action
- `confirm(α)`, where α : Action

These two moves are sufficient for activities where actions are performed instantly or near-instantly, and always succeed. If these requirements are not fulfilled, the `confirm` move can be replaced by or complemented with a more general `report(α , Status)` move which reports on the status of action α . Possible values of *Status* could be `done`, `failed`, `pending`, `initiated` etc.; `report(α , done)` would correspond to `confirm(α)`.

4.4 Interacting with menu-based devices

As a sample subtype of action oriented dialogue we will explore menu-based AOD. While menu interfaces are ubiquitous in modern technology they are often tedious and frustrating. The mechanisms of accommodation introduced in Chapter 3 offers the possibility of allowing the user to present several pieces of relevant information at one time or to present information in the order in which the user finds most natural. This means that users can use their own conception of the knowledge space and not be locked to that of the designer of the menu system.

First, we describe a general method for connecting devices to GODIS, and then we show how menu interfaces can be converted into dialogue plans using a simple conversion schema.

4.4.1 Connecting devices to GODIS

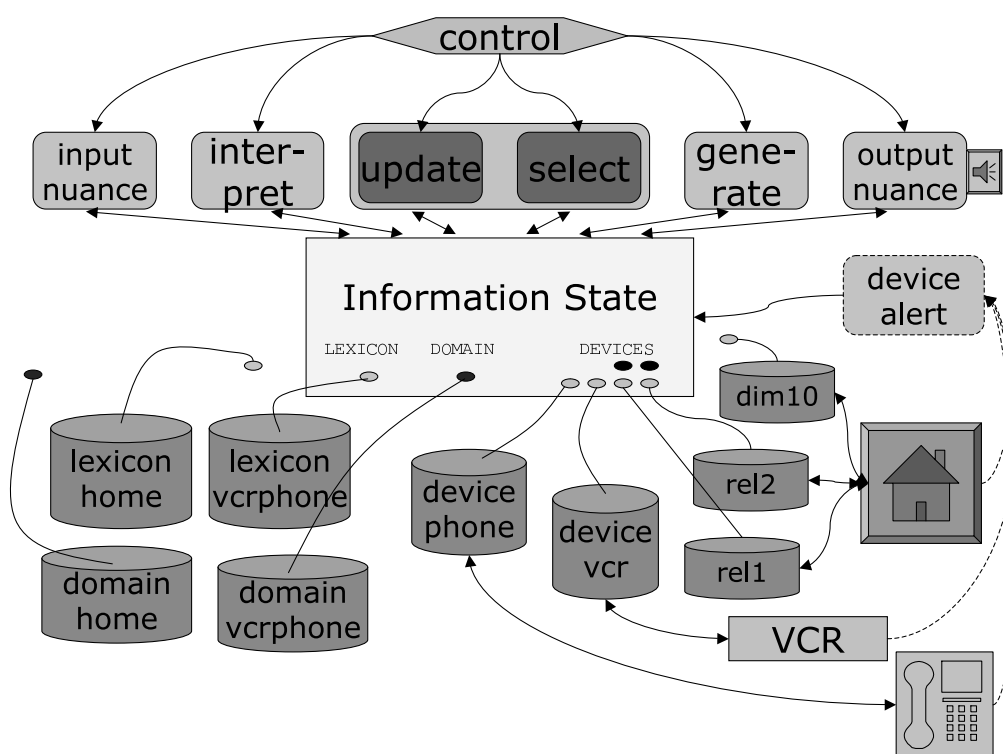


Figure 4.2: Connecting devices to GODIS

In this section we describe briefly how GODIS can interact with devices using the UPnP protocol. In Figure 4.2, we see an impression of how various devices can be connected to GODIS. We

will mainly be dealing with devices that can be modelled as resources, i.e. that are *passive* (or *reactive*) in the sense that they cannot send out information unless queried by some other module. Of course, many devices are not passive in this sense but rather *active* (or *pro-active*), e.g. burglar alarms or robots. To handle active devices, we would need to build a TRINDIKIT module which could write information to a designated part of the information state based on signals from the device; this information could then trigger various processes in other modules. Still, even for an active device the solution we present here would be very useful; minimally, we would only need to add a module which sets a flag in the information state whenever the device indicates that something needs to be taken care of, triggering other modules to query the device about exactly what has happened.

To be able to hook up passive UPnP devices to GODIS, we need the following:

1. device handler resources which communicate directly with the device itself; the device handlers can be said to represent the device in GODIS;
2. a resource type for UPnP devices, specifying how devices may be accessed as objects of this type;
3. a resource interface variable to the TIS whose values are of the UPnP resource type; this variable hooks up devices to the TIS;
4. plan constructs for interacting with devices, and update rules for executing these plan constructs;
5. dialogue plans for interacting with devices.

UPnP device handlers

The device handler mediates communication between GODIS and the device itself, and can be said to represent the device for GODIS. We assume that each specific device has a unique ID, and is accessed via a separate device handler process. A device handler is built for a certain device type (e.g. the Panasonic NV-SD200 VCR), and each device of that type needs to be connected to a process running the device handler, in order to be accessed by GODIS.

For UPnP devices, the device handler contains a specification partly derivable from the UPnP specifications, but made readable for GODIS (i.e. converted from XML to prolog).

The device handler does the following:

- specifies a set of actions and associated arguments

- specifies a set of variables, their range of allowed values, and (optionally) their default value
- routines for setting and reading variables (`dev_set` and `dev_get`), for performing queries (`dev_query`), and for executing actions (`dev_do`)
- accesses the devicesimulation

The UPnP resource interface

In order to hook up a device to GODIS one needs to define an abstract datatype for devices and declare a set of conditions and operations on that datatype. For GODIS, we implement a generic resource interface in the form of an abstract datatype for UPnP devices.

In UPnP, a device is defined in terms of

- a set of variables
- a set of actions with optional arguments

In addition to getting the value of a variable, setting a variable to a new value, and issuing a command, we also add the option of defining *queries* to the device. These queries allow more complex conditions to be checked, e.g. whether two variables have the same value.

Based on this we define the datatype `upnp_dev` as in (1); here, *Var* is a device variable; *Val* is the value of a device variable, *Query* is a question, *Answer* is a proposition, α_{dev} is a device action, and *PropSet* is a set of propositions.

$$(1) \quad \begin{array}{ll} \text{TYPE:} & \text{upnp_dev} \\ \text{REL:} & \left\{ \begin{array}{l} \text{dev_get}(Var, Val) \\ \text{dev_query}(Query, Answer) \end{array} \right. \\ \text{OP:} & \left\{ \begin{array}{l} \text{dev_set}(Var, Val) \\ \text{dev_do}(\alpha_{dev}, PropSet) \end{array} \right. \end{array}$$

Device actions may have one or more parameters; for example, in the VCR control domain there is an action `AddProgram` which takes parameters specifying date, program number, start time, and end time. The *PropSet* argument of `dev_do` is a set of propositions, some of which may serve as arguments to α_{dev} . In the resource interface definition, this set is searched by the device interface for arguments. This means that *PropSet* is not the exact set of arguments needed for α_{dev} ; rather, it is a repository of potential arguments.

The relation between UPnP actions, device actions, and device operations is exemplified below:

- `dev_do(my_vcr, AddProgram)` is a UPnP action, which may appear in a plan
- `AddProgram` is a device action
- `dev_do(AddProgram, {channel_to_store(1), start_time_to_store(13:45), ... })` is a device update operation

In addition to the datatype definition, one can define objects to be of that datatype. For each device that the system should recognize, the device ID should be declared to be of type `upnp_dev`.

4.4.2 From menu to dialogue plan

Having describe a general method for connecting devices to GODiS, we will now show how menu interfaces can be converted into dialogue plans using a simple conversion schema. We assume menu interfaces consist of (at least) the following elements:

- *multi-choice lists*, where the user specifies one of several choices
- *dialogue windows*, where the user enters requested information using the keyboard
- *tick-box*, which the user can select or de-select
- *pop-up messages* confirming actions performed system

The correspondence between menu elements and plan constructs is shown in Table 4.1.

Menu construct	Plan construct
multi-choice list $\langle \alpha_1, \alpha_2, \dots, \alpha_n \rangle$	action to resolve alternative question about action <code>findout({ ?action(α_1), ..., ?action(α_n) })</code>
tick-box or equivalent $\pm P$	action to resolve y/n-question <code>findout(?P)</code>
dialogue window $parameter = _$	action to resolve <i>wh</i> -question <code>findout(?x.parameter(x))</code>
pop-up message confirming α	<code>confirm(α)</code>

Table 4.1: Conversion of menus into dialogue plans

Regarding confirmations, we provide a general solution for confirming actions in Section 4.6.3. Confirmations thus do not need to be included in the plan.

4.4.3 Extending the **resolves** relation for menu-based AOD

In menu-based AOD, the system may ask an alternative-question about which action the user wants the system to perform. The user may then answer by choosing one of the listed alternatives. However, if the user selects an action which is not in the listed alternatives but further down in the hierarchy of actions, this should also be regarded as an answer that **resolves** the system's question. To handle this, we need to extend the definition of the **resolves** relation (see Larsson *et al.* (2002)).

(2) **action**(α) resolves $\{?action(\alpha_1), \dots, ?action(\alpha_n)\}$ if

- $\alpha = \alpha_i$ or
- α_i dominates α ($1 \leq i \leq n$)

The **dominates** relation is defined recursively as in (3).

(3) α dominates α' if

- there is a plan P for α such that P includes $findout(AltQ)$ and $?action(\alpha) \in AltQ$, or
- α dominates some action α'' and α'' dominates α'

The idea is, then, that domination reflects the menu structure so that an action **dominates** any actions below it in the menu.

4.5 Implementation of the VCR control domain

A VCR menu section

We start from a section of the menu structure for a VCR as shown in (4).

- (4)
- toplevel: $\langle \text{change-play-status, change-channel, timer-recording, } \dots \rangle$
 - change play status: $\langle \text{play, stop, } \dots \rangle$
 - change channel
 - * new-channel = _
 - * confirm new channel
 - timer recording: $\langle \text{add-program, delete-program} \rangle$
 - * add program
 - channel-to-store = _
 - date-to-store = _
 - start-time-to-store = _
 - end-time-to-store = _
 - confirm program added
 - * delete program
 - display existing programs
 - program-to-delete: _
 - confirm program deleted
 - change-settings: $\langle \text{set-clock, } \dots \rangle$

Dialogue plans for VCR control

Using the conversion schema in Table 4.1 we can convert the menu structures in (4) into dialogue plans as those shown in (5).

- (5) a. ACTION : vcr_top
 PLAN: \langle
 raise(?*x*.action(*x*))
 findout($\left\{ \begin{array}{l} ?\text{action}(\text{vcr_change_play_status}) \\ ?\text{action}(\text{vcr_new_channel}) \\ ?\text{action}(\text{vcr_timer_recording}) \\ ?\text{action}(\text{vcr_settings}) \end{array} \right\}$)
 \rangle
 POST : -
- b. ACTION : vcr_timer_recording
 PLAN: findout($\left\{ \begin{array}{l} ?\text{action}(\text{vcr_add_program}), \\ ?\text{action}(\text{vcr_delete_program}) \end{array} \right\}$)
 POST : done(vcr_add_program) or
 done(vcr_delete_program)
- c. ACTION : vcr_add_program
 PLAN: \langle
 findout(?*x*.channel_to_store(*x*))
 findout(?*x*.date_to_store(*x*))
 findout(?*x*.start_time_to_store(*x*))
 findout(?*x*.stop_time_to_store(*x*))
 dev_do(vcr, 'AddProgram')
 \rangle
 POST : done('AddProgram')

4.6 Update rules and dialogue examples

In this section we show how update rules for action oriented dialogue have been implemented in GODIS4, and give examples of dialogues from the VCR control domain.

4.6.1 Integrating and rejecting requests

First, we introduce update rules for integrating request moves. Since we are limiting this implementation to domains where the system performs all the actions, we will not provide rules for integrating requests from the system to the user; however, these could be straightforwardly implemented since the relation between system requests and user requests is very similar to the relation between system and user ask moves.

The rule for integrating user requests is shown in (RULE 4.1).

(RULE 4.1) RULE: **integrateUsrRequest**
 CLASS: **integrate**
 PRE: {
 \$/PRIVATE/NIM/FST/SND=request(*A*)
 \$/SHARED/LU/SPEAKER==usr
 \$SCORE=*Score*
 Score > 0.7
 \$DOMAIN :: plan(*A*, *Plan*)
 EFF: {
 pop(/PRIVATE/NIM)
 add(/SHARED/LU/MOVES, request(*A*))
 push(/PRIVATE/AGENDA, icm:acc*pos)
 if_do(*Score* ≤ 0.9,
 push(/PRIVATE/AGENDA, icm:und*pos:usr*action(*A*)))
 push(/SHARED/ACTIONS, *A*)
 push(/PRIVATE/AGENDA, *A*)

This rule is similar to that for integrating user **ask** moves (see Section 2.6.6); instead of pushing an issue *Q* on ISSUES and QUD, and pushing **respond**(*Q*) on the agenda, this rule pushes the requested action *A* on /SHARED/ACTIONS and /PRIVATE/AGENDA.

As for user **ask** moves we also need to deal with the case where the system must reject an action since it does not have a plan for dealing with it. This rule is shown in (RULE 4.2).

(RULE 4.2) RULE: **rejectAction**
 CLASS: **select_action**
 PRE: {
 in(\$/PRIVATE/NIM, request(*A*))
 \$/SHARED/LU/SPEAKER=usr
 not \$DOMAIN :: plan(*A*, *Plan*)
 EFF: {
 del(/PRIVATE/NIM, request(*A*))
 push(/PRIVATE/AGENDA, icm:und*pos:usr*action(*A*))
 push(/PRIVATE/AGENDA, icm:acc*neg:action(*A*))

4.6.2 Executing device actions

The update rule for executing the **dev_do** device action is shown in (RULE 4.3).

(RULE 4.3) RULE: **exec_dev_do**
 CLASS: **exec_plan**
 PRE: $\left\{ \begin{array}{l} \text{fst}(\$/\text{PRIVATE}/\text{PLAN}, \text{dev_do}(Dev, A_{dev})) \\ \text{pop}(/ \text{PRIVATE}/\text{PLAN}) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} ! \$/\text{SHARED}/\text{COM} = PropSet \\ \text{DEVICES}/Dev :: \text{dev_do}(PropSet, A_{dev}) \\ \text{add}(/ \text{PRIVATE}/\text{BEL}, \text{done}(A_{dev})) \end{array} \right.$

The condition looks for a **dev_do** upnp action in the plan, with arguments *Dev*, the device path name, and *A_{dev}*, the device action. The updates pop the action off the plan, and applies the corresponding update **dev_do**(*PropSet*, *A_{dev}*) to the device *Dev*. Finally, the proposition **done**(*A_{dev}*) is added the the private beliefs.

In addition, we have implemented rules for executing the **dev_get**, **dev_set** and **dev_query** actions.

4.6.3 Selecting and integrating confirm-moves

The selection rule for the **confirm** action is shown in (RULE 4.4).

(RULE 4.4) RULE: **selectConfirmAction**
 CLASS: **select_action**
 PRE: $\left\{ \begin{array}{l} \text{fst}(\$/\text{SHARED}/\text{ACTIONS}, A) \\ \$\text{DOMAIN} :: \text{postcond}(A, PC) \\ \text{in}(\$/\text{PRIVATE}/\text{BEL}, PC) \\ \text{not in}(\$/\text{SHARED}/\text{COM}, PC) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{push}(/ \text{PRIVATE}/\text{AGENDA}, \text{confirm}(A)) \end{array} \right.$

The conditions in this rule check that the there is an action in */SHARED/ACTIONS* whose post-condition is believed by the system to be true, however, this is not yet shared information. If this is true, a **confirm** action is pushed on the agenda. Eventually, this action (which also is a dialogue move) is moved to *NEXT_MOVES* by (RULE 4.5).

(RULE 4.5) RULE: **selectConfirm**
 CLASS: **select_move**
 PRE: $\left\{ \begin{array}{l} \text{fst}(\$/\text{PRIVATE}/\text{AGENDA}, \text{confirm}(A)) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{push}(\text{NEXT_MOVES}, \text{confirm}(A)) \\ \text{pop}(/ \text{PRIVATE}/\text{AGENDA}) \end{array} \right.$

When the confirmation move has been made, it is integrated by the rule in (RULE 4.6).

(RULE 4.6) RULE: **integrateConfirm**
 CLASS: **integrate**
 PRE: $\left\{ \begin{array}{l} \$/PRIVATE/NIM/FST/SND=\text{confirm}(A) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{pop}(/PRIVATE/NIM) \\ \text{add}(/SHARED/COM, \text{done}(A)) \end{array} \right.$

This rule adds the proposition **done**(*A*) to the shared commitments which enables the **downdate-Actions** rule in (RULE 4.7) to trigger.

(RULE 4.7) RULE: **downdateActions**
 CLASS: **downdate_issues**
 PRE: $\left\{ \begin{array}{l} \text{fst}(/SHARED/ACTIONS, A) \\ \$DOMAIN :: \text{postcond}(A, PC) \\ \text{in}(/SHARED/COM, PC) \end{array} \right.$
 EFF: $\left\{ \begin{array}{l} \text{pop}(/SHARED/ACTIONS) \end{array} \right.$

This rule removes an action *A* whose postcondition is jointly believed to be true from ACTIONS³.

4.6.4 Dialogue example: menu traversal and multiple threads

In (DIALOGUE 4.1) we show a sample dialogue interaction with the menu-based VCR application. It shows both menu traversal and accommodation, as well as dealing with multiple tasks (issues and actions).

(DIALOGUE 4.1)

S> Welcome to the VCR manager!

S> Lets see. What can I do for you?

U>

\$S> Do you want to change play status , change channel , go to timer recording or go to settings?

³Note that **done**(α) is trivially a postcondition for any action α .

U>go to timer recording

\$S> Okay. Lets see. Do you want to add a program or delete a program?

U> add a program today

S> Okay. Lets see. What channel do you want?

U> what channel is on now

$$\left[\begin{array}{l} \text{PRIVATE} \\ \text{SHARED} \end{array} = \left[\begin{array}{l} \text{AGENDA} = \langle \langle \text{icm:acc*pos} \rangle \rangle \\ \text{PLAN} = \langle \rangle \\ \text{BEL} = \{ \text{current_channel}(1) \} \\ \text{TMP} = \dots \\ \text{NIM} = \langle \langle \rangle \rangle \\ \text{COM} = \{ \text{date_to_store(today)} \} \\ \text{ACTIONS} = \left\langle \begin{array}{l} \text{vcr_add_program} \\ \text{vcr_timer_recording} \\ \text{vcr_top} \end{array} \right\rangle \\ \text{ISSUES} = \langle ?E.\text{current_channel}(E) \rangle \\ \text{QUD} = \langle \rangle \\ \text{PM} = \{ \text{ask}(?A.\text{channel_to_store}(A)), \text{icm:loadplan}, \text{icm:acc*pos} \} \\ \text{LU} = \left[\begin{array}{l} \text{SPEAKER} = \text{usr} \\ \text{MOVES} = \{ \text{ask}(?E.\text{current_channel}(E)) \} \end{array} \right] \end{array} \right] \right]$$

S> Okay. The current channel is 1

S> Returning to add a program. What channel do you want?

U> one

S> Okay. What time do you want to start recording?

U> stop the vcr

device resource issues command to device: [UPnP] Stop

S> Okay. The VCR is now stopped.

S> Returning to add a program. What time do you want to start recording?

U> twelve thirty

S> Okay. What time do you want to stop recording?

U> two fifteen

device resource issues command to device: [UPnP] AddProgram(1,today,1230,215)

S> Okay. The program has been added.

S> Returning to timer recording. Do you want to add a program or delete a program?

N.B.: In this dialogue, the user's question "What channel is on now" is regarded as an irrelevant followup to "What channel do you want?". To change this one could add a dependency between these questions to the domain resource (see Larsson *et al.* (2002)), as in (6).

(6) `depends(?x.channel_to_add(x), ?x.current_channel(x))`

4.6.5 Action accommodation and clarification

We also implement an action-related counterpart to the rule **accommodateDependentIssue** presented in Section 3.6.4. This allows the user to provide parameters for some action without making an explicit request.

(RULE 4.8) RULE: **accommodateAction**
 CLASS: **accommodate**
 PRE: {
 setof(A , $\$/PRIVATE/NIM/ELEM/SND=answer(A)$, $AnsSet$)
 $\$Sarity(AnsSet) > 0$
 $\$DOMAIN :: plan(Action, Plan)$
 $\$DOMAIN :: action(Action)$
 forall(in($AnsSet$, A), in($Plan$, findout(Q)) and
 $\$DOMAIN :: relevant(A, Q)$
 not $\$DOMAIN :: plan(Action', Plan')$ and $Action' \neq Action$ and
 forall(in($AnsSet$, A), in($Plan'$, findout(Q)) and
 $\$DOMAIN :: relevant(A, Q)$
 not in($\$/PRIVATE/AGENDA$, icm:und*int:usr*action($Action$))
 EFF: {
 push($\$/SHARED/ACTIONS$, $Action$)
 push($\$/PRIVATE/AGENDA$, icm:accommodate: $Action$)
 push($\$/PRIVATE/AGENDA$, icm:und*pos:usr*action($action$))
 set($\$/PRIVATE/PLAN$, $Plan$)
 push($\$/PRIVATE/AGENDA$, icm:loadplan)

This rule is very similar to the **accommodateDependentIssue** (see Section 3.6.4), except that it accommodates a dependent action rather than a dependent issue.

If the system finds several actions matching the information given by the user, a clarification question is raised. This is again similar to the behaviour for issues described in Section 3.6.5; in fact, the rule below replaces the previous **clarifyDependentIssue** rule.

(RULE 4.9) RULE: **clarifyIssueAction**
 CLASS: **select_action**
 PRE: {
 in($\$/PRIVATE/NIM$, pair(usr, answer(A)))
 setof($Action$, $\$DOMAIN :: depends(\alpha, Q)$ and
 $\$DOMAIN :: relevant(A, Q)$, $ActionSet$)
 remove_unifiables($Actions$, $Actions'$)
 $\$Sarity(Actions') > 1$
 EFF: {
 ! setof($?IssueProp$, in($Actions'$, $Issue'$) and
 not $\$DOMAIN :: action(Issue')$ and
 $IssueProp=issue(Issue')$, $IssueQuestions$)
 ! setof($?ActionProp$, in($Actions'$, $Action'$) and
 $\$DOMAIN :: action(Action')$ and
 $ActionProp=action(Action)$, $ActionQuestions$)
 ! union($IssueQuestions$, $ActionQuestions$, $AltQ$)
 push($\$/PRIVATE/AGENDA$, findout($AltQ$))

The preconditions check that there is more than one plan (for performing actions or resolving issues) to which all answers in /PRIVATE/NIM are relevant. The first update creates an alternative question regarding all issues which the non-integrated answers are relevant to, and the second update does the same for actions. The third update joins these two alternative-questions (sets of *y/n*-questions) into a single *y/n*-question, which is added to the agenda by the final update.

4.6.6 Dialogue examples: action accommodation and clarification

The dialogue in (DIALOGUE 4.2) demonstrates action accommodation, and the dialogue in (DIALOGUE 4.3) demonstrates a clarification question by the system.

(DIALOGUE 4.2)

```
S> Welcome to the VCR manager!
S> Lets see. What can I do for you?
U> today from five thirty on channel one
S> Alright. add a program.
S> Lets see. Okay. What time do you want to stop recording?
```

(DIALOGUE 4.3)

```
S> Welcome to the VCR manager!
S> Lets see. What can I do for you?
U> six thirty
$S> six thirty. I dont quite understand. Do you want to add a
program or set the clock?
U> add a program
$S> Okay. Lets see. Do you want to record from six thirty or
until six thirty?
U> from six thirty
S> Okay. What channel do you want?
```

4.7 Issues under negotiation in negotiative dialogue

We will now turn to negotiative dialogue, and describe an issue-based account of a simple kind of collaborative negotiative dialogue. We also sketch a formalization of this account and discuss its implementation in GODIS.

We start from a previous formal account of negotiative dialogue (Sidner, 1994a) and argue for a slightly different idea of what negotiative dialogue is. We want to make a distinction between the process of accepting an utterance and its content, which applies to all utterances, and a concept of negotiation defined, roughly, as a discussion of several alternative solutions to some problem. This latter account is formulated in terms of *Issues Under Negotiation* (IUN), representing the question or problem to be resolved, and a set of alternative answers, representing the proposed solutions.

First, we will give a brief review of Sidner’s theory and discuss its merits and drawbacks⁴. We then provide an alternative account based on the concept of Issues Under Negotiation. We explain how IUN can be added to GODIS, and give an information state analysis of a simple negotiative dialogue.

4.7.1 Sidner’s theory of negotiative dialogue

As the title of the paper says, Sidner’s (1994a) theory is formulated as “an artificial discourse language for collaborative negotiation”. This language consists of a set of messages (or message types) with propositional contents (“beliefs”). The effects of an agent transmitting these messages to another agent is formulated in terms of the “state of communication” after the message has been received. The state of communication includes individual beliefs and intentions, mutual beliefs, and two stacks for Open Beliefs and Rejected Beliefs. Some of the central messages are

- `ProposeForAccept (PFA agt1 belief agt2)`: agt1 expresses belief to agt2.
- `Reject (RJ agt1 belief agt2)`: agt1 does not believe belief, which has been offered as a proposal
- `AcceptProposal (AP agt1 belief agt2)`: agt1 and agt2 now hold belief as a mutual belief
- `Counter (CO agt1 belief1 agt2 belief2)`: Without rejecting belief1, agt1 offers belief2 to agt2

In addition, there are three kinds of acknowledgement messages, the most important being `AcknowledgeReceipt (AR agt1 belief agt2)`, which may occur after a `ProposeForAccept` message and results in belief being pushed on the stack for Open Beliefs. Acknowledgement indicates that a previous message from agt2 about belief has been heard; the agents will not hold belief as a mutual belief until an `AcceptProposal` message has been sent.

⁴An in-depth description of Sidner’s account and its relation to the GoDiS system, including a reformulation of Sidner’s artificial negotiation language in terms of GoDiS information state updates, can be found in Cooper *et al.* (2001).

While we will not give a detailed analysis of the effects of each of these messages, some observations are important for the purposes of this paper. Specifically, a counter-proposal (`CO agt1 belief1 agt2 belief2`) is analyzed as a composite message consisting of two PFA messages with propositional contents. The first proposed proposition is `belief2` (the “new” proposal), and the second is `(Supports (Not belief1) belief2)`, i.e. that `belief2` supports the negation of `belief1` (the “old” proposal). Exactly what is meant by “supports” here is left unspecified, but perhaps logical entailment is at least a simple kind of support.

- `(PFA agt1 belief2 agt2)`
- `(PFA agt1 (Supports (Not belief1) belief2) agt2)`

Sidner’s analysis of proposals is only concerned with propositional contents. A Request for action is modelled as a proposal whose content is of the form `(Should-Do Agt Action)`. A question is a proposal for the action to provide certain information. This brings us to our first problem with Sidner’s account.

Problem 1: Negotiation vs. utterance acceptance

In Sidner’s theory, all dialogue is negotiative in the sense that all utterances (except acceptances, rejections, and acknowledgements) are seen as proposals. This is correct if we consider negotiation as possibly concerning meta-aspects of the dialogue. Since any utterance (content) can be rejected, all utterances can indeed be seen as proposals.

So in one sense of “negotiative”, all dialogue is negotiative since assertions (and questions, instructions etc.) can be rejected or accepted as part of the grounding process. But some dialogues are negotiative in another sense, in that they contain explicitly discussions about different solutions to a problem. Negotiation, on this view, is distinct from grounding.

There is thus a stronger sense of negotiation which is not present in all dialogue. A minimum requirement on negotiation in this stronger sense could be that several alternative solutions (answers) to a problem (question or issue) can be discussed and compared before a solution is finally settled on. Sidner is aware of this aspect of negotiation, and notes that “maintaining more than one open proposal is a common feature of human discourses and negotiations.” What we want to do is to find a way of capturing this property independently of grounding and of other aspects of negotiation, and use it as a minimal requirement on any dialogue that is to be regarded as negotiative.

On our view, proposal-moves are moves on the same level as other dialogue moves: greetings, questions, answers etc., and can thus be accepted or rejected on the grounding level. Accepting

a proposal-move on the grounding level merely means accepting the content of the move *as a proposal*, i.e. as a potential answer to a question. This is different from accepting the proposed alternative as the *actual* solution to a problem (answer to a question).

To give a concrete example of these different concepts of negotiativity, we can compare the dialogues in Examples (5) and (6).

- (7) A : Today is January 6th.
propose proposition
B(alt. 1) : Uhuh
accept proposition
B(alt. 2) : No, it's not!
reject proposition
- (8) S : where do you want to go?
ask question
U : flights to paris on september 13 please
answer question
S : there is one flight at 07:45 and one at 12:00
propose alternatives, give information about alternatives
U : what airline is the 12:00 one
ask question
S : the 12:00 flight is an SAS flight
answer question
U : I'll take the 7:45 flight please
accept alternative, answer question "which flight?"

The type negotiation in (7) concerns acceptance-level grounding of the utterance and its content. By contrast, the type of negotiation in (8) concerns domain-level issues rather than some aspect of grounding.

Problem 2: Alternatives and counterproposals

When analyzing a travel agency dialogue (Sidner, 1994b), the travel agent's successive proposals of flights are seen as counterproposals to his own previous proposals, each modelled as a proposition. The difference between proposals and counterproposals is that the latter not only make a new proposal but also proposes the proposition that the new proposal conflicts with the previous proposal (by supporting the negation of the previous proposal). This can be seen as an attempt by

Sidner to establish the connection between the two proposals as somehow concerning the same issue.

This analysis is problematic in that it excludes cases where alternatives are not mutually exclusive, which is natural when e.g. booking a flight (since the user presumably only want one flight) but not e.g. when buying a CD (since the user may want to buy more than one). Also, it seems odd to make counterproposals to your own previous proposals, especially since making a proposal commits you to intending the addressee to accept that proposal rather than your previous ones. In many cases (including the travel agency domain) it seems that the agent may often be quite indifferent to which flight the user selects. Travel agents may often make several proposals in one utterance, e.g. “There is one flight at 7:45 and another one at 12:00”, in which case it does not make sense to see “one at 12:00” as a counterproposal as Sidner defines them.

We do not want to use the term “counterproposal” in these cases; what we need is some way of proposing alternatives without seeing them as counterproposals. The basic problem seems to be that when several proposals are “on the table” at once, one needs some way of representing the fact that they are not independent of each other. Sidner does this by adding propositions of the form `(Supports (Not belief1) belief2)` to show that belief1 and belief2 are not independent; however, this proposition not only claims that the propositions are somehow dependent, but also that they are (logically or rhetorically) mutually exclusive. In our view, this indicates a need for a theory of negotiation which makes it possible to represent several alternatives as somehow *concerning the same issue*, independently of rhetorical or logical relations between the alternatives. Negotiation, in our view, should not in general be seen in terms of proposals and counterproposals, but in terms of proposing and choosing between several alternatives.

4.7.2 Negotiation as discussing alternatives

In this section, we will attempt to provide a more detailed description of negotiative dialogue. Clearly, negotiation is a type of problem-solving (Di Eugenio *et al.*, 1998). We define negotiative dialogue more specifically to be *dialogue where DPs discuss several alternative solutions to a problem (issue) before choosing one (or several) of them*. In line with our issue-based approach to dialogue management, we propose to model negotiable problems (issues) semantically as questions and alternative solutions as alternative answers to a question.

We also propose to keep track of issues under negotiation and the answers being considered as potential solutions to each issue in the /SHARED/ISSUES field, represented as questions associated with sets of answers.

Degrees of negotiativity

Starting from this definition, we can distinguish between fully negotiative dialogue and semi-negotiative dialogue. In non-negotiative dialogue, only one alternative can be discussed. In semi-negotiative dialogue, a new alternative can be introduced by revising parameters of the previous alternative; however, previous alternatives are not retained. Finally, in negotiative dialogue: several alternatives can be introduced, and old alternatives are retained and can be returned to.

Semi-negotiative information-oriented dialogue does not require keeping track of several alternatives. All that is required is that information is revisable, and that new database queries can be formed from old ones by replacing some piece of information. This property is implemented in a limited way for example in the Swedish railway information system (a variant of the Philips system described in Aust *et al.*, 1994), which after providing information about a trip will ask the user “Do you want an earlier or later train?”. This allows the user to modify the previous query (although in a very limited way) and get information about further alternatives. However, it is not possible to compare the alternatives by asking questions about them; indeed, there is no sign that information about previous alternatives is retained in the system. The implementation of reaccommodation in GODIS3 (Section 3.6.6) also allowed semi-negotiative dialogue in this sense.

Factors influencing negotiation

There are a number of aspects of the dialogue situation which affect the complexity of negotiative dialogues, and allows further sub-classification of them. This sub-classification allows us to pick out a subspecies of negotiative dialogue to implement.

On our definition, negotiation does not require conflicting goals or interests, and for this reason it may not correspond perfectly to the everyday use of the word “negotiation”. However, we feel it is useful to keep collaborativity (i.e. lack of conflicting goals) as a separate dimension from negotiation. Also, it is common practice in other fields dealing with negotiation (e.g. game theory, economy) to include collaborative negotiation (cf. Lewin *et al.*, 2000).

A second factor influencing negotiation is the distribution of information between DPs. In some activities, information may be symmetrically distributed, i.e. DPs have roughly the same kind of information, and also the same kind of information needs (questions they want answered). This is the case e.g. in the Coconut (Di Eugenio *et al.*, 1998) dialogues where DPs each have an amount of money and they have to decide jointly on a number of furniture items to purchase. In other activities, such as a travel agency, the information and information needs of the DPs is asymmetrically distributed. The customer has access to information about her destination, approximate time of travel etc., and wants to know e.g. exact flight times and prices. The travel

agent has access to a database of flight information, but needs to know when the customer wants to leave, where she wants to travel, etc.

A third variable is whether DPs must commit jointly (as in e.g. the Coconut dialogues) or one DP can make the commitment by herself (as e.g. in flight booking). In the latter case, the acceptance of one of the alternatives can be modelled as an answer to an IUN by the DP responsible for the commitment, without the need for an explicit agreement from the other DP. In the former case, a similar analysis is possible, but here it is more likely that an explicit expression of agreement is needed from both DPs. This variable may perhaps be referred to as “distribution of decision rights”. In some dialogues (such as ticket booking) one DP has the decision rights for all negotiable issues; in this case there is no need for explicitly representing decision rights. However, if decision rights are distributed differently for different issues, an explicit representation of rights is needed.

Ticket booking dialogue, and dialogue in other domains with clear differences in information and decision-right distribution between roles, has the advantage of making dialogue move interpretation easier since the presence of a certain bits of information in an utterance together with knowledge about the role of the speaker and the role-related information distribution often can be used to determine dialogue move type. For example, an utterance containing the phrase “to Paris” spoken by a customer in a travel agency is likely to be intended to provide information about the customer’s desired destination.

4.7.3 Issues Under Negotiation (IUN)

In this section we discuss the notion of Issues Under Negotiation represented by questions, and how proposals relate to such issues. We also discuss how this approach differs from Sidner’s.

Negotiable issues and activity

Which issues are negotiable depends on the activity. For example, it is usually not the case that the name of a DP is a negotiable issue; this is why it would perhaps seem counterintuitive to view an introduction (“Hi, my name is NN”) as a proposal (as is done in Sidner, 1994b). However, it cannot be ruled out that there is some activity where even this may become a matter of negotiation. Also, it is usually possible in principle to make any issue into a negotiable issue, e.g. by raising doubts about a previous answer (see Section 4.8.2) .

Alternatives as answers to Issues Under Negotiation

Given that we analyze Issues Under Negotiation as questions, it is natural to analyze the alternative solutions to this issue as potential answers. On this view, a proposal has the effect of adding an alternative answer to the set of alternative answers to an IUN. For a DP with decision rights over an IUN, giving an answer to this IUN is equivalent to accepting one of the potential answers as the actual answer. That is, an IUN is resolved when an alternative answer is accepted.

Here we see how our concept of acceptance differs from Sidner. On our view a proposed alternative can be accepted in two different ways: as a proposal, or as *the* answer to an IUN. Accepting a proposal move as adding an alternative corresponds to meta-level acceptance. However, accepting an alternative as the answer to an IUN is different from accepting an utterance. Given the optimistic approach to acceptance, all proposals will be assumed to be accepted *as proposals*; however, it takes an **answer**-move to get the proposed alternative accepted as the solution to a problem.

Semantics

To represent issues under negotiation, we will use pairs of questions (usually *wh*-questions but possibly also *y/n*-questions) and sets of proposed answers. This is in fact an alternative representation of alternative-questions to that which we have used previously. The additional semantic representation is shown in (9).

- (9) $Q \bullet AnsSet : AltQ$ if $Q : WHQ$ (or $Q : YNQ$) and $AnsSet : Set(ShortAns)$

4.7.4 An example

In the (invented) example in Figure 4.3, the question on ISSUES is **?*x.desired_flight(x)***, i.e. “Which flight does the user want?”. The user supplies information about her desired destination and departure date; this utterance is interpreted as a set of **answer**-moves by the system since it provides answers to questions that the system has asked or was going to ask. As a response to this, the system performs a database search which returns two flights **f1** and **f2** matching the specification, and stores the database results in /PRIVATE/BEL. The system then proposes these flights as answers to the current IUN. The system also supplies some information about them. As a result, the IUN is now associated with two alternative answers, **f1** and **f2**. Finally, the user provides an answer to the current IUN, thereby accepting one of these alternatives as the flight she wants to take.

A> flights to paris, june 13
 answer(desired_dest_city(paris))
 answer(desired_dept_date(13/5))

B> OK, there's one flight leaving at 07:45 and one at 12:00
 propose(f1)
 propose(f2)
 inform(dept_time(f1,07:45))
 inform(dept_time(f2,12:00))

PRIVATE	AGENDA	=	< findout(?x.desired_flight(x)) >
		=	< findout(?x.credit-card-no(x)) >
		=	< updateDB(add_reservation) >
	PLAN	=	{ flight(f1)
		=	{ dept_time(f1,0745)
SHARED	BEL	=	{ ...
		=	{ dept_time(f1,0745)
	COM	=	{ dept_time(f2,1200)
		=	{ desired_dest_city(paris)
	ISSUES	=	{ desired_dept_date(13/5) ... }
		=	{ ?x.desired_flight(x) • { f1, f2 } }
LU	ACTIONS	=	< book_ticket >
		=	< >
	XS QUD	=	< >
		=	< >
	SPEAKER	=	sys
		=	{ propose(f1)
MOVES		=	{ propose(f2)
		=	{ ... }
		=	{ ... }
		=	{ ... }
		=	{ ... }

A> I'll take the 07:45 one
 answer(desired_flight(X)&dept_time(X, 07:45))
 (after contextual interpretation: answer(desired_flight(f1)))

PRIVATE	AGENDA	=	$\langle \text{findout}(?x.\text{credit-card-no}(x)) \rangle$	
		PLAN	=	$\langle \text{updateDB}(\text{add_reservation}) \rangle$
		BEL	=	$\left\{ \begin{array}{l} \text{flight}(\text{f1}) \\ \text{dept_time}(\text{f1}, 0745) \\ \dots \end{array} \right\}$
	COM	=	$\left\{ \begin{array}{l} \text{desired_flight}(\text{f1}) \\ \text{dept_time}(\text{f1}, 0745) \\ \text{dept_time}(\text{f2}, 1200) \\ \text{desired_dest_city}(\text{paris}) \\ \text{desired_dept_date}(13/5) \dots \end{array} \right\}$	
SHARED	ISSUES	=	$\langle \rangle$	
	ACTIONS	=	$\langle \text{book_ticket} \rangle$	
	QUD	=	$\langle \rangle$	
	LU	=	$\left[\begin{array}{ll} \text{SPEAKER} & = \text{sys} \\ \text{MOVES} & = \left\{ \begin{array}{l} \text{answer}(\text{desired_flight}(\text{f1})) \\ \dots \end{array} \right\} \end{array} \right]$	

Figure 4.3: Example dialogue
 182

This dialogue does not include any discussion or comparison of alternatives, but it could easily be extended to cover e.g. the dialogue in (5.8).

4.8 Discussion

4.8.1 Negotiation in inquiry-oriented dialogue

The model presented here is not committed to the view that negotiation only takes place in the context of collaborative planning, or even action-oriented dialogue. In the sense of negotiative dialogue used here, i.e. dialogue involving several alternative solutions to some problem, negotiation may also concern matters of fact. This can be useful e.g. in tutorial dialogue where a tutor asks a question, gives some alternative answers, and the student's task is to reason about the different alternatives and decide on one of them. In the travel agency domain, it is often not necessary to explicitly represent e.g. that deciding on a flight is a precondition of a general plan for travelling; instead, we can represent it simply as a fact concerning which flight the user wants to take.

A related point is that collaborative planning dialogue is not necessarily action-oriented dialogue, since the activity of planning may be directed at coming up with an abstract plan regardless of who actually performs the actions in the plan. Only when some DP becomes obliged to carry out some part of the plan does the dialogue become what we refer to as an action-oriented dialogue.

4.8.2 Rejection, negotiation and downshifting

In the context of discussing referent identification in instructional assembly dialogues, Cohen (1981) makes an analogy between shifts in dialogue strategy and shifting gears when driving a car. In a dialogue in high gear, the speaker introduces several subgoals in each utterance, whereas fewer goals are introduced in low-gear dialogue. The type of subgoals discussed by Cohen are mainly identifying a referent, requests to pick up objects, and requesting an assembly action. As long as the dialogue proceeds smoothly and the hearer is able to correctly identify referents and carry out actions, the speaker requests assembly actions and expects the hearer to be able to identify and pick up the objects referred to without explicit requests for this. However, when this fails and the hearer fails to identify a referent, the speaker may shift into a lower gear (downshift) and make explicit requests for identification of referents. At a later stage, the speaker may shift to a higher gear and request the hearer to pick up an object and then to perform an assembly action. Finally, the speaker may return to the initial gear and only make requests for assembly actions.

Severinsson (1983) views to the process of downshifting as making *latent* subgames into *explicit* subgames. In the case mentioned above, the goals of the latent subgames are (1) to get the hearer to identify a referent, and (2) for the hearer to pick up the object referred to. In high gear, these subgames are latent in the sense that they do not give rise to any utterances (dialogue moves). When the latent subgames become explicit, the process that was previously carried out silently is instead carried out using utterances.

This view fits well with the concept of tacit moves introduced in Section 3.4.2. Updates for latent referent identification and utterance acceptance can be regarded as tacit moves (or games) corresponding to explicit referent identification or negotiation subdialogues, similar to the way that question accommodation updates are tacit moves corresponding to the **ask** dialogue moves.

Both these notions, shifting gears in dialogue and latent subgames, are useful for shedding light on the relation between negotiative dialogue and utterance acceptance. Firstly, the notions of optimism and pessimism regarding grounding strategies seem intimately related to the notion of gears, both metaphorically and factually. Metaphorically, we may say that an optimistic driver will use a higher gear than a pessimistic one; only when she encounters a bumpy road will she shift into lower gear (thus taking a more pessimistic approach). Later, when the road becomes smoother, she may again resume her optimistic strategy and use a higher gear. Similarly, speakers can be expected to switch between higher and lower gears, and between optimistic and pessimistic grounding strategies regarding the grounding of their utterances. Thus we claim that the notion of shifting gears is applicable not only to referent identification, but also to other grounding related games, including utterance acceptance.

In Chapter 2, we talked about optimism and pessimism in regard to grounding on the acceptance level; we now add that DPs may shift gears regarding grounding on the acceptance level. In a dialogue in high gear, the speaker optimistically assumes the hearer to accept her utterances. However, should the speaker reject some utterance, the dialogue is downshifted and the latent uptake subgame becomes explicit. We would claim (contrary to Sidner) that it is only when the dialogue is downshifted in this sense that moves such as questions and assertions should be regarded as proposals. At this stage, DPs may introduce alternatives to the proposal, and they may argue for or against proposals.

The concept of downshift is related to Ginzburg's case where a proposition p is rejected as a fact but $?p$ is accepted as a question for discussion. This appears to be a potential case of downshifting which could be modelled by regarding $?p \bullet \{\text{yes}, \text{no}\}$ as an issue under negotiation. In addition, alterations of p may be proposed, roughly corresponding to Clark's "cooperative alterations". It appears this can be modelled as an issue under negotiation $?x.p_x \bullet \{a, b, \dots\}$ (where p_x is the proposition p with some argument a replaced by x , and thus $p = p_x(a)$). The alterations are then represented as alternatives b, \dots to a .

Thus, if a question q has been raised in a dialogue and if an answer a relevant to q is rejected (on the grounding level), q may become negotiable (depending on the activity). If so, the DP

who rejected a may propose an alternative answer a' to q . It is then possible for the DPs to start a (probably argumentative) negotiation regarding which of a and a' , or perhaps some other answer, should be accepted as the answer to q . We thus believe that downshifting of dialogue from optimistic acceptance to negotiation can shed light on various grounding-related phenomena, e.g. alterations (see Section 2.2.1), and the relation between grounding and negotiation.

4.8.3 Dialogue structure and issue-based dialogue management

In this section we discuss the implications of issue-based dialogue management on the structure of dialogue. We discuss the dialogue model of Grosz and Sidner (1986), elaborated in Grosz and Sidner (1987), and relate it to the issue-based model. The authors present a theory of discourse structure based on three structural components:

- linguistic structure: utterances, phrases, clauses etc.
- intentional structure: intentions, related by dominance and satisfaction-precedence
- attentional state: salient objects, properties, relations and discourse intentions

The intentional structure is related to dialogue structure through *Discourse Segment Purposes (DSPs)*. A dialogue can be divided into segments where each segment is engaged in for the purpose of satisfying a particular intention, designated as the DSP of that segment. This relation is used to explain the close correspondence between task structure and dialogue structure observed in collaborative planning dialogue. With regard to dialogue management, it is claimed that “a conversational participant needs to recognize the DSPs and the dominance relationships between them in order to process subsequent utterances of the discourse” (Grosz and Sidner, 1987, p. 418). The authors also sketch a process model based on the concept of a *SharedPlan*, formalized in terms of individual intentions and mutual beliefs.

There are some interesting but rough correspondences between this model and the issue-based model, and the latter can perhaps be seen (at least to some extent) as an alternative (or complement) to the SharedPlans formalization.

The simplest correspondence is that between the linguistic structure and the LU field (and perhaps also the INPUT variable) in the issue-based model, although our model of linguistic structure is extremely impoverished.

In the issue-based model, DSPs roughly correspond to the ISSUES and (in AOD) ACTIONS fields, and should thus be useful for segmenting dialogue in a manner similar to Grosz and Sidner’s.

Sequencing ICM, which (among other things) reflect changes in ISSUES can thus be regarded as indicating dialogue segment shifts.

The local focus of attention is partially modelled by QUD, although so far our attentional model lacks e.g. a representation of “objects under discussion”. Discourse intentions seem to correspond roughly to the AGENDA field, and possibly also the PLAN field although the latter is more global in nature. Of course, our representation of dialogue plans is quite different from that of Grosz and Sidner, who use a modal logic with operators for intentions.

It should be noted that the intentional structure, modelled as SharedPlans, is part of the shared knowledge. Grosz and Sidner are primarily interested in dialogues aimed at the collaborative creation and execution of these SharedPlans, which means that their model does not trivially extend to other kinds of dialogue, e.g. simple inquiry-oriented dialogue or tutorial dialogue. For the kinds of dialogue we have dealt with so far, the kind of complex representations needed for modelling SharedPlans appear not to be needed. The closest correspondence to SharedPlans in our model is the ACTIONS field which contains domain actions to be performed by one of the DPs. It can be expected that when the issue-based model is extended to handle collaborative planning dialogue, the structure of the ACTIONS field will become more complex and more similar to SharedPlans.

4.9 Summary

Firstly, we extended the issue-based approach to action-oriented dialogue, and implemented a dialogue interface to a VCR where dialogue plans were based on an existing menu interface. We modified the information state by adding a field /SHARED/ACTIONS, and also added two new dialogue moves specific to AOD **request** and **confirm**. We also implemented update rules in GODIS to handle integration and selection of these moves, as well as interaction with a device, and also provided an additional accommodation rule for actions.

Secondly, we proposed a view of negotiation as discussing several alternative solutions to an issue under negotiation. On our approach, an issue under negotiation is represented as a question, e.g. what flight the user wants. In general, this means viewing problems as issues and solutions as answers. This approach has several advantages. Firstly, it provides a straightforward and intuitively sound way of capturing the idea that negotiative dialogue involves several alternative solutions to some issue or problem, and that proposals introduce such alternatives. Secondly, it distinguishes two types of negotiation (grounding-related negotiation and negotiation of issues) and clarifies the relation between them.

Chapter 5

Conditional responses

5.1 Introduction

A flexible and natural dialogue is characterized by different ways of responding to questions. For instance, typical responses in an information-seeking dialogue (ISD) are the short answers described in Larsson *et al.* (2002). Short answers which contain just the response particles *yes*, *no* or *ok* only affirm or negate the propositional content of the question without conveying any additional information. However, there are situations in which such answers may be insufficiently collaborative with respect to the task. For instance, in the travel domain, the task is to determine a set of parameters of a possible journey with respect to a database to which only the system has access. In a typical application like GODIS, the system collects from the user a set of parameters constraining a journey by asking questions. It then performs a database search with these constraints. If the search succeeds, GODIS returns the price of the journey (10e). Otherwise, it indicates the search has failed (10f).

- (10)
- a. S: Welcome to the travel agency!
 - b. U: The price for a flight from Malmö to Paris on the first of April please.
 - c. S: What class did you have in mind?
 - d. U: Economy.
 - e. S: The price is 7654 crowns.
 - f. S': Sorry, there is nothing matching your request about price.

However, a dialogue like (10) often does not stop at the result of the database search, be it negative or positive. The user may revise or refine some parameter(s) and initiate a new search. For example, after (10f), the user may continue by changing the departure day as in (11a). Then, after (11b), the user may continue by trying to further constrain the search by specifying an additional parameter, for example the airline as in (11c). However, such continuations can become dull as the user is trying to find out which combinations of parameters succeed, as in (11c)-(11f).

- (11) a. U: Can I fly on the second?
b. S: Yes.
c. U: Can I fly with Ryanair?
d. S: Sorry, there is nothing matching your request.
e. U: What about Lufthansa?
f. S: Sorry, there is nothing matching your request.

It has been argued that task-oriented dialogues are natural and efficient when they are collaborative (Chu-Carroll and Brown, 1997; Rich *et al.*, 2000). Collaboration means that both the system and the user are contributing to solving the task at hand. ISDs are a particular kind of task-oriented dialogues. A typical ISD system however does not enable any collaboration in the process of determining the journey parameters: The system does not provide the user with any indications of what journeys are (still) available in the database. The user has to “blindly” specify her desires, and equally blindly revise them, refining (if they are under-constraining) or relaxing them (if they are over-constraining). One reason for that in the travel domain may be that it is not a viable option for the system to guide the user in the initial specification of journey parameters. It is impossible to enumerate all available options for individual parameters since the number of potential journeys in the database is typically large. On the other hand, once the search space is restricted by setting some initial set of parameters, the system could be collaborative in the subsequent phases. This is what we are trying to achieve.

A way to model a collaborative system behavior in ISDs is to enable the system help the user in finding a satisfiable set of parameters by providing responses that help the user to revise or refine the initial parameters. This involves the system being able to indicate a parameter to relax upon failed database search, or to indicate a parameter to keep in cases where some hits are found but they are too many to enumerate, and thus the search criteria need to be refined.

One useful way of accomplishing collaboration in the parameter revision and refinement phase is by providing a *conditional response* (CR): a positive or negative response clarifying the condition(s) under which this response holds. It is the making of the condition explicit that makes the response collaborative. For example, a negative CR can be collaborative by mentioning such

parameter(s) in the condition, whose relaxation could result in a positive response instead (*economy class* in (12b)). A positive CR can be collaborative by mentioning such parameter(s) that are necessary for preserving the positive response (*business class* in (12c)).

- (12) a. U: Can I fly on the second?
b. S: Not if you want to fly economy class.
c. S': Yes, if you can fly business class.

In (12a) (as a continuation of (10f)), the user changes the departure day to April 2nd. In (12b) the system not only gives a negative answer, but also indicates that the failure of the database search with the changed parameter is conditional on the parameter *economy class* which the user has specified earlier. In an alternative response (12c) in this context, the system suggests *business class* as an alternative for which the database query would be successful.

The result of the database search, and therefore the answer to a user's question, can also be contingent on a parameter the user has not specified. In this case, too, it makes sense to indicate this contingency to the user: in (13b) the system gives a positive answer to the question and also indicates that the database search is successful (and thus the answer to the question is positive) as long as an additional parameter, namely the SAS airline, is assumed.

- (13) a. U: Can I fly on the second?
b. S: Yes, if you can fly with SAS.

Another example is (14) where the response of the system is contingent on the citizenship of the user.

- (14) a. U: Do I need a visa to enter the U.S.?
b. S: Not if you are a EU citizen.

There are various ways to realize this collaborative system behavior other than by using CRs. For instance, a response like (15b) also proposes an alternative after a failed database search with the user specified parameters. However, it does not indicate the reason why the database search has failed as in the case of (12b).

- (15) a. U: Can I fly on the second?
b. S: No, but you can fly on the third.

This chapter is organized as follows. In the next section we describe the nature of CRs. Section 5.2.1 focuses on the meaning of CRs in terms of their semantic content and some implicatures they give rise to. In section 5.2.2, we describe the dialogue behavior of CRs in terms of the conditions of their use and the dialogue moves they realize. Section 5.3 describes the implementation of CRs in GODiS. Issues of further research are discussed in 5.4.

5.2 The nature of conditional responses

In this section we present our analysis of CRs. This analysis is supported by the results of a small corpus study we conducted using the SRI's American Express (AMEX) travel agency data (AMEX (1989)) and the Verbmobil appointment-scheduling corpus (Verbmobil-Corpus (1995)).

In the corpora, we looked at conditional expressions like English *if*, *unless*, *as long as* and German *wenn*, *es sei denn*, *solange* in combination with response particles like *yes*, *no*, *ok* and the negation *not*. Apart from explicitly conditional utterances like the ones in (16) (AMEX (1989)) and (17) (Verbmobil-Corpus (1995)), we also found implicitly conditional ones where an adverbial is in the scope of the negation (18) (AMEX (1989)), (19) (Verbmobil-Corpus (1995)).

- (16) a. A: and penalty plus we pay for the going rate
b. B: **Yes, if it is going to be a higher rate.**

- (17) a. A: how does that sound to you
B: **okay unless you want to try to squeeze it in on Thursday**
b. A: have you time in the afternoon I have a date but before
B: **only if we meet early in the morning**

- (18) a. A: Any other flights?
b. B: **Not from Oakland.**

- (19) B: **Not on Mondays.**

The positive CRs can be implicitly positive (without a *yes*) (20a) (AMEX (1989)), (20b) (Verbmobil-Corpus (1995)).

- (20) a. A: can I make the reservation and and change it by to-morrow?
 B: **if it's still available**, right
- b. A: what should be Thursday the twentieth?
 B: **but only in the afternoon**

We also observed that such responses are typically elliptical. For instance, the CR in (16b) can intuitively be expanded to the complete propositions as illustrated in (21).

- (21) No, there are no other flights if you want to fly from Oakland.

Similarly, the positive CR in (18b) can be expanded to (22).

- (22) Yes, you pay for the going rate, if it is going to be a higher rate.

CRs can be however also non-elliptical sentences.

We observed that all these expressions have similar semantic properties and exhibit a similar behavior in dialogue. We will describe those in turn in what follows and give corpus examples to illustrate them.

5.2.1 Meaning

The prototypical CRs we currently consider have the form *Not if c / Yes if c*. They are elliptical utterances. The material for resolving the ellipsis comes from the immediately preceding context. In the ISU approach we work with, ellipsis is resolved with respect to the current *question under discussion* QUD (see Section 3.3). CRs are typically used as answers to yes/no-questions. However, they can be also used as answers to wh-questions as in (23b)-(23c) (from Verbmobil-Corpus (1995)).

- (23) a. A: I think we should definitely meet this month
- b. B: okay what date would be good
- c. A: **almost any day as long as it is not the weekend or Wednesday**
- d. B: okay then how about a Thursday
- e. A: Thursday evening would that be fine

The latter example suggests also that the condition introduced by a CR may be complex, e.g. a disjunction.

On the other hand, CRs can be also used as acknowledgments to preceding assertions like in (24) (from AMEX (1989), A and C refer to *Agent* and *Customer* respectively) where the CR (24h) is acknowledging/accepting the assertion in (24g)).

- (24) a. C: now she said that if we don't ticket if we do ticket the eleven twenty four today and that Lufthansa flight comes up you know la-
- b. A: mmm hmmm
- c. C: later and the- then we have to cancel that first fare
- d. A: uh huh
- e. C: there's a hundred dollar fine
- f. A: okay
- g. C: and p- penalty plus ahh we pay for the going rate
- h. A: **yeah** if ther- **if it's going to be a higher rate**
- i. C: it's going to be a higher rate, ok now if we wait and the Lufthansa flight does come up what rate do we get that eleven twenty four or something close to that

In the ISU approach, QUD models a conversation as the process of setting up possible questions to discuss and the subsequent resolving of some of these questions. A discourse participant (DP) can choose at any time to add something to the QUD or to address one of the questions on QUD. This view treats an assertion like *I want to travel economy class* as an answer to the question on QUD *What class did you have in mind?*. Similarly, (24g) will be treated as addressing some possible question under discussion, e.g. *What happens if we cancel that first fare*. The CR in (24h) would be then treated as an answer to the question on QUD *Is there a penalty and do we pay for the going rate?*.

Assertion vs implicature

A CR does not address a QUD by providing an answer simpliciter: It provides an answer that is *contingent* on the value of some attribute. Consider again (25).

- (25) a. U: Can I fly on the second?
- b. S: Not if you want to fly economy class.
- c. S': Yes, if you can fly business class.

The system's reply (25b) provides an answer that is contingent on the value of the *class* attribute. If the value is (or implies) economy, the answer is *negative*: If the user flies economy, she cannot fly on the second. This CR also seems to suggest the contrapositive that if the value is "non-economy", i.e. a contextual alternative to the user specified parameter which in this context happens to be "business class", the answer is positive. (25c) illustrates this case. Thus, CRs suggest that the respective polarity of the answer concerns only the case where the parameter specified in the condition is set to the respective value, whereas for a different value of this parameter the answer may have the opposite polarity. We propose to consider this additional suggestion as an *implicature*, rather than part of the assertion that CRs express. An argument in favor of this treatment is the fact that this suggestion seems to be cancellable. Thus in (26), the suggestion in (26i) that there might be flights from the other airport discussed earlier, namely San Francisco, is cancelled in the next utterance (26j) (this example is from AMEX (1989)).

- (26) a. A: uh, let's see what would get you there then leaving
probabl- the seventh. from San Jose or San Francisco?
- b. C: San Francisco. actually Oakland would be good too
on that
- c. A: I don't know if there are any red eyes from there let's
see
- d. C: ok
- e. A: there is one on United that leaves Oakland at eleven
thirty p.m. and arrives Chicago five twenty five a.m.
- f. C: so that's a two hour hold there
- g. A: yes
- h. C: waiting for that flight ok any others?
- i. A: uh **not from Oakland.**
- j. A: departing from San Francisco it's about the same

Thus, as an answer to a question $?p$, a CR not only answers it (positively or negatively), but it also implicates that if the condition c does not hold, the answer would have the opposite polarity.

Table 5.1: Patterns of conditional responses

	<i>Negative CR</i>	<i>Positive CR</i>
<i>QUD</i>	? <i>p</i>	? <i>p</i>
<i>Response</i>	Not if <i>c</i>	Yes if <i>c</i>
<i>Assertion</i>	If <i>c</i> , then $\neg p$	If <i>c</i> , then <i>p</i>
<i>Implicature</i>	If <i>c'</i> , then <i>p</i>	If <i>c'</i> , then $\neg p$

Table 5.1 summarizes the patterns of CRs. A negative answer means that the propositional context of the question is negated. On the other hand, the alternative to the condition *c* suggested in the implicature is not simply a negation of *c* but a *contextual alternative* *c'* of it.

Other conditional expressions like *unless* also seem to restrict the validity of the response and to propose an alternative as a condition which, when it applies, would reverse the polarity of the response. Consider (27b). B's answer to A's question is negative with respect to the (alternative) parameters "on Monday Tuesday and Wednesday in the morning" and proposes an alternative as a condition which, when it applies, would reverse the polarity of the response.

- (27) a. A: I have my mornings completely free do you have any time then
- b. B: No it does not look like it unless you might be able to squeeze it in on Tuesday afternoon

A similar case is (28) where the response to the QUD (28a) is positive if the condition introduced by *unless* does not apply and negative otherwise. (Both examples are from Verbmobil-Corpus (1995)).

- (28) a. A: how does that sound to you?
- b. B: okay unless you want to try to squeeze it in on Thursday the 13th from anywhere

Translated into the patterns in Table 5.1, a CR of the form *Yes, unless c* which resolves to *p*, *unless c* (where ?*p* is a question on QUD), asserts *if c', then p* and implicates *if c, then $\neg p$* . Here, *c'* is what currently holds.

5.2.2 Dialogue behavior

We describe the dialogue behavior of CRs in terms of conditions of use and dialogue moves. In this section we present two different types of CRs depending on whether the condition expresses a parameter already established in the context or not, and discuss in what contexts they are appropriate. The dialogue moves CRs initiate are also discussed, as well as the question when to use a negative and when a positive CR.

Conditions of use

As our corpus study revealed, CRs can be used in two types of context: (a) when the parameter on which the CR is contingent has not yet been determined in the preceding context (or cannot be assumed), or (b) when it has been determined. CRs are discussed in Green and Carberry (1999) and characterized in terms of the speaker's motivation to provide information "about conditions that could affect the veracity of the response". However, only case (a) is considered where the speaker does not know whether the condition holds, while utterances in which the condition is already specified are left unnoticed.

We have found examples in the Verbmobil appointment-scheduling corpus (Verbmobil-Corpus (1995)) and the SRI American Express (AMEX) travel agency data (AMEX (1989)) supporting our distinction. An instance of the former case is (24), for the latter (26).

Consequently, we distinguish between two types of CRs with respect to the contextual status of the parameter in the condition on which the CR is contingent:

- (i) CRs contingent on a contextually-determined parameter (**CDCRs**)
- (ii) CRs contingent on a contextually non-determined parameter (**NDCRs**)

We discuss each of the cases below.

CRs with not-determined parameter. The parameter on which a CR is contingent can be one that has not yet been determined in the preceding context. We call this type of CR a non-determined parameter CR, or NDCR for short. Besides the assertion and the implicature that answer the question on QUD as specified in Table 5.1, the CR amounts to indirectly giving rise to the implicit question "whether *c* holds". Support to this intuition is given in (24i) where the customer answers positively the implicit question in (24h).

The example shows also that the implicitly raised question is such that cannot be answered just by "yes" or "no". Rather, it requires some content that matches with the condition *c*. Consider

the user's utterances in (29d)-(29f).

- (29) a. U: Do I need a visa to enter the U.S.?
b. S: Not if you are an EU citizen.
c. S': Yes, if you are not an EU citizen.
d. U: Yes. | No.
e. U': Yes, I am. | No, I am not.
f. U'': Yes, I have German citizenship. | No, I have Czech citizenship.

The responses in (29d) could be interpreted as acknowledgments, but certainly not as answers to whether the user is an EU citizen. One way to model this in the ISU approach is that the implicit question does not become the top-most QUD instead of the explicit question whether a visa is needed, i.e., the latter question is still “pending”. This is corroborated by the following continuation of (29e) where the system does answer the pending question.

- (30) a. S: Then you do (not) need one.
b. S': Then you do (not) need a visa.

(30a) is elliptical for (30b). Correct resolution of the ellipsis is possible only if the question whether the user needs a visa is still the top-most QUD.

On the other hand, the need to answer the implicitly raised question is dependent on what goals the participants try to achieve. For example, the question “Do I need a visa?” in (29a) is satisfactorily dealt with when either a *yes* or *no* answer is given, or when enough information is provided such that the asker can find out the answer herself. On the other hand, consider (31).

- (31) a. U: Can I fly to Paris tomorrow?
b. S: Not if you want to fly economy class.

In (31) the response is contingent on whether the user wants to fly economy class. Before flight selection can proceed further, the question whether *c* holds must be answered. The difference between (31) and (29) is that in order to satisfy its goal of selecting a flight which satisfies the user requirements, the system does need to know whether *c* holds, because it needs to find out whether *p* holds (in (31)). In (29), the goal is merely to answer the user question. This is

modeled straightforwardly in the ISU approach, because raised questions need not be answered, in contrast to findout actions, see Larsson *et al.* (2002)).

CRs with contextually determined parameter. Another context in which a CR is appropriate is when an answer to a question is contingent on a parameter that has already been established in the preceding context. We call this type of CR contextually determined CR or CDCR for short. What does a CDCR communicate besides the assertion and implicature that answer the question as specified in Table 5.1? We suggest that it initiates a negotiation about the already established parameter. However, this cannot happen by simply raising the question whether *c* holds, because *c* has already been established. We suggest that a CDCR implicitly proposes to the user to consider *changing* the parameter: It *reraises* the question whether *c* holds (see Section 2.6.9 on question reraising). Thus, a negotiation phase is opened in which either the conflicting parameter is revised, or is confirmed. In the latter case, a different solution to the overall goal needs to be sought.

Reraising *c* differs from raising a “new” question at least in two aspects: *c* must be *negotiable*, and reraising *c* means it cannot be answered simply by providing a sufficiently discriminative positive or negative response. To see the difference, consider (32).

- (32) a. U: Can I fly on the second?
b. S: Not if you want to fly economy class.
c. U: Yes. | No.
d. U': Yes, I do. | No, I don't.
e. U'': Yes, I want business class. | No, I don't want business class.

Similarly to the case with a non-determined *c* in (29), the responses in (32c)-(32e) cannot be interpreted as an answer to whether the user wants to change her mind from business to economy class. They seem hard to interpret even as acknowledgments. But then we observe a number of differences from the non-determined case: The responses in (32d) and (32e) are not appropriate as answers to the implicitly reraised *c*, because a revision of a parameter is involved. Hence, some kind of acknowledgment of the revision is needed in addition to the answering whether or not the parameter is to be revised (and how). Such acknowledgments are present in (33).

- (33) a. U: OK, I can fly ECONOMY.
b. U': But I DO want business class.

In (33a), *OK* can be seen as acknowledging the revision from business to economy class. In (33b), *but* acknowledges the contrast between the proposed revision and the actual preservation of the parameter (here, business class). The continuation in (34) on the other hand refuses the proposed revision only implicitly, but proposes instead to check the flight possibilities on another day.

(34) U: How about Tuesday?

Another observation concerning the appropriateness of a CDCR is that a CR cannot immediately follow after an utterance in which the value is established, as the inappropriateness of (35b) and (35c) illustrates.

- (35) a. U: Can I fly business class from Saarbrücken to Paris on Sunday?
b. S: Not if you want BUSINESS class.
c. S': Yes if you want ECONOMY class.

Intuitively, the reason for this is that there needs to be some degree of uncertainty (in the sense of being assumed but not known to be shared) about the parameter in order for a conditional to be felicitous. For example, in (12) on p. 189, the business class requirement is assumed to be maintained when the day is revised. The inappropriateness of (35b) and (35c) can also be explained on purely semantic grounds. When both the assertion and the implicature as specified in Table 5.1 are taken into account, a contradiction arises: Given that the elliptical answer is resolved to the previous utterance, (35b) asserts *If user wants business class, then a business flight from Saarbrücken to Paris on Sunday is not available*, and implicates *If user does not want business class, then a business flight from Saarbrücken to Paris on Sunday is available*. Similarly for (35c). (For the contradiction, the modalities need to be ignored.)

The implicit question

Sometimes, the question that is implicitly raised by a NDCR and reraised by a CDCR can be made explicit. As a matter of fact, there are cases in which making this question explicit increases the informativity and cooperativity of the response. Thus, a bare CR like (36b) may be not sufficiently informative compared to a response like (36c) where the CR is followed by a *verification question*.

- (36) a. U: Can I fly on the second?
b. S: Not if you fly economy class.
c. S': Not if you fly economy class. Do you want to fly business class instead?

This response seems to be more efficient with respect to the overall task of finding an appropriate flight since it indicates that it is important that the user answers it. As noted earlier, there are cases in which the implicit question needs not be answered. Thus in (14) on p. 189, the answer of the user does not influence the overall task. A criterion for deciding when it is reasonable to make the implicit question (re)raised by CRs explicit can be thus the question whether the progress of solving the task depends on the answer of the user, that is in the ISU approach, whether it is on the domain plan or on the agenda.

Negative or positive CR

Another issue concerning the conditions of use of CRs is the question about the choice between producing the negative or the positive version of the respective type of CR. It seems that the choice is made along two dimensions. First, the form of the user question may indicate certain preferences of the user towards the answer. For instance, questions like (37) indicate intuitively a positive answer preference, whereas (38) a negative one. This is irrespective of being a CDCR or NDCR.

- (37) Can I/Is it possible to fly on Monday?

- (38) Do I need to/Is it necessary to fly on Monday?

On the other hand, a positive CDCR like the one in (39d) seems to be marked in the context of a failed search. On the second thought the problem seems to be not the positive form but the fact that the speaker suddenly introduces a contextual alternative of the parameter which was discussed earlier. This is supported by the intuition that a more appropriate positive response is (39e) where the contrastive conjunction *but* indicates that the speaker is aware of this change from the parameter that has been discussed so far to its contextual alternative.

- (39) a. U: I want to fly economy class.
 b. S: What day do you want to travel?
 c. U: Can I travel on the first?
 d. S: Yes, if you fly business class.
 e. S': Yes, but only if you fly business class.

Furthermore, in some cases a negative NDCR evokes a presupposition that the condition *c* holds:

- (40) a. Do I need a visa to go to the US?
 b. Not if you are an EU-citizen. \Rightarrow I suppose you are an EU-citizen.

Similarly in the positive case, cf. (41).

- (41) Yes, if you are a non EU citizen. \Rightarrow I suppose you are a non-EU citizen.

More neutral seems to be (42a) or even more so (42b).

- (42) a. S: Only if you are a non EU citizen.
 b. S': Not if one is a EU-citizen.

This presupposition effect is however context dependent. Another dimension along which the system can choose between a negative or positive NDCR seems to be the economy of expression. That is, in cases in which more than one alternative to a parameter is found, the polarity of the response can be varied depending on which way of presenting the results would be more efficient. Thus, given the situation in (43a), a positive CR should be preferred.

- (43) a. U: Can I fly with SAS?
 database contains flights on Monday and Tuesday
 b. S: Not if you fly on Wednesday, Thursday, Friday, Saturday or Sunday.
 c. S': Yes, if you fly on Monday or Tuesday.

Dialogue moves

As we have seen, the two different kinds of CRs provide a response contingent on a parameter where a CDCR makes a proposal for revising the contextually determined parameter, and a NDCR raises the question whether the parameter holds. This suggests that CDCR and NDCR perform different dialogue moves. A CDCR makes a proposal for revising the contextually determined parameter, and a NDCR raises the question whether the parameter holds.

In the Verbmobil corpus, the affirmative or rejecting part of a CR is annotated either as an ACCEPT or REJECT act which are sub-concepts of the acts FEEDBACK_POSITIVE and FEEDBACK_NEGATIVE. These subconcepts are specified as a move by which the speaker explicitly accepts or rejects a proposal respectively. The conditional part of a CR is labeled as a SUGGEST act defined as an act with which the speaker proposes an explicit instance or aspect of the negotiated topic (Alexandersson and et al. (1998)).

We believe that the Verbmobil annotation scheme does not allow to account for the complex-siri role of CRs in dialogue. CRs perform multiple dialogue acts: they are a response (backward-looking function) and a question or a proposal (forward-looking function) at the same time. Neither does this annotation scheme allow a distinction between CDCRs and NDCRs.

No dialogue act annotation is available for the AMEX-SIRI corpus.

We propose to characterize CRs in terms of the DAMSL standard for dialogue annotation (Allen and Core (1997)) in the following way:

Forward looking function. We propose to assign CDCRs the forward looking function of *open option* (an *open option* move suggests a course of action but puts no obligation), and NDCRs the one of *assert* (since the other dialogue participant is not always obliged to provide an answer to the implicit question).

Backward looking function. We propose to assign CRs multiple backward looking functions. Both CDCRs and NDCRs are assigned the function *answer* (if preceded by a question) or *non-answer*,¹ otherwise. In addition, CDCRs can be assigned the function *partial reject/accept* (depending on polarity), and NDCRs the function *hold* (a *hold* move leaves the decision open pending further discussion). This proposal is presented in Table 5.2.

5.3 Implementation

Implementing CRs in GODIS involves both production and interpretation of this kind of responses on the part of the system. We consider currently only the case where CRs are responses

¹E.g. *confirm*.

Table 5.2: Dialogue moves for CRs

	Contextually determined c (CDCR)	Contextually non-determined c (NDCR)
Function:		
backward-looking	answer / confirmation reject / accept part	answer / confirmation hold
forward-looking	open option	assert
Effect on context:	“Should c be kept?”	“Does c hold?”

of a DP to a yes/no-question of the other DP.

As already argued, the **interpretation** of a CDCR as an answer to a question $?p$ is that (i) it is determined whether p or $\neg p$ holds, because (ii) the answer is contingent on a condition c and c is established. Moreover, (iii) the CDCR indicates the reason for the answer. By reminding of the parameter on which the response is contingent, a CDCR (iv) proposes to reconsider the earlier made decision by implicitly reraising the question whether c should hold. The interpretation of a NDCR as a response to a question $?p$ is that (i) it is still not determined whether p , because (ii) the answer is contingent on c , and thus (iii) the question whether c holds is implicitly raised.

From the **production** point of view, we observed that it is appropriate to produce a CDCR when (i) responding to a question $?p$, where (ii) the response is either p or $\neg p$, depending on a parameter c which has been established in the preceding context and is negotiable. An NDCR is appropriate to produce when (i) responding to a QUD $?p$, where (ii) the response is either p or $\neg p$, depending on some additional parameter c which has not yet been established in the context.

We also argued that for both NDCR and CDCR, the choice between a positive or a negative one depends on the question which one is more efficient and more cooperative in the context. The latter in turn depends on what the preferred answer to the question whether p is assumed to be.

Both the interpretation and the production of CRs requires various extensions and modifications of the update and selection rules as well as the analysis, generation and search components in GODIS. That is, the GODIS functionality needs to be extended in several ways. First of all, we need the system to be able to treat *questions about availability* of a particular parameter of a flight like the ones in (44).

- (44) a. Can I fly on the second?
b. Do I have to fly business class?
c. Is it possible to fly on Monday?

Such questions involve searching the database with respect to the parameter specified in the question. Providing a CR to such a question requires in turn an extended database search involving the identification of parameters responsible for a failure as well as of such guaranteeing successful search. We focus on these issues in what follows.

5.3.1 User questions

As already indicated, implementing CRs requires to enable them under certain circumstances as answers to user questions about the availability of flights. Currently, GODIS deals with user questions with respect to parameters like flight prices and visa requirements for destination countries. In GODIS, any user question is dealt with by providing a domain plan for dealing with it (Larsson *et al.* (2002)). Thus, GODIS provides currently domain plans for dealing with questions about price and visa information. These plans specify inter alia the database search with respect to the two parameters.

To deal with user questions concerning other parameters, we need to extend the treatment of user questions in the GODIS system for the travel agency domain. To this end, we implement a domain plan for answering such questions. This plan is specified below.

```

PLAN:  ( exists(X),
(45)   [ findout(?X1.dept_city(X1)))
        consultDB(X) ] ).

```

One condition that has to be ensured to hold before consulting the database is whether the system has enough flight parameters to search the database with. For instance, in the TA domain, at least the departure city should be known. Otherwise, the system should not start a database search, since the set of search results would be too big to be presented in a sensible way. This is dealt with by the first action in the plan which leads to a system question concerning the departure city. The case where the departure city is already known is taken care of by the rule **removeFindout** which pops an element of the private plan of the system if this element is already in /SHARED/COM (Larsson *et al.* (2002)). The plan for answering user question about the availability of particular parameters of a flight (other than the price) is embedded into the more global plan for finding out the price of a flight.

The user question is then integrated in the IS by the existing integration rule **integrateUsrAsk** (see Section 2.6.6). Below, we include for illustration a special rule **integrateUsrAskExists** for the case where the user question has the form *ask(exists(*X*))*. The rule **integrateUsrAskExists** fires whenever there is a nonintegrated ask-move of the form *ask(exists(*X*))* which represents a user question about availability of some parameter *X* of a flight.

(RULE 5.10) RULE: **integrateUsrAskExists**
 CLASS: **integrate**
 PRE: {
 \$/PRIVATE/NIM/FST= A
 $A/\text{FST} == \text{usr}$
 $A/\text{SND} = \text{ask}(\text{exists}(B))$
 $\text{exists}(B) = C$
 \$DOMAIN :: plan(C, D)
 EFF: {
 pop(/PRIVATE/NIM)
 push(/PRIVATE/AGENDA, icm:acc*pos)
 ! \$SCORE= E
 if_then_else($E \leq 0.7$,
 push(/PRIVATE/AGENDA, icm:und*int:usr*issue(C)),
 [add(/SHARED/LU/MOVES, ask(C))
 if_do($E \leq 0.9$, push(/PRIVATE/AGENDA,
 icm:und*pos: usr*issue(C)))
 if_do(in(\$/SHARED/ISSUES, C) and
 not fst(\$/SHARED/ISSUES, C),
 push(/PRIVATE/AGENDA, icm:reraise: C))
 if_do(in(\$/SHARED/COM, F) and \$DOMAIN :: resolves(F, C),
 [del(/SHARED/COM, F)
 if_do(in(\$/PRIVATE/BEL, F), del(/PRIVATE/BEL, F))
 push(/PRIVATE/AGENDA, icm:reraise: C)
])
 push(/SHARED/ISSUES, C)
 push(/SHARED/QUD, C)
 push(/PRIVATE/AGENDA, respond(C))
 add(/SHARED/LU/MOVES, ask(C))
])
 }

In principle, the propositional content of the question is not put onto SHARED/COM. If the question revises a parameter that has already been specified earlier, the parameters provided in the question itself take precedence in the search; for example in answering (46e) the search needs to be for a flight on April 2nd even though the user earlier specified the departure day as April 1st. The parameters specified in a follow-up question should not replace the earlier specified parameters. (46f) is an example where the immediate revision could lead to trouble.

- (46) a. S: Welcome to the travel agency!
- b. U: The price for an economy flight from Frankfurt to Paris on April first please.
- c. S: Sorry, there is nothing matching your request about price.
- d. S': The price is 200 Euro.
- e. U: Can I fly on the second?
- f. S: Not if you want to fly economy class.
- g. U: Can I fly from Luxembourg?

Intuitively, what the user is asking about in (46g) are economy flights from Luxembourg to Paris on April 1st. However, the immediate revision strategy would lead to a search for economy flights from Luxembourg to Paris on April 2nd. This arises as follows: In (46b), the user specifies the parameters as *dept_city(frankfurt) & dest_city(paris) & month(april) & dept_day(first)*. No matter whether the search result is negative (46c) or positive (46d), the user may want to explore other possibilities by asking (46e). If the system uses the immediate revision strategy, the departure day parameter will now be set to *dept_day(first)*, and remain that also when (46g) is being interpreted, which is wrong. The revision should therefore be at least conditioned upon success of the database search and possibly also upon the user's subsequent acknowledgment. This solution also allows the system to make inference about parameters in case the user accepts the system's suggestion as will be proposed in section 5.3.3.

After the plan for answering user questions about availability of flight parameters is executed, the global price plan is recovered.

Alternatively, user questions about availability of flights could be dealt with by the integration rule for user questions itself. We find this second possibility more intuitive in the sense that while it is sensible to treat questions about visa requirements as a separate plan largely independent from the task of specifying parameters of a flight (other than the destination city), asking for the availability of parameters specifying a flight is a natural component of this task. This is alternatively implemented by modifying the update rule **integrateUsrAskExists** for dealing with user questions in such a way that the database search with respect to the parameter specified in the user question is triggered directly:

RULE: **integrateUsrAskExists**
 CLASS: integrate

$$\begin{array}{lcl}
\text{PRE:} & \left\{ \begin{array}{l} \$/PRIVATE/NIM/FST=A \\ A/FST==usr \\ A/SND=ask(exists(B)) \\ exists(B)=C \end{array} \right. & \\
\text{EFF:} & \left\{ \begin{array}{l} \text{pop}(/PRIVATE/NIM) \\ \text{push}(/PRIVATE/AGENDA, \text{icm:acc*pos}) \\ ! \$SCORE=D \\ \text{if_then_else}(D \leq 0.7, \\ \text{push}(/PRIVATE/AGENDA, \text{icm:und*int:usr*issue}(C)), \\ \text{[add}(/SHARED/LU/MOVES, \text{ask}(C)) \\ \text{if_do}(D \leq 0.9, \text{push}(/PRIVATE/AGENDA, \text{icm:und*pos:usr*issue}(C))) \\ \text{if_do}(\text{in}(\$/SHARED/ISSUES, C) \text{ and not fst}(\$/SHARED/ISSUES, C), \\ \text{push}(/PRIVATE/AGENDA, \text{icm:reraise:}(C)) \\ \text{if_do}(\text{in}(\$/SHARED/COM, E) \text{ and } \$DOMAIN :: \text{resolves}(E, C), \text{[del}(/SHARED/COM, E) \\ \text{if_do}(\text{in}(\$/PRIVATE/BEL, E), \text{del}(/PRIVATE/BEL, E)) \\ \text{push}(/PRIVATE/AGENDA, \text{icm:reraise:}(C)) \\ \text{])} \\ \text{push}(/SHARED/ISSUES, C) \\ \text{push}(/SHARED/QUD, C) \\ \text{push}(/PRIVATE/PLAN, \text{consultDB}(B)) \\ \text{push}(/PRIVATE/AGENDA, \text{respond}(C)) \\ \text{add}(/SHARED/LU/MOVES, \text{ask}(C)) \\ \text{])} \end{array} \right. &
\end{array}$$

The crucial difference is that the rule above gives rise to an immediate database search with respect to the parameter in the user question by pushing the action *consultDB(X)* onto the plan-stack of the system. This is however a deviation from the notion of a domain plan as a fixed set of actions (see Larsson *et al.* (2002)).

A question that needs to be answered here is what happens when the user starts the information-seeking dialogue by asking a question about availability as in (47).

- (47) a. S: Welcome to the travel agency!
b. U: Can I fly on the second of march?

Although this is not likely that this case would happen, the system should be able to deal with it. As already mentioned, it would be wrong to start a database search, if at least the departure city is not known yet, since the set of search results would be too big to be presented in a sensible way. Both in the case where database search is triggered via a domain plan and as a direct effect of the user question, the system is not able to deal with the situation described in (47). In the case where a plan deals with the user question, the system loads the respective plan for dealing with

this user question. This plan however does not provide for the case in (47) (neither does the plan for answering user questions about visa requirements implemented as part of GODIS1 (Larsson *et al.* (2002))). The reason for that is that the rule for integrating user questions puts a respond action on the private agenda of the system. The system tries then to respond to the user question. What it should do however in the case of (47) is to ask the user about the departure day. This action is however part of a domain plan which cannot be accessed unless the private agenda is empty. If the agenda is empty, the rule **selectFromPlan** fires which moves the first item of the plan to the agenda. One possible way to solve this problem is to use additional plan constructs as described in Larsson *et al.* (2002) which would ensure that certain conditions hold prior to trying to answer the user question.

In the case where no domain plan is used for dealing with user questions, the system has not loaded any plan yet and thus does not know how to interpret the question of the user. To remedy this, a condition like the one in (48) can be added to the effects of the rule **integrateUsrAskExists** which then would trigger database search only after it has been checked whether the departure city is already established, and if not, would push the respective *findout* action on the agenda (provided that this is the only item on the agenda).

(48) if_do (not(in(\$/SHARED/ISSUES, dept_city(A)))),
 push(/PRIVATE/AGENDA, ?A.dept_city(A))

5.3.2 The search

Another extension of GODIS needed in order to enable it to produce CRs concerns the process of consulting the database. This is necessary since producing a particular kind of response (at least in information-seeking dialogues) is driven by the database search results.

In GODIS, the database search is triggered by the database consultation action *consultDB(q)* where *q* is a question. The database search looks up the answer to the question in the database. When consulting the database, the values of some parameters are known, and the values of others are requested from the database. In GODIS, a single parameter is requested, namely the one specified in the question *q*. The parameters used for the search consist of the parameters provided earlier (if any), retrieved from the shared-commitments part of the information state (see Larsson *et al.* (2002)).

In GODIS, the result of a database search is either (i) a proposition specifying a unique value of the requested parameter (realized as in *The price is 7654 crowns*),² (ii) a proposition **fail**(*q*) indicating that no answer to the question was found (realized as in *Sorry, there is nothing matching your request about price*) or (iii) a set of alternative answers (realized as in (49d)).

²In case more than one result is found, the system enumerates them one by one.

- (49) a. U: I want to fly from Malmö to Paris on the first of April
- b. S: What class did you have in mind?
- c. U: It doesn't matter.
- d. S: The price is 7654 crowns. business class. The price is 456 crowns. economy class.

The resulting proposition(s) are the answer to the question q . The result of a successful database search is a set of propositions of the form $db_entry(set(SpecProps), set(UnspecProps), Answer)$. This set is stored in PRIVATE/BEL.

Producing CRs requires a certain processing of the search results. On the one hand, we want to produce a negative CR instead of plain *No* after failed DB search to suggest a parameter the user might consider relaxing to get successful search. Similarly, a positive CR can also be generated instead of plain *No* after failed database search in order to suggest an alternative parameter leading to successful search. This involves the identification of a parameter responsible for the failure and new search with this parameter relaxed. On the other hand, we want to produce positive CR instead of plain *Yes* after successful search to indicate that search success depends on some as yet unspecified parameter, to avoid future failed search and to constrain the space of possibilities. This involves comparison of database records in the case that more than one record satisfies the query and identifying a parameter that all records share.

For instance, given the database in figure 5.1, if the search fails with the parameters specified by the user but succeeds with one parameter relaxed, then it is possible for the system to suggest this parameter to the user to consider relaxing. In this case the system can produce a CR like the CDCR in (50d). Alternatively, it can suggest directly the alternative for a parameter and produce the respective positive CR in (50e).

- (50) a. U: I want to fly from London to Hongkong on the second of March, cheap
Search parameters:
 $\{dept_city(london), dest_city(hongkong), dept_day(second), month(march), class(economy)\}$
Search fails:
- b. S: Sorry, there is nothing matching your request.
- c. U: Can I travel on the first?
Modified search parameters:
 $\{dept_city(london), dest_city(hongkong), dept_day(first), month(march), class(economy)\}$
Search fails with class(economy) but succeeds with class(business):
- d. S: Not if you want economy class.
- e. S': Yes, if you fly business class.

On the other hand, if the database returns many hits, and all of them share some as yet unspecified parameter(s) on which the potential search is contingent, a positive NDCR can be produced to indicate this parameter to the user (51d). Alternatively, a negative NDCR may be produced to indicate potential failure with the contextual alternative of this parameter (51e).³

³This negative CR is however rather marked and suggests that the system has a model of the user preferences.

Figure 5.1: A toy database

dep	dest	month	dep_day	class	price	airline
London	Hongkong	March	1 st	business	1555	BA
London	Hongkong	April	2 nd	economy	654	BA
London	Hongkong	April	2 nd	economy	654	BA
London	Hongkong	March	3 rd	business	1555	BA
London	Hongkong	March	3 rd	economy	654	BA
London	Hongkong	March	3 rd	economy	854	SAS
London	Hongkong	March	3 rd	economy	854	SAS

- (51) a. U: I want to fly from London to Hongkong on the first of April, cheap
Search parameters:
 $\{dept_city(london), dest_city(hongkong), dept_day(first), month(april), class(economy)\}$
Search fails:
- b. S: Sorry, there is nothing matching your request.
- c. U: Can I travel on the second?
Modified search parameters:
 $\{dept_city(london), dest_city(hongkong), dept_day(second), month(april), class(economy)\}$
Search succeeds with two hits, additional parameter shared by all DB records: airline(ba)
Potential success indicated:
- d. S: Yes, if you fly with British Airways.
Potential failure indicated:
- e. S': ?Not if you want to fly with SAS.

Thus, in order to produce CRs, additional processing of the search results is needed. The database is looked up with a set of parameters retrieved from /SHARED/COM. This set is matched onto the database. The parameter specified in the user question is the one that is looked for. If the matching succeeds, the standard procedure described above provides that the result in the form of a proposition or a set of alternative answers is written to PRIVATE/BEL of the system as already described above.

If no matching database record is found, the set of parameters is compared with the database records and the difference list is obtained. The difference list contains the elements of the set

of user specified parameter that could not be matched in the DB. For instance, in the case described in (50), the difference list after lookup with a modified parameter set would contain the parameters *class(economy)*, *class(business)*.

Then, this difference list is checked for relaxable parameters. If it does not contain relaxable parameters, then there is no solution to the user query and the system generates an utterance like *Sorry, there is nothing matching your request about X* where *X* is the requested parameter that was looked up. Alternatively, the system could say just *No*. If it does contain relaxable parameters, they are collected in a list. The first element of this list is taken and its counterpart (contextual alternative) is deleted from the original set of user specified parameters. The so obtained relaxed set of parameters provides the parameters for a new search. For instance, in (50), the set of parameters for the new search will consist of the following elements: $\{dept_city(london), dest_city(hongkong), dept_day(first), mont(march)\}$. Searching with this set of parameters may again succeed or fail. In case it succeeds, a list is obtained which contains the possible alternatives to the relaxed parameter, as well as some parameters not contained in the user query, that is, some unspecified parameters. Finally, from the list of alternatives the element(s) is selected which can be identified as the counterpart(s) (contextual alternative(s)) of the relaxed parameter. In case the new search fails, another parameter is relaxed and a new search with a relaxed set of parameters is conducted. If no parameter can be relaxed, the system answers with a non-conditional negative response.

The result of the successful modified search with parameter relaxation has the form *db_entry(set(SpecProps), relaxable(Relaxable), alternative(Alternative), set(UnspecProps), Answer)* where *relaxable(Relaxable)* is the set of parameters identified as responsible for the failure and *alternative(Alternative)* is the set of its contextual alternative and *Answer* is the parameter requested in the user question. This set is stored in PRIVATE/BEL.

The parameter relaxation procedure we described is simplified in several respects. First of all, the question which parameters are relaxable and which not is dealt with by just declaring certain parameters for being relaxable. Currently, these are the parameters class and departure day. Also, the relaxation strategy is to relax these parameters one after another in some arbitrary order until a solution is found. Moreover, we currently only model the relaxation of one parameter p_j , but it is conceivable to look for a combination of parameters. A more sophisticated way to deal with this issue is by choosing some fixed ordering of parameters for relaxation, possibly using the user's preferences provided that the system had a user model.

Some of the problems we are dealing with in providing alternative suggestions to the user in information-seeking dialogues in over-constrained situations, have been addressed in work concerned with conflict resolution (Qu and Beale, 1999; Chu-Carroll and Carberry, 1994) (see Qu and Beale (1999) for further references). The general issue of negotiation strategies after failed database search is presently being addressed by various teams developing commercial dialogue systems, e.g., the Soliloquy system. Another such system we have recently seen demonstrated is developed at IBM (Hochberg *et al.* (2002)). Qu and Beale (1999) for instance propose a

constraint-based model for cooperative response generation aiming at detecting and resolving situations in which the user's information needs have been over-constrained. In contrast to earlier work using heuristics for identifying relaxation candidates (e.g., based on constraint weights -Abella *et al.*, 1996; Pieraccini *et al.*, 1997), Qu and Beale (1999) employ AI techniques like constraint satisfaction, solution synthesis and constraint hierarchy. There is work on detecting invalid beliefs or plans and suggesting alternative solutions by relaxing over-constrained queries and proposing relaxation modification (Chu-Carroll and Carberry (1994)) which is also related to the issues presented above.

In contrast to the works cited above, we also deal with *under*-constrained situations where a positive CR indicates additional parameters in order to prevent failed future search. The only other work addressing also this issue we are aware of (Hochberg, p.c.) is described in Hochberg *et al.* (2002) but does not provide details in this respect.

Employing more sophisticated parameter identification and relaxation techniques is however beyond the scope of our present work.

When the user query succeeds and the user question about availability can be answered positively, the database may contain additional parameters which the user has left unspecified or which the system has not asked about (for instance, airline). Our system can be collaborative especially in the case where there are more results than can be sensibly conveyed to the user at once. A CRs is appropriate in this case to indicate when the success of the search depends on any as yet unspecified by the user. This helps to avoid future failed search. This relies on the database search returning all the records that satisfy the search criteria. We implement a simple subsequent processing which compares the records and determines whether they all have any other parameter(s) in common. When this is the case, a positive NDCR with this parameter as a condition can be generated. Currently, we only cover the case where there is only one such parameter. The case where there are more than one additional parameters to suggest we leave for future work. The database search result after this processing has the format *db_entry(set(SpecProps), additional(Additional), set(UnspecProps), Answer)* where *additional(Additional)* is the set of parameters that all successful records share.

A negative NDCR can be alternatively produced by using the contextual alternative of the additional parameter. This requires however to define contextual alternatives in the domain knowledge of the system (see the discussion about that in 5.3.3).

An alternative (but much simpler) way to deal with the case of providing additional parameters after successful search is already provided by the standard database consultation procedure without additional processing of the search results. A positive NDCR is produced then after a question about availability when only one database record is retrieved which contains unspecified parameters. One (or two) of them can be picked up and realized as the condition of the positive NDCR. For instance, given the database in figure 5.1, a positive NDCR as (52d) can be

produced.⁴

- (52) a. U: I want to fly from London to Hongkong on the first of April.
Search parameters:
 $\{dept_city(london), dest_city(hongkong), dept_day(first), month(april)\}$
Search fails:
- b. S: Sorry, there is nothing matching your request.
- c. U: Can I travel on the second?
Modified search parameters:
 $\{dept_city(london), dest_city(hongkong), dept_day(second), month(march)\}$
Search succeeds, unspecified parameters found:
class(economy), airline(ba)
Potential success indicated:
- d. S: Yes, if you want to fly economy class with British Airways.
Potential failure indicated:
- e. S': ?Not if you want to fly business class with SAS.

The modified database consultation procedure we described makes sure that a contextual alternative to the condition asserted in a CDCR is found prior to producing the response. Since we argued that for a system it is desirable to be cooperative, we suggest that the system should produce a (negative or positive) CDCR after failed search only if it has found an alternative satisfying also the implicature that arises from the CDCR. In comparison, a human agent can behave differently in this respect. In (26) on p. 193, the human travel agent produces a negative CR upon discovering that the answer is negative for Oakland. However, it seems that she does this prior to checking whether the answer is positive for the contextual alternative San Francisco. Eventually, no solution is found for the alternative departure city either, and asserting this in (26j) cancels the implicature of (26i).

For NDCRs, the database consultation procedure makes sure that the additional parameter suggested has no alternative in the database for which the answer would be positive, which satisfies the implicature arising from positive NDCRs. Similar considerations hold for the negative NDCRs where the procedure makes sure that the implicature is satisfied by identifying a parameter which is shared by all database records matching the user query.

After having set the stage for the implementation of CRs, in what follows we describe the actual implementation of CRs starting with their production.

⁴Again, the negative NDCR is marked and requires a user model.

5.3.3 Production of CRs

In this section we describe the production of CRs by the system. The dialogues we want the system to produce are presented in (53) (CDCR) and (54) (NDCR) below.

- (53)
- a. S: Welcome to the travel agency!
 - b. U: A flight from London to Hongkong, cheap.
 - c. S: Alright. You want to know about price.
 - d. U: Yes.
 - e. S: What month do you want to leave?
 - f. U: In March.
 - g. S: What day do you want to leave?
 - h. U: Can I travel on the first?
 - i. S: Not if you want to travel economy class . /Yes, if you travel business class.
 - j. S: So, Do you want business class ?
 - k. U: Yes
 - l. S: Okay. The price is 1555 crowns.

- (54)
- a. S: Welcome to the travel agency!
 - b. U: A flight from London to Hongkong
 - c. S: Alright. You want to know about price.
 - d. U: Yes.
 - e. S: What month do you want to leave?
 - f. U: In March.
 - g. S: What day do you want to leave?
 - h. U: Can I travel on the first
 - i. S: Yes, if you want to travel business class . / Not if you travel economy class.
 - j. S: Do you want business class ?
 - k. U: Yes.
 - l. S: The price is 1555.

In the information state update approach to dialogue, dialogue moves are modeled in two steps. First, a *selection rule* specifies the conditions under which this move should be realized, and second, an *integration rule* specifies the effects of the move on the information state. We argued that in a way CDCRs and NDCRs realize different moves. This distinction is however determined by the context and not by the response itself. Therefore we treat both kinds of CRs as answer or response moves with different preconditions for their selection and different effects of the IS. In the previous section we argued that in their forward-looking function, CRs are special kinds of responses to a question on QUD. This question concerns in the TA domain typically the availability of a particular parameter of a flight. Consequently, in the selection rule for CRs we need to specify that a CR must be selected as a response to a question on QUD about the availability of a parameter. Questions about availability trigger in our application a database search with respect to this parameter. The kind of CR that needs to be selected depends on what the database search result looks like. As already said, we want to generate a negative CR instead of plain *No* after failed DB search to suggest a parameter the user might consider relaxing to get successful search. Similarly, a positive CR can also be generated instead of plain *No* after failed database search in order to suggest an alternative parameter leading to successful search. On the other hand, we want to generate positive CR instead of plain *Yes* after successful search to indicate that search success depends on some as yet unspecified parameter, to avoid future failed search. The effect of a CR on the IS is defined in terms of its semantics, that is, both what it asserts and implicates is added to the dialogue history, as well as in terms of the succeeding move of the system.

In what follows, we concentrate on producing negative CDCRs and positive NDCRs. One reason for neglecting negative NDCRs like the one in (51e) on p. 210 which was mentioned in section 5.2.2 is that negative NDCRs seem to be rather marked by giving the impression that the speaker has some beliefs about the preferences of the other DP. Integrating the case of positive CDCR into the implementation is simple as shown in the next section. However, in order to have a version of the system which can produce all of these alternatives, we would need to implement a decision procedure for choosing the one or the other. This would require in turn additional extensions of the system including user modeling with respect to user preferences or additional processing of the database search results in order the system to be able to find the most economical way of presenting the search results as suggested at the end of section 5.2.2.

Selection of CRs

We said that we consider four cases of CRs according to their polarity and contextual givenness: positive and negative CDCR and positive and negative NDCR. We also illustrated the conditions upon which these different kinds of CRs should be produced, that is, CDCRs as a collaborative recovery from a failed DB search and NDCRs as a collaborative continuation after a successful database search

We argued that CDCRs and NDCRs can be assigned different dialogue moves. However, in the implementation we treat them uniformly as answer moves and account for the differences between them in terms of different contexts of use and different effects on the IS. These differences are accounted for in the integration rules for CRs provided in the next section.

The conditions relevant for selecting a CR are specified in the selection algorithm in figure 5.2. We discuss the respective portions of the selection algorithm below.

CDCR. When the database search with a set of user-specified parameters fails, the system attempts to collaborate by suggesting which (if any) parameter the user might relax to get a successful search instead. To find out, the system performs additional database searches with the parameters relaxed one at a time. If the relaxation of some parameter leads to a successful search the system produces a negative CDCR contingent on this parameter.

For illustration, consider (55) and the database records in figure 5.1.

Figure 5.2: CR selection algorithm

Given a set of search parameters $P = \{p_1, \dots, p_n\}$, where each $p_i = \alpha_i(v_i)$ is a proposition specifying the value v_i of an attribute α_i . Given a (possibly empty) set of database search solutions $SOL = \{S_1, \dots, S_m\}$ s.t. $\forall S_i \in SOL : P \subseteq S_i$.

If responding to a yes/no question q which specifies search parameters $Q = \{q_1, \dots, q_k\}$

if $SOL = \emptyset$ (database search failed)

then

if $\exists p_i \in P/Q$ s.t. $\exists SOL' = \{S_1, \dots, S_k\}$ s.t. $\forall S_i \in SOL' : P/p_i \subset S_i$ (some parameter identified as responsible for search failure; if this parameter is relaxed, search succeeds)

then answer q with a conditional negative response with condition p_i

else answer q with an unconditional negative response

else (database search succeeded)

if $\forall S_i \in SOL : S_i/P \neq \emptyset$ (there are other user-unspecified parameters in search result)

then

if the size of SOL is less than Max (the results are enumerable)

then answer q by enumerating SOL

else (the results cannot be enumerated)

if $\exists p_j$ s.t. $\forall S_i \in SOL : p_j \in S_i/P$ (all results share some parameter p_j)

then answer q with a conditional positive response with condition p_j

else answer q with a positive response

- (55) a. U: I want to fly from London to Hongkong on the second of March, cheap
- b. S: Sorry, there is nothing matching your request.
- c. U: Can I travel on the first?
- d. S: Not if you want economy class.

A search with the parameter set (56) fails. However, a search with the modified parameter set (57), where the *class* parameter is relaxed, succeeds, so (55d) can be generated, indicating that the negative answer depends on *class*.

(56) {*dept_city*(london), *dest_city*(hongkong), *dept_day*(second), *mont*(march), *class*(economy)}

(57) {*dept_city*(london), *dest_city*(hongkong), *dept_day*(first), *mont*(march)}

As already pointed out in section 5.2, we do not allow a CDCR contingent on parameters mentioned in the question, because of the oddity of answering a question such as (58a) with (58b).

- (58) a. S: Can I fly from Malmö to Paris on April 1st?
- b. U: Not if you want to fly on April 1st.

NDCR. Our system is collaborative also when the database search with the user-specified parameters succeeds, but there are more results than can be sensibly conveyed to the user at once. In this case, the system attempts to use a CR to indicate when the success of the search depends on any parameter as yet unspecified by the user. This helps to avoid future failed search. This part of the algorithm relies on the database search returning all the records that satisfy the search criteria and identifying some parameter(s) that all records have in common. When this is the case, a positive NDCR can be generated. (A simplified version of this we described in the previous section is the case where there is only one record satisfying the search criteria.) For illustration, consider (59).

- (59) a. U: I want to fly from London to Hongkong on the first of April, cheap.
- b. S: Sorry, there is nothing matching your request.
- c. U: Can I travel on the second?
- d. S: Yes, if you fly with British airways.

Given the database in figure 5.1, a search with the parameter set (60) returns more than one hit. They all share the parameter *airline(ba)*, so (59d) can be generated, indicating this parameter as one on which the positive answer is contingent.

(60) {*dept_city(london)*, *dest_city(hongkong)*, *mont(april)*, *dept_day(second)*}

As already mentioned, the conditions for selecting particular dialogue moves by the system are specified in the selection rules for these moves. We specify those for selecting a CDCR as a special kind of answer move, namely **selectCondResp**. The rule implements the selection algorithm in figure 5.2. The particular CR is selected on the basis of the form of the database search result. The rule only specifies the cases where negative CDCRs and positive NDCRs are selected. The selection algorithm in figure 5.2 specifies that if the search results cannot be additionally processed, that is, if no negative CDCR or positive NDCR can be selected, the system generates a non-conditional negative or positive answer. A non-conditional negative answer the system generates is plain *No* or alternatively *Sorry, there is nothing matching your request about X*. A non-conditional positive answer is a plain *Yes*.

(RULE 5.11) RULE: **selectCondResp**

CLASS: **select_move**

PRE: {
 fst(\$/PRIVATE/AGENDA, respond(A))
 in(\$/PRIVATE/BEL, B)
 not in(\$/SHARED/COM, B)
 A=exists(C)
 }

EFF: {
 forall_do(in(\$/PRIVATE/BEL, D),
 [if_then_else(D=db_entry(E, relaxable(F), alternative(G), H, I),
 push(NEXT_MOVES, answer(implies(F, not(C)))),
 if_do(D=unknown(C),
 push(NEXT_MOVES, answer(unknown(C))))
 if_then_else(D=db_entry(E, additional(.(J, K)), H, I),
 push(NEXT_MOVES, answer(implies(J, C))),
 if_do(D=db_entry(E, L, M),
 push(NEXT_MOVES, answer(possible(M))))
])
 pop(/PRIVATE/AGENDA)
 pop(NEXT_MOVES)
 }

The rule specifies that CRs are *relevant answers* since they *resolve* the question under discussion (see Larsson *et al.* (2002) on relations between questions and answers). According to the definition provided there, CRs are relevant answers since they are resolving answers, although the resolution does not occur as a direct consequence of the CR as will be argued below.

For selecting positive CDCRs, a slight modification of the rule is needed ensuring that not the relaxable but the alternative of the relaxable parameter is selected. The alternative is obtained in this case during the relaxation procedure. For selecting negative NDCRs, the contextual alternative is not obtained from the search result. In the implementation, we represent contextual alternatives the same way we treat negation, i.e. by using the atom *not*. However, an improvement would be if the system is able to interpret this negation in some cases as a contextual alternative of the negated parameter. It may be necessary to define contextual alternatives in the domain knowledge of the system such that for instance *alt(class(business))* is defined as *class(economy)* and vice versa. This will ensure that the alternative of the additional parameter found in the database will be selected as an answer. We give a separate rule for these cases below. For selecting one or the other version of the rule **selectCondResp**, we need additional preconditions that have to be specified for each of them. However, as already pointed out, we leave this for future work.

RULE: **selectCondResp**

CLASS: **select_move**

PRE: {
 fst(\$/PRIVATE/AGENDA, respond(*A*))
 in(\$/PRIVATE/BEL, *B*)
 not in(\$/SHARED/COM, *B*)
 \$DOMAIN :: resolves(*B*, *A*)
 A=exists(*C*)
 ...
EFF: {
 forall_do(in(\$/PRIVATE/BEL, *D*), [if_then_else(*D*=db_entry(*E*, relaxable(*F*), alternative(*G*), *H*, *I*),
 push(NEXT_MOVES, answer(implies(*G*, *C*))),
 if_do(*D*=unknown(*C*), push(NEXT_MOVES, answer(unknown(*C*))))
 if_then_else(*D*=db_entry(*E*, additional(.(*J*, *K*)), *H*, *I*),
 push(NEXT_MOVES, answer(implies(not(*K*), not(*C*))))
 if_do(*D*=db_entry(*E*, *L*, *M*), push(NEXT_MOVES, answer(possible(*M*))))
])
 pop(/PRIVATE/AGENDA)
 pop(NEXT_MOVES)

The so selected move is sent to the generation module and the actual CR is generated.

Integration of CRs

After the system has produced an utterance, it has to be integrated in the resulting new IS. As already mentioned, the integration rules for dialogue moves specify the effects of the particular dialogue moves on the IS. The effects of CRs on context are more complex-siri than the ones of short answers or non-conditional answers. Also, NDCRs and CDCR have different effects on the context which makes them look like different moves as was argued in section 5.2.2. The way we

Table 5.3: Effect of CRs on information state

	<i>neg CDCR</i> <i>Not if c</i>	<i>pos CDCR</i> <i>Yes if c</i>	<i>neg NDCR</i> <i>Not if c</i>	<i>pos NDCR</i> <i>Yes if c</i>
<i>IS before:</i>				
<i>QUD</i>	? <i>p</i>	? <i>p</i>	? <i>p</i>	? <i>p</i>
<i>Shared</i>	<i>c</i>	<i>c</i>		
<i>IS after:</i>				
<i>Shared</i>				
<i>Assertion</i>	$c \rightarrow \neg p,$	$c \rightarrow p,$	$c \rightarrow \neg p,$	$c \rightarrow p,$
<i>Implicature</i>	$c' \rightarrow p$	$c' \rightarrow \neg p$	$c' \rightarrow p$	$c' \rightarrow \neg p$
<i>QUD</i>	? <i>c'</i>	? <i>c</i>	? <i>c</i>	? <i>c</i>

model the various effects of a CR on the information state is shown schematically in Table 5.3.⁵ As already pointed out, the question ?*p* that a CR addresses is represented in GODIS as the question under discussion (QUD). The answer provided by a CR is added to the shared commitments. The semantics of a CR is represented as a conditional using the format *implies*(*X*,*Y*). Next to the asserted conditional, a CR implicates that the answer is opposite for the respective contextual alternative as argued in section 5.2.1. The implicature is also represented as a conditional, using the format *implicates*(*implies*(*X*,*Y*)).

In addition to answering the question ?*p*, the effect of a CR is that it puts a question on QUD corresponding to the condition. The effects of CDCRs and NDCRs on the IS differ, because they occur in different contexts: A NDCR contingent upon *c* has the effect of raising the question whether this condition *c*, which had not been previously mentioned, should hold. In contrast, asserting a CDCR contingent upon *c* cannot simply raise the question whether the condition *c* should hold, because *c* has been already determined and is part of the shared commitments. A CDCR therefore has the effect of *reraising* the question whether *c* should hold.

For reasons that we discussed in section 5.2, we decided to treat the implicit question that a CR (re)raises as a separate subsequent move by the system, namely a verification yes/no-question. After a CR, the system pushes a *raise*-action on the private agenda which leads to the realization of an *ask* move. Since we argued that the implicit question need not be answered by the hearer, we model the effect of asking a verification question as a raise action and not as a findout action (see Larsson *et al.* (2002) for the difference between raise and findout). In the case of a negative CDCR, we chose to produce the verification question with respect not to the already established parameter *c* but with respect to its contextual alternative *c'* (see (61c)). This way of presenting the information to the user seems to be more informative compared to (61d).

⁵where *c* and *c'* denote contextual alternatives.

- (61) a. U: Can I travel on the first?
 b. S: Not if you want to travel economy class .
 c. S: So, Do you want to fly business class?
 d. S': So, Do you want to fly economy class?

Alternatively, one could utter instead of (61c) something like (62) which indicates the decision process that the user has to make.

- (62) Do you still want to fly economy class?

In the case of the positive CDCRs, it is more natural to produce something like (63c) rather than (63d). We therefore produce the verification question with the condition *c* and not with its contextual alternative *c'*.

- (63) a. U: Can I travel on the first?
 b. S: Yes, if you want to travel economy class .
 c. S: So, Do you want to fly economy class?
 d. S': So, Do you want to fly business class?

After the verification question is (re)raised by the system, it is put onto the QUD. It may be reasonable to have a decision when to put the verification question and when not. However, this should be based on an extended analysis of dialogue corpora and is therefore left for future work.

The difference between CDCRs and NDCRs is also accounted for in terms of the preconditions of the integration rule, that is, by checking whether the condition *c* is part of the shared commitments, and if yes, the effect of reraising the question is modeled by using the respective positive feedback grounding feature of GODIS described in Section 2.6. The act of reraising is signaled by the system by the cue phrase *So, ...* (see (61c)).

The respective rules for negative CDCRs and positive NDCRs **integrateSysNegCDCR** and **integrateSysPosNDCR** are given below.

(RULE 5.12) RULE: **integrateSysNegCDCR**
 CLASS: **integrate**
 PRE: {
 \$/PRIVATE/NIM/FST= A
 $A/\text{FST} == \text{sys}$
 $A/\text{SND} = \text{answer}(B)$
 $B = \text{implies}(C, \text{not}(D))$
 $E = \text{db_entry}(F, \text{relaxable}(C), \text{alternative}([G \mid H]), I, J)$
 $\$/\text{SHARED}/\text{PM} = \text{set}([\text{ask}(\text{exists}(D))])$
 $\text{in}(\$/\text{SHARED}/\text{COM}, C)$
 }
 EFF: {
 $\text{pop}(/ \text{PRIVATE}/\text{NIM})$
 $\text{add}(/ \text{SHARED}/\text{LU}/\text{MOVES}, \text{answer}(B))$
 $\text{add}(/ \text{SHARED}/\text{COM}, \text{implies}(C, \text{not}(D)))$
 $\text{add}(/ \text{SHARED}/\text{COM}, \text{implicates}(\text{implies}(G, D)))$
 $\text{if_do}(\text{in}(\$/\text{SHARED}/\text{COM}, C), \text{push}(/ \text{PRIVATE}/\text{AGENDA},$
 $\text{icm:reraise}))$
 $\text{push}(/ \text{PRIVATE}/\text{AGENDA}, \text{raise}(G))$
 $\text{pop}(/ \text{SHARED}/\text{ISSUES})$
 $\text{del}(/ \text{PRIVATE}/\text{BEL}, E)$
 }

(RULE 5.13) RULE: **integrateSysPosNDCR**
 CLASS: **integrate**
 PRE: {
 \$/PRIVATE/NIM/FST= A
 $A/\text{FST} == \text{sys}$
 $A/\text{SND} = \text{answer}(B)$
 $B = \text{implies}(C, D)$
 $E = \text{db_entry}(F, G, D, H)$
 $G = \text{additional}([C \mid I])$
 $\$/\text{SHARED}/\text{PM} = \text{set}([\text{ask}(\text{exists}(H))])$
 }
 EFF: {
 $\text{pop}(/ \text{PRIVATE}/\text{NIM})$
 $\text{add}(/ \text{SHARED}/\text{LU}/\text{MOVES}, \text{answer}(B))$
 $\text{add}(/ \text{SHARED}/\text{COM}, \text{implies}(C, H))$
 $\text{add}(/ \text{SHARED}/\text{COM}, \text{implicates}(\text{implies}(\text{not } C, \text{not}(H))))$
 $\text{push}(/ \text{PRIVATE}/\text{AGENDA}, \text{raise}(C))$
 $\text{pop}(/ \text{SHARED}/\text{ISSUES})$
 $\text{clear}(/ \text{PRIVATE}/\text{BEL})$
 }

Since CRs are resolving answers, they are also relevant answers.

For positive CDCRs and negative NDCRs, the rules have to be modified such that they ensure that the responses have the respective semantic form.

(RULE 5.14) RULE: **integrateSysPosCDCR**
 CLASS: **integrate**
 PRE: {
 \$/PRIVATE/NIM/FST= A
 $A/\text{FST} == \text{sys}$
 $A/\text{SND} = \text{answer}(B)$
 $B = \text{implies}(C, \text{not}(D))$
 $E = \text{db_entry}(F, \text{relaxable}(C), \text{alternative}([G \mid H]), I, J)$
 $\$/\text{SHARED}/\text{PM} = \text{set}([\text{ask}(\text{exists}(D))])$
 $\text{in}(\$/\text{SHARED}/\text{COM}, C)$
 }
 EFF: {
 $\text{pop}(/ \text{PRIVATE}/\text{NIM})$
 $\text{add}(/ \text{SHARED}/\text{LU}/\text{MOVES}, \text{answer}(B))$
 $\text{add}(/ \text{SHARED}/\text{COM}, \text{implies}(G, D))$
 $\text{add}(/ \text{SHARED}/\text{COM}, \text{implicates}(\text{implies}(C, \text{not}(D))))$
 $\text{if_do}(\text{in}(\$/\text{SHARED}/\text{COM}, C), \text{push}(/ \text{PRIVATE}/\text{AGENDA}, \text{icm:reraise}))$
 $\text{push}(/ \text{PRIVATE}/\text{AGENDA}, \text{raise}(G))$
 $\text{pop}(/ \text{SHARED}/\text{ISSUES})$
 $\text{del}(/ \text{PRIVATE}/\text{BEL}, E)$
 }

(RULE 5.15) RULE: **integrateSysNegNDCR**
 CLASS: **integrate**
 PRE: {
 \$/PRIVATE/NIM/FST= A
 $A/\text{FST} == \text{sys}$
 $A/\text{SND} = \text{answer}(B)$
 $B = \text{implies}(C, D)$
 $E = \text{db_entry}(F, G, D, H)$
 $G = \text{additional}([C \mid I])$
 $\$/\text{SHARED}/\text{PM} = \text{set}([\text{ask}(\text{exists}(H))])$
 }
 EFF: {
 $\text{pop}(/ \text{PRIVATE}/\text{NIM})$
 $\text{add}(/ \text{SHARED}/\text{LU}/\text{MOVES}, \text{answer}(B))$
 $\text{add}(/ \text{SHARED}/\text{COM}, \text{implies}(\text{not}(C), \text{not}(H)))$
 $\text{add}(/ \text{SHARED}/\text{COM}, \text{implicates}(\text{implies}(C, H)))$
 $\text{push}(/ \text{PRIVATE}/\text{AGENDA}, \text{raise}(C))$
 $\text{pop}(/ \text{SHARED}/\text{ISSUES})$
 $\text{clear}(/ \text{PRIVATE}/\text{BEL})$
 }

One issue that needs to be considered in relation to this is whether a CDCR involves removing the previously established proposition c (downdate). As already argued, the examples (64c) and (64d) as alternative continuations of (64b) show that the previously established proposition should not be downdated as a direct result of the CR, because the proposed revision can still be either accepted or rejected by the user.

- (64) a. U: Can I fly on the second?
 b. S: Not if you want to fly economy class.
 c. U: Ok, I'll fly business class.
 d. U': What about the third?

In (64a), the user asks about economy flights from London to Hongkong on March 2nd, and the system gives a negative CDCR suggesting the parameter *class(economy)* as responsible for the failed database search. In (64c), the user accepts the alternative of flying in business class. In this case, the proposition *class(economy)* can be deleted from the shared commitments, and replaced by *class(business)* as a result of (64c). (64d), on the other hand, should be interpreted as asking about economy flights from London to Hongkong on March 3rd, indicating (in an indirect way) that the user does not want to revise the parameter *class(economy)* to *class(business)*, but rather tries the parameter *dept_day(third)* instead of *dept_day(second)*. In this case it would have been wrong to remove the proposition *class(economy)* from the shared commitments as a result of the CDCR. Therefore the established proposition *c* should be kept in the shared commitments, while the alternative *c'* proposed in the CDCR is being considered.

We do not downdate the previously established proposition as a direct consequence of the CDCR. Instead, the system waits for the user to accept or reject the system's proposal. The conditional form of the CR and the contextual presence of the antecedent of the conditional in the case of a CDCR allow the system to make inference with respect to the parameter in the question about availability of flights:

- (65) a. U: Can I fly on the first?
 b. S: Not if you fly economy. Do you want to fly business class?
 c. U: Yes, (I'll fly business).
 class(business) put on /SHARED/COM
 On /SHARED/COM is also: implicates(implies
 (class(business), dept_day(first)))
 System infers: departure_day(first), i.e. puts it on
 /SHARED/COM

Similarly with the positive CR:

- (66) a. U: Can I fly on the first?
- b. S: Yes, if you fly business. Do you want to fly business class?
- c. U: Yes /Ok, (I'll fly business)
class(business) put on /SHARED/COM
On /SHARED/COM is also: implies (class(business),
dept_day(first))
System infers: departure_day(first), i.e. puts it on
/SHARED/COM

Thus, if the user answers the system verification question positively, the user question (66a) is resolved positively.

If the user answers negatively, the negated proposition is added to /SHARED/COM, for instance, *not class(business)* after (67d).

- (67) a. U: Can I travel on the first?
- b. S: Not if you want to travel economy class .
- c. S: So, Do you want to fly business class?
- d. U: No.

Here, the rule for integrating user answers should also ensure that the system makes an inference to the effect that the user question in (67a) is answered negatively: the inference *not dept_day(first)* is added to /SHARED/COM.

Without making this inference, the system would ask the question about the departure day (*What day would you like to travel?*) again. The rule for integrating the answer of the user is therefore modified to provide for this special case. This is done in the last two effects of the answer move. The content of the answer move and the content of the implicature of a negative CDCR represent the premises, and the proposition to be added to /SHARED/COM is the conclusion of an argument derived via Modus Ponens. The rule **integrateUsrAnswer** is given below.

(RULE 5.16) RULE: **integrateUsrAnswer**
 CLASS: **integrate**

```

PRE: {
  fst(/PRIVATE/NIM, A)
  A/FST==usr
  A/SND=answer(B)
  fst(/SHARED/ISSUES, C)
  $DOMAIN :: relevant(B, C)
  not not $DOMAIN :: proposition(B) and
  not in(/SHARED/QUD, C)
  $DOMAIN :: combine(C, B, D)
  $DATABASE :: validDBparameter(D) or D=not(E)

EFF: {
  pop(/PRIVATE/NIM)
  if_then_else(in($LATEST_MOVES, answer(B)),
    ! $SCORE=F,
    ! F=0.6)
  if_then_else(F ≤ 0.7,
    push(/PRIVATE/AGENDA, icm:und*int:usr*D),
    [ add(/SHARED/COM, D)
      add(/SHARED/LU/MOVES, answer(D))
      if_do(not in(/PRIVATE/AGENDA,
        icm:acc*pos), push(/PRIVATE/AGENDA, icm:acc*pos))
      if_do(F ≤ 0.9 and B
        =yes and B
        =no, push(/PRIVATE/AGENDA, icm:und*pos:usr*D))
      ])
    if_do(in(/SHARED/COM, implicates(implies(B, G))),
      [ add(/SHARED/COM, G)
        push(/PRIVATE/AGENDA, icm:acc*pos)
        ])
    if_do(in(/SHARED/COM, implies(B, G)),
      add(/SHARED/COM, G))
  }

```

Alternatively, one may want to take on a more cautious strategy and put the parameter onto /SHARED/COM only after the user has confirmed the system's inference by a verification assertion as in (68a) where after the positive answer of the user (68b) the departure day will be set on *first* and put onto /SHARED/COM.

- (68) a. S: So you want to fly on the first.
 b. U: Yes.

As a matter of fact, a question like *Can I X* suggests by its form that the user is not committing to the parameter specified in it. Such a question should be interpreted as (69a) and not as (69b).

- (69) a. I want to know the price on the first.
b. I want to fly on the first.

If the user answers the verification question positively by saying just Yes, he may be still just exploring the possibility. This is supported by the intuition that the user may answer (68a) by saying (70).

(70) U: No, I am just checking.

In this case it might be better to ask (71) instead of (68a) (provided that we are in the price plan). Then if the user answers positively, he would be making a commitment towards flying on the first.

(71) S: So you want to know the price on the first.

These are however subtle distinctions which cannot be accounted for with the simple semantics used in GODIS but require ways of representing modality.

Another issue that needs mention here is the question about the resolvedness of CRs. CDCRs and NDCRs behave differently in this respect. A CDCR resolves a question under discussion $?p$ negatively or positively because of the inference that can be made from the conditional form of the CR and the contextually determined condition. NDCRs on the other hand, do not resolve the question $?p$ before the condition c or its alternative becomes part of the shared commitments.

After the user answers the verification question of the system, the original global plan (e.g. the price plan) is recovered and the next question on the global plan is uttered by the system.

Discussion

One further issue that we want to pursue here is related to the possibility of using inference to enable the system to make further suggestions to the user given the database search results.

If the user answers negatively to the verification question following a positive or negative CDCR in our implementation or does not answer at all, the already established parameter is not changed. In this case, instead of asking the same question for the second time as in (72l), the system could ask (72m) or suggest (72n).

- (72)
- a. S: Welcome to the travel agency !
 - b. U: A flight from London to Hongkong, cheap
 - c. S: Alright. You want to know about price.
 - d. U: Yes.
 - e. S: What month do you want to leave?
 - f. U: March
 - g. S: What day do you want to leave?
 - h. U: Can I travel on the first?
 - i. S: Not if you want to travel economy class .
 - j. S: So, Do you want business class ?
 - k. U: No / -
 - l. S: What day do you want to leave?
 - m. S': On which other day do you want to leave?
 - n. S'': You have to choose another day then.
 - o. S'': You can fly economy on the second.

Another possibility would be that the system proposes an alternative departure day as in (72o) provided that it has found such in the database. In this case, the system proposes a contextual alternative p' (which can be defined as $\neg p$ for a set of contextual alternatives) of the rejected parameter. This is also done via inference as shown in (73) and (74).

- (73) a. U: Can I travel on the first?
QUD: ?p
- b. S: Not if you want to travel economy class .
Asserts: $c \rightarrow \neg p$
Implicates: $c' \rightarrow p$
- c. S: Do you want to fly business class ?
 c' proposed
- d. U: No.
not class(business) on /SHARED/COM, i.e. $\neg c' = c$
From c and $c \rightarrow \neg p$ infer $\neg p$; if there is such a flight with
a contextual alternative p' , propose it
- e. You can fly economy on the second.
- (74) a. U: Can I travel on the first?
QUD: ?p
- b. S: Yes, if you want to travel business class .
Asserts: $c \rightarrow p$
Implicates: $c' \rightarrow \neg p$
- c. S: Do you want business class ?
 c' proposed
- d. U: No.
not class(business) on /SHARED/COM, i.e. $\neg c' = c$
From c and $c \rightarrow \neg p$ infer $\neg p$;
if there is such a flight with a contextual alternative p' ,
propose it
- e. You can fly economy on the second.

If the parameter turns out to be not relaxable, a different parameter should be tried for relaxation. The system can be cooperative also in this phase of the task-solving by suggesting another relaxable parameter provided that the search procedure has taken care of this case. This is however an issue of future work.

One problem related to the implementation of this proposal also concerns the representation of contextual alternatives. Thus in the case of the negative CDCR in (73), the system's inference is disabled because of the lack of knowledge that *not class(business)* and *class(economy)* are basically the same thing, and it cannot derive from

implicates(implies(class(economy),dept_day(first))) and *not class(business)* the answer to the user question *dept_day(first)*.

5.3.4 Interpretation of CRs

It might be also useful for a system to be able to interpret CRs uttered by the user as responses to the system's questions. Such responses may be conditional on information that the system still needs to provide. Consider (75b).

- (75) a. S: Do you want to fly business?
 b. U: Not if it is a Lufthansa flight.
 c. U': Yes, if it is a SAS flight.
 d. S: It is a Lufthansa flight.
 e. S': It is a SAS flight.

If the airline is not known yet, the system should interpret this utterance as a question *Is it a Lufthansa flight?* and respond either by performing database search or by using intermediate search results if any. Then, if it finds that it is a Lufthansa flight (75d), the system should be able to infer that the user does not want to fly business class and thus resolve the question in (75a) negatively. If it finds that it is not a Lufthansa flight (75e), the system should infer that the user may want to fly business class and resolve the question in (75a) positively.

The interpretation of CRs by the system is not implemented yet. However, the implementation can be based on work we have described on the production of CRs. In what follows, we propose a specification of the update rules for CRs as responses of the user to a question *?p* asked by the system.

For integrating negative user NDCRs like (b), we propose the following. After (75a), a question *?P* asked by the system is topmost on QUD. The semantics of the negative user NDCR is represented as usual, i.e. the assertion as *implies(C, not P)* and the implicature as *implicates(implies(not C, P))*. Both the assertion and the implicature are put onto /SHARED/COM. For NDCRs, a condition must hold that *C* is not in /SHARED/COM. As already argued, NDCRs have the effect of asking an implicit question whether the condition *C* holds. This implicit question can be interpreted as a request for searching the database with respect to the parameter *C*. This can be handled by interpreting the user CRs as a user question about availability and applying the rule **integrateUsrAskExists** which triggers the domain plan for answering user questions about availability. The plan will trigger database search with respect to *C*. The subsequent

moves of the system depend on the search result. If the system finds a database record which answers the implicit user question positively, it presents the result to the user and puts them onto /SHARED/COM. Additionally, it is also able to resolve the question $?P$ topmost on QUD by using inference: If the search finds a record containing C , then C is added to /SHARED/COM. From this and from the semantics of the CRs which is also part of /SHARED/COM, the system can infer *not* P , i.e., the question $?P$ can be resolved negatively. In the case of a failed database search, the proposition *not* C will be added to /SHARED/COM, and from this and the content of the implicature that the CR gives rise to which is also on /SHARED/COM, the system can infer P , i.e., the question $?P$ can be resolved positively. Positive NDCRs can be treated in a similar way.

For CDCRs, the condition C is already on /SHARED/COM. After (75a), a question $?P$ asked by the system is topmost on QUD. The integration of the user negative CR as answer to this question will result in adding to /SHARED/COM the inference *not* P (from C and *implies*(C , *not* P)) which resolves the system question negatively. In the case of a positive CR, the inference will be P and the question will be resolved positively. Thus, no implicit question whether C and no need to look up the database are involved in interpreting CDCRs. Although this gives the impression that CDCRs as user answers are redundant, they may have a grounding function. Exploring the function of user CDCRs is however an issue of future work.

5.4 Summary and future work

In this chapter, we described the analysis of CRs and their implementation in the GODIS system which is based on the ISU approach to dialogue. We argued that CRs are collaborative responses which help the user to determine a set of parameters of a possible journey (in the TA domain). We described in detail the production of negative CDCRs as collaborative responses after failed database search and of positive NDCRs as collaborative responses after successful search in GODIS. We also provided detailed proposals for the production of positive CDCRs and negative NDCRs as well as for the interpretation of user CRs by the system. Implementing these proposals is left for future work.

There are several further issues which can be considered for future work. One issue for future research mentioned in the previous sections was the implementation of contextual alternatives (section 5.3.3). We proposed to define contextual alternatives in the domain knowledge of the system. For instance, the contextual alternative of business class is economy class. Other contextual alternatives can be defined concerning parameters like airport (e.g., Frankfurt vs Hahn) and departure day. In the latter case, the parameter has more than one contextual alternatives, e.g. the alternative of departure day second is a set of all dates other than the second. In such cases it is more sensible to restrict these alternatives, e.g. to define the contextual alternatives of a departure day to be the day before and the day after the one specified of the user.

<i>QUD</i>	<i>CR-form</i>	<i>Polarity</i>	<i>Assertion</i>	<i>Implicature</i>
?p	Not if	-	$c \rightarrow \neg p$	$c' \rightarrow p$
	(Yes,) if	+	$c \rightarrow p$	$c' \rightarrow \neg p$
	No, only if	-	$p \rightarrow c \Leftrightarrow$ $\neg c \rightarrow \neg p$	$c' \rightarrow p?$
	(Yes,) but only if	+	$p \rightarrow c' \Leftrightarrow$ $\neg c' \rightarrow \neg p$	$c' \rightarrow p?$
	Only if	+	$p \rightarrow c' \Leftrightarrow$ $\neg c' \rightarrow \neg p$	$c' \rightarrow p?$
	Yes, unless	+	$\neg c \rightarrow p$	$c \rightarrow \neg p$
	No, unless	-	$\neg c' \rightarrow \neg p$	$c' \rightarrow p$

Other issues for future work mentioned in the previous sections were the identification of conditions allowing to choose between negative vs. positive CDCRs and negative vs. NDCRs (section 5.3.3), the question about the function of user CDCRs (section 5.3.4), the decision when to utter a verification question after a system CDCR and when not (section 5.3.3) and the extension of the database consultation procedure to cover the cases where there are more than one relaxable or additional parameters to suggest (section 5.3.2).

Another extension concerns the integration rule for CDCRs. Currently, we are implementing the effect of a CDCR on the IS as reraising a question. However, reraising the question whether a proposition should hold can be seen as opening a negotiation whether the already specified answer to that question should be preserved or revised (because it has already once been determined). It would therefore be natural to provide an account of CDCRs as proposals opening negotiation. This can be done by using the issue-based account of collaborative negotiative dialogues proposed in Chapter 4. Thus, the effect of CDCR can be represented as a set of two alternative proposals: To keep c or to revise c .

A further extension of our work is the implementation of CRs as answers to wh-questions as in (76), as well as as responses to assertions as in (24) on p. 192.

- (76) a. A: What class did you have in mind?
b. B: Business, if it costs less than 800 euro.

Another issue of future work is the integration of other forms with which CRs can be realized like *unless*, *not/only before/after*, *provided that*, *as long as*. A preliminary analysis of the semantics of some of them is shown in Table 5.4.

A further issue of future interest is the realization of CRs. We have observed that CDCRs and NDCRs differ as for what kind of prosodic patterns are appropriate in English, i.e. the NDCR

in (77) is appropriate with a neutral/unmarked pattern (just the nuclear stress realizing a default intonation pattern), whereas the CDCR in (78) is marked prosodically using a contrastive pitch accent.

- (77) a. U: A flight from Köln to Paris on Sunday.
b. S: I'm sorry, there are no flights from Köln to Paris on Sunday.
c. U: Can I fly on Monday?
d. S: Not if you want BUSINESS class.
e. S': Yes, if you want ECONOMY class.
- (78) a. U: I want a business class flight from Köln to Paris on Sunday.
b. S: I'm sorry, there are no flights from Köln to Paris on Sunday.
c. U: Can I fly on Monday?
d. S: Not if you want **business** class.
e. S': Yes, if you want **economy** class.

We believe that this different realization is due to distinct information structure (Steedman (2000)). The contingent parameter is realized as part of the Rheme (possibly Rheme-focus) in a NDCR and as part of the Theme (Theme-focus or contrastive topic) in a CDCR. More research on these correlations is needed though, as well as on the relation to Steedman's discussion of the "responsibility" of the hearer or the speaker for a particular Theme.

Chapter 6

Tutorial Dialogues

6.1 Introduction

An attempt will be made in this section to assess the possibility of developing a tutorial dialogue system using the TRINDIKIT toolkit. As part of this goal, we will consider the use of GODIS as an example of a system that was built with TRINDIKIT. In particular, we are considering the version of GODIS for action oriented dialogues (Larsson *et al.* (2000a)), and within that the IMDiS experimental implementation for instructional dialogues (Larsson and Zaenen (2000)). First, an overview of the characteristics of *tutorial dialogues* will be given in 6.2. In Section 6.3 we suggest a framework for characterising the discourse behaviour in tutorial dialogues. The assessment in 6.4 will be based on how well GODIS/IMDiS and TRINDIKIT itself can cater for the characteristics and the behaviour looked at.

6.2 Characteristics of Tutorial Dialogues

In this analysis, we are going to be looking both at the characteristics that hold for the genre of tutorial dialogues, independently of the tutoring method employed, as well as those characteristics that are dependent on the method.

We begin by giving an example (79) from the BE&E corpus (Rose *et al.* (August 2001)) to illustrate some of the points that will be discussed below. In the example, the tutor gives a few hints trying to make the student follow her reasoning (in T[2], T[4], T[6], and T[7]). Having done that, she realises that the student does not remember the lesson well, because he is so bad at interpreting her hints that she is forced to give explanations about basic concepts (T[9]). In

the end, she asks him to read the lesson again (T[11]), not wanting to just give the answers away. The overall teaching strategy followed here will be discussed in 6.2.1 and 6.2.2.

Notice that in T[2] the student asks a question. It is obvious that the tutor knows the answer to it, but she does not provide it. It is also interesting that in T[4] the tutor says “OK” which does not mean that she accepts the student’s answer as correct. Quite on the contrary, it turns out that the student is not capable of answering the question in T[2] at all. We will have a closer look at phenomena of this kind in 6.2.5.

This dialogue example is additionally interesting because of the embedded dialogue about the strategy followed, a kind of meta-dialogue move in S[8] and T[9].

- (79) S[1]: I have no idea what a sinewave is. Was this covered in the tutorial?
- T[2]: Yes, remember the wave that represented alternating current in the lesson?
- S[3]: I think i remember it being represented as a \sim on the ammeter control panel
- T[4]: OK, that’s true about the multimeter’s function dial. But do you remember
- S[5]: Nope
- T[6]: a graph of a wave in the lesson that represented alternating current?
- (20 sec later)
- T[7]: Do you remember reading about frequency and amplitude and all that?
- S[8]: I’m not sure. Is this a trick question to see if you can get me to invent a memory?
- T[9]: No, this is not a psychology experiment. :) I’m just trying to see how much you remember. A sinewave starts out at 0 and increases to the maximum amplitude then decreases past 0 in the negative direction and then returns to 0 again. Does any of this ring a bell?
- S[10]: It really doesn’t. But I think I’m following your explanation.
- T[11]: Go ahead and reread the lesson.

6.2.1 Guided Problem Solving

In tutorial dialogues, the tutor needs to guide the student through the problem at hand. The major characteristic of tutorial dialogues is the fact that they are dialogues between an expert, the tutor, and an amateur, the student. The two dialogue participants do not collaborate towards

the completion of the problem solving task in equal terms. The tutor is an expert in the domain taught. The student is supposed to have only partial knowledge of it. However, it is the dialogue participant with the least knowledge, i.e., the student, that is expected to complete the task. Due to that, the roles of the two participants are different and the expert needs to guide the amateur. That has a number of repercussions.

First of all, the tutor has to be able to follow the student's reasoning, recognise mistakes and correct them in an appropriate way. The way of correcting has to be both pedagogically and cognitively justified. The former will concern us in Section 6.2.5, where we consider the discourse level of tutorial dialogues in more detail. The latter refers to the teaching tactic that is best, regarding the possibility of the student assimilating the knowledge provided.

The tutor must be in a position to evaluate the student's answer and give positive or negative feedback in order to achieve learning, independently of any specific tutoring method. In the case of negative feedback, some correction is also necessary. Correction might take the uncontroversial form of letting the student know that they are mistaken and giving away the right answer. A more demanding way of correction is to at least give explanations or general guidance that is relevant to the student's reasoning, so that the student can follow it more easily. That means that in addition to evaluating the correctness or falsity of an answer, the tutor also has to recognise the source of mistakes. Only then is she able to give the appropriate explanation, based on what it is that the student does not understand.

To model guided problem solving adequately, mixed initiative is necessary at all levels. Having the possibility to evaluate the student's performance and provide tutoring, corrections and explanations, presupposes that the student can take both task and dialogue initiative. The student, for example, should take task initiative in order to be allowed to suggest solutions to problems and dialogue initiative to ask for guidance. However, since the tutor has to intervene whenever the student does something wrong and correct it, the task initiative cannot stay only with the student. In addition, the tutor needs to take the dialogue initiative when she wants to align the student's replies or actions with what she has expected or understood, or in order to follow the student's reasoning and give an explanation based on that etc. (Chu-Carroll and Brown (1998)).

In summary, in order for a system to simulate the characteristics of tutorial dialogues, it has to be able to guide the student. It must follow the student's reasoning from the input that the student gives. That amounts to recognising the plan for addressing the problem at hand, that the student has in mind. It also has to give feedback on the student's progress, which means that it has to recognise mistakes and the reason behind them as well, before it can provide explanations. To make all the above possible, mixed initiative on both task and dialogue level is necessary (see example 79).

6.2.2 Tutoring Methods

This section looks at the issue of teaching methods and the choice between the didactic and the socratic method. It also proposes the socratic method as the desired one for tutorial dialogues and gives an account of its characteristics. The *socratic* method is the method that aims at getting the student to do as much as possible in order to encourage active learning. Explanations are kept to the lowest possible minimum and are provided only when absolutely necessary in order to prevent frustration.

The *didactic* method is the traditional method where the teacher provides the student with long explanations. The answer is given away as soon as he has a problem understanding a concept or performing a task. The reasoning behind it is also explained extensively. After that, the tutor checks if the student has understood (Rose *et al.* (August 2001)).

There are, of course, variations of the two methods. We have only tried to capture in the above definitions their defining characteristics.

We will now concentrate on the socratic tutoring method. The reason we are choosing to do that is twofold; On the one hand, it has been shown that the socratic method is more efficient in terms of learning (Rose *et al.* (August 2001)) and is thus the required method that a tutoring system should be able to simulate. Another reason for the purposes of this exposition is that the major rival of the socratic method, the didactic method, can be seen as a special case that is evoked within the socratic method anyway. In that case, when a system can manage the socratic method, it will by definition be able to simulate the didactic one, as well.

6.2.3 Hinting

The major characteristic of the socratic method is the use of hints. A *hint* can be seen as an instrument of active learning. Hints prompt the student for self-explanations as an alternative to explanations being provided to him by the tutor (example 79 T[2], T[4], T[6] and T[7]). They can take the form of eliciting information that the student is unable to access without prompting, or information which he can access but whose relevance he is unaware of with respect to the problem at hand. Alternatively, a hint points to an inference that the student is expected to make based on knowledge available to him, which helps the general reasoning needed to deal with a problem (Hume (1995); Hume *et al.* (1996)).

The effects of self-explanation have been the subject of study of cognitive psychology experiments (Chi *et al.* (1989); Chi *et al.* (1994)). The results of those experiments show that self-explanation in the study of example-exercises refine the steps of the example and through that help the students generalise about the conditions of the example. They are, thus, able to apply

what they've learned in other cases. They also integrate the new knowledge with the knowledge they already possess. In doing that, students build a coherent body of knowledge that can be used productively.

Let us now consider the characteristics of tutorial dialogues in the socratic method and the additional issues that this brings up.

The tutor has to be able to come up with hints that closely relate to the student's answer as well as the topic at hand, instead of just following the student's reasoning passively. That in turn makes the need of having a multi-level abstraction ability clear. The tutor must provide hints at the level where they are necessary and most helpful. A hint is no use if the student already understands what is being hinted at, or if the level of abstraction is way above his reach (example 79, T[4]). From that it follows that the content of hints must use information that the student is supposed to have either from the study material or because of previous mention in the course of the lesson. Hints have to be tailored to the student and the current purpose.

Moreover, hints have to be adjusted to the overall performance of the student both for cognitive and pedagogical reasons. It is pointless to carry on hinting when a student just cannot follow any of the hints (example 79, T[11]). To mention the opposite extreme, if the tutor provides more information than what the student needs, it can be frustrating for the student. Most importantly it defeats the general purpose of hinting, which is to elicit as much information as possible from the student towards the completion of the task.

Another level at which hints have to be tailored to the student answer is the surface realisation, which is important for making the relevance of the hint obvious in the given context. That can be done, for example, by making use of discourse markers that obviate relationships between different pieces of information given (Moore (1993)), (example 79, T[4]). At the same level, a hint can be realised as a question or as a statement based on the dialogue context and what form better fits in with the preceding dialogue act.

On the whole, the use of the socratic method presupposes that the system can produce hints. It needs to reason at multiple levels according to the respective level of the student's reasoning and performance. It also must make the connection of the hints obvious at the surface level, that is, in the realisation of the hint.

6.2.4 Explanations

One can observe that tutors avoid helping out the student by sharing their knowledge, e.g., by giving direct instructions as to what has to be done. That behaviour holds for both the socratic as well as the didactic method. The difference between them is the degree to which this is followed. The former is more hint based, whereas the latter is more explanation based. If the tutor realises

that the student has insufficient knowledge on how to perform the task, she gradually reveals information to guide the student. Even then, the information is not given openhandedly and clearly, as one would normally expect in information seeking and even in other kinds of task oriented dialogues. In effect, tutorial dialogues might appear non-collaborative.

One might argue that the overall goal of tutorial dialogues is to enable the student's active thinking, or to activate the knowledge necessary to complete the task and not to provide information. Hence, even tutorial dialogues are collaborative. That, however, has no explanatory power with regard to certain phenomena common in tutorial dialogues. In the next section we look into this kind of phenomena, as any analysis of the behaviour in tutorial dialogues has to account for them.

6.2.5 Collaborative Responses in Tutorial Dialogues

In this section we take a closer look at the discourse behaviour in tutorial dialogues. Together with the teaching method employed, they constitute the pedagogical knowledge that a tutor must have and apply. However, discourse behaviour is domain independent. It is the part of the pedagogical knowledge that deals with dialogue moves and their interrelation in the dialogue rather than the content of the tutoring, which depends on the teaching method. Of course, it can always be argued that one cannot separate the two, but the distinction is useful for descriptive and implementation reasons.

It is characteristic of the tutorial dialogue genre that the tutor only gives partial answers to the student's questions. These answers take the form of hints. It is often not clear that those partial answers address the student's questions at all. However, they seem to be interpreted indeed as if they did address the questions. The tutor does not feel that she has to do anything more than that. The student does not expect anything more (example 79, T[2], T[4], T[6], T[7]). Interestingly, whenever and as soon as the dialogue switches to everyday conversation, that phenomenon is lifted and questions are answered as expected outside the genre (example 79, T[9]).

Looking at collaborative responses at a more detailed level, there are two distinct responses that the tutor gives: *acknowledgement* and *accept*. The former can be realised as "Yes", "OK" etc and indicate that the tutor has understood the propositional content of what the student said (example 79, T[4]). The latter take the form of "Good", "Very good", "You did well" and are a means of encouragement for the student. The tutor seems to be using the *acknowledgement* dialogue act in almost every turn, and in any case, with a frequency that is totally uncommon outside the genre. In different genres encouragement of that kind and with the same frequency would be very condescending and possibly annoying.

In contrast with the continuous acknowledgments from the tutor there is a striking lack of overt signals from the student that he intends to cooperate. It is common practice that the student

will just go silent after the tutor has asked him a question or has requested that he performs an action. This silence would normally provoke a request for an overt signal of the kind that is missing in tutorial dialogues. Such signals are central to collaboration in other genres, since the two dialogue participants need to be aware of each others intentions in order for collaboration to succeed.

In the next section we propose a framework for analysing the phenomena that we just mentioned.

6.3 Obligations-based Modeling of Tutorial Dialogues

In this section, we are going to characterise the discourse behaviour in terms of the *obligation* theory (Traum and Allen, 1994a; Poesio and Traum, 1998a; Matheson *et al.*, 2000) and argue for the advantages of doing that. The theory of obligations introduces the notion of discourse and social obligation as a way of analysing some of the social aspects of interactions and provides an explanation for behaviour that other theories do not predict. It is an augmentation to the *intentions* of the dialogue participants that is intended to capture the natural flow of conversation. Intentions are still necessary but they are not the only driving force behind an utterance. Treating tutorial dialogues in terms of obligations is an intuitive way of analysing and predicting some specific kinds of dialogue behaviour that do not seem to follow the rules of everyday discourse.

On the other hand, applying a non-obligation based approach to tutorial dialogues, such as Shared Plans theory (Rich and Sidner (1998)), would soon prove problematic. A prerequisite for achieving goals, according to Shared Plans, is that the dialogue participants should be as clear as they can about their beliefs and plans. In tutorial dialogues, though, the tutor avoids doing that as much as possible. She does not reveal her plan of addressing the problem at hand. In fact, she hardly answers the student's questions about the way the problem should be addressed. This behaviour is even more obvious in the socratic method where the use of hints is predominant. It seems as if the tutor does not follow the principle of co-operativity which is central to the theory of Shared Plans. Once that is taken away the theory loses the basis of its explanatory power. For example, Shared Plans do not explain why the tutor in T[2] example 79 does not just tell the student what a sine-wave is, which she definitely has the knowledge to do.

We are now going to consider how the behaviour observed in tutorial dialogues can be better analysed by use of the obligation theory.

An effect of the tutor's authoritative role in solving the task is the obligation to give explicit grounding feedback as well as positive responses to every correct answer the student gives. That explains the behaviour that we saw in Section 6.2.5 as follows.

As we have seen, the tutor hardly ever answers questions directly, contrary to the norm outside

the genre. Because of the lack of direct answers, the tutor has to let the student know that she has taken their last move into account. So, she is obliged to give explicit acknowledgments when it is not obvious from the content of the partial answer that the tutor has taken the student's last move into account.

If we keep the previous point in mind then, exactly because of the frequency with which the tutor produces those acknowledgments in the form of "Yes" and "OK", these particles are only enough for grounding at the understanding level. The propositional content of what is being acknowledged remains to be assessed after the *acknowledgement*. It can then be accepted, or rejected (e.g., via a hint). The tutor has the obligation to perform an explicit *accept* move when the decision on that level is positive for the reasons we just mentioned. Without an explicit *accept* act by the tutor following them, "Yes" and "OK" will never be interpreted as *accept*.

The lack of overt signals by the student that he intends to cooperate can also be explained and modeled in the obligation framework. One can consider it the student's obligation to address the tutor's questions and follow her directives. Indeed, questions of the type "Do you know...?" and suggestions like "Why don't you..." are interpreted as action directives. The student is obliged to respond to the propositional content of the former and act upon the latter's suggestion. Therefore, the student attempts to perform what is expected of him, even when in different genres the dialogue participant would have just given a negative answer "I don't know", or tried to negotiate over the suggestion.

This perspective also explains the fact that the tutor interprets the lack of signals correctly. In the context of obligations, the tutor knows what the student's intentions are since these are formed based on the obligations of the genre. Both tutor and student are aware of the obligations. With regard to the specific obligation of the student to answer questions and follow directives, the tutor knows that the student is obliged to do so. Therefore, she assumes that the student is going to answer her questions and follow her directives and is not concerned with the lack of signals.

Finally, it is convenient to use the obligations framework to analyse the tutor's reluctance to answer questions and give direction towards the completion of the task. The nature of tutoring makes it the tutor's obligation to not provide all the information that they can immediately, or answer questions to the best of their ability. Instead, she gives partial answers that always withhold some information. Characterising that phenomenon in terms of obligations explains why the student does not get frustrated by it. He is aware of the obligations in the genre and the reasons that these obligations hold. Therefore, the short answers that the tutor gives are seen as discharging the obligation to address student utterances by both dialogue participants.

6.4 Reconfigurability of TRINDIKIT, GODiS and IMDiS for Tutorial Dialogues

In this section we are considering the characteristics of tutorial dialogues with regard to the possibility of using software that has been developed within the Siridus project in order to cater for them. More specifically we examine TRINDIKIT through the version of GODiS that was intended for Action Oriented Dialogues (Larsson *et al.* (2000b)), and the IMDiS experimental implementation for instructional dialogues (Larsson and Zaenen (2000)).

6.4.1 Dialogue Moves and Dialogue Context

We will now look into the use of dialogue moves and dialogue context in the available software. In GODiS and IMDiS the content of moves is, in a way, interpreted based on dialogue context. When, for example, short answers are integrated, questions in *qud*, which is the structure that represents questions under discussion¹ are used. Move selection takes context into account as well, to a certain extent and in an indirect way, by consulting the different fields. For tutorial dialogues we need to represent dialogue context explicitly and encode all information by use of dialogue moves that will make it easy to manipulate that information. As we have seen, there is a lot of knowledge about dialogue moves that derives from the context. Both the tutor and the student have to keep in mind what the previous moves have been, so that they can determine the current dialogue move and also decide about the following move that they should perform. In order to support that point, let us consider the following example, which is characteristic of the genre.

The tutor must be wary of frustrating the student by asking too many things that the student cannot answer. A way of testing that is by keeping track of what moves, in terms of tutoring, the tutor has performed so far, and what was the evaluation of the responses by the student. If the tutor has already explained a step enough times and the student still does not reply satisfactorily, the tutor can use that knowledge to decide to stop explaining, seeing that it is no use and that carrying on like that would only frustrate the student. She will then choose a more appropriate way to continue the tutoring, if at all. As decisions like this are dependent on context, the interpretation and generation of moves should also be strictly dependent on the explicit representation of dialogue context.

The existing dialogue move taxonomy should also be sufficiently extended. It must be possible for the tutor to tailor her answers to the knowledge and the level of understanding of the student, as it is revealed through the tutoring session. For that, we need a move taxonomy that is more extensive and more fine-grained than the basic necessary moves provided by GODiS or even

¹For the way questions are handled in *qud* see (Larsson *et al.* (2000b)).

IMDiS at the moment. We would also need to capture the inter-relation of moves. Examples, of some moves that are necessary for tutorial dialogues include *hint*, *t-elic*, *inform* (Tsovaltzi and Matheson (2002)).

When we have a full dialogue move taxonomy, we can include it in the dialogue context representation and make use of it to dynamically select the right dialogue moves to be produced. That is required in order for the tutor to relate guidance to what has been said already. She can, then, speak to the student's answers and manipulate their utterances in a way that leads the student to the right direction. For example, sometimes it is more appropriate to realise a hint as a question and sometimes as an affirmative sentence. *T-elic* can be used in the first case and *inform* in the latter. The core move, however, would still be *hint*.

Once the system can provide tailored answers at the dialogue move levels, we can start to think about the best way to realise the moves themselves at the utterance level. For example, we can use discourse markers to make the relationships between different pieces of information obvious and provide structure. This structure helps the students understand explanations (Moore (1993)). Some discourse markers are already available in GODiS in the form of sequencing ICM (Interactive Communication Management). Enriching this gamut is required for tutorial dialogues.

Some of the above properties are either not currently provided in GODiS, or are only partly supplied. Providing the possibility to add or augment them is part of the main philosophy behind GODiS. It has an interpretation and a generation module. It is up to the user to develop those modules with the necessary requirements specified here, either by using additional tools off the shelf, or by creating them from scratch to treat their needs.

Furthermore, the idea of representing the *information state* itself by fields foresaw the need to add new fields or modify the existing ones depending on the needs of every genre. Hence, the prerequisite of dialogue context can be handled by adding a *dialogue history* field to the *information state* to keep dialogue acts that have been performed so far. This has been done before, for example, in (Bos *et al.* (1999)). We could also use the *dialogue history* to maintain a student model for the current tutorial session, that is, how well the student is doing in this session.

Let us see now, via a couple of specific examples, how moves already used by IMDiS for instructional dialogues can be reused for tutorial dialogues. *findout*, for instance, is used in IMDiS to find out if conditions hold. In tutorial dialogues it can be used for hinting. The content will be informed by the choice of the hinting strategy (whether that is decided dynamically or based on structured plans (see Section 6.4.3)). *Instruct* is used in IMDiS to tell the user of the system what to do next. In tutorial dialogues it can be used for explanations, where the tutor tells the student how to proceed. *Confirm* allows the user to confirm what actions he has performed. It can possibly be used in place of a more appropriate way of following the steps the student takes².

²See also Section 6.4.3.

Ask can model the move *information request*, necessary for tutorial dialogues as both dialogue participants request information from each other quite often. However, the integration of the move should be augmented to allow the tutor to respond by a rephrasing, an explanation or whatever is necessary at the right level, in other words an appropriate hint. IMDiS does model that to a certain extent. By having well structured plans more detailed information can be given on demand. In Section (6.4.3) we will talk about this aspect of planning in more detail.

6.4.2 Semantic Representation and Lexicon

The current state of semantics defined in the domain resources and the resource interface, which GODiS makes use of, is very simple. For example, the utterance ‘I want to go to Paris’ would be represented as *destination(paris)* in the travel domain. For tutorial dialogues that is not sufficient. The way words are used is important for the reasoning behind the utterances to be worked out. The system has to tell interrogatives from affirmatives and negatives. It also needs to deal with quantification as a common aspect that can obscure the meaning of what the student says. Modality is also often used since different possibilities of addressing goals are discussed during tutorial sessions, and, thus, needs to be deciphered by the system. In addition, the lack of reference resolution can prove detrimental. The concepts taught have to be related to each other, therefore are often referred to in different ways.

The above list is not, of course, exhaustive. It is just indicative of some semantical aspects that need to be covered for tutorial dialogues. However, there is nothing in this list that is particular to tutorial dialogues. Any kind of more complex dialogues than the ones that have been implemented in GODiS would require more refined semantics. It is, indeed, possible to define any kind of semantics desired in the domain resources. It is the implementor’s choice to have simple or more sophisticated semantics that would suit their purposes. For example, in the EDiS system that was built with TRINDiKIT, DRT was used (Traum *et al.* (1999)).

Another thing that has to be added for every GODiS application to fit the specific needs is the Lexicon. The Lexicon is independent of GODiS itself and can be anything required for each application. For toy implementations, a simple move-utterance connection can serve to test at least the things that are specific to tutorial dialogues as a genre of dialogue, and how these can be handled by TRINDiKIT.

Since both the Lexicon and the Semantics are independent of the dialogue manager they are only mentioned here for the sake of completeness of the things that an implementor must be aware of.

6.4.3 Planning

This section deals with the way the planning in the system must be adapted. Plans are domain specific in TRINDIKIT, so changing them does not affect the rest of the system. We will be concerned with the planning in IMDiS, which implements the planning that is closer to the needs of tutorial dialogue. However, plans in IMDiS are still not very flexible. They are an ordered list of goals to be achieved. Another drawback of the present status is that there is no clear distinction between dialogue and task goals. The existing discourse plans in IMDiS only instruct the user to perform actions which are in the task plan, or inquire about whether the actions have been performed.

Guiding the student

As we have already mentioned in Section 6.4.1, the implementation in IMDiS allows the system to give more detailed guidance on demand and at different levels (Larsson and Zaenen (2000)). If the user does not know how to perform a substep, he can ask the system for more detailed instructions. For tutorial dialogues that is a crucial quality. It is more than likely that the student will need more information than what has already been provided. The system can also withhold information that it has when explanation is not necessary, that is, when the user understands it. That is also a useful property. It can help avoid frustrating the student by providing information he possesses, when there is so much information that he does not possess.

The structuring just mentioned is exactly what we need for tutorial dialogues. For the purposes of tutoring, a bit more depth would have to be added to capture the level of reasoning, as opposed to just the performable steps. That is to say, we need more levels of abstraction with which to represent the plan of addressing a task. From the student performance and assumed knowledge, as well as the information already provided in the course of the current tutoring, the kind of hint to be given next will be formulated.

Multilevel plans can be used so that the tutor can pinpoint the problem behind a mistake by the student, as well. The problem might be obvious from the performed actions towards the completion of the task by the student. It might also be a problem in the reasoning. Thus, the Tutor needs to be able to follow the different levels of the student's reasoning and also guide him. The plans must represent these levels.

Manual creation of structured plans with steps and substeps is possible in TRINDIKIT. We can have a database of plans instead of dynamically generating them³. From the database the relevant plan would be down-loaded to the appropriate field, as discussed below. The substeps will be

³Even when some other form of plan recognition is possible, e.g. a theorem prover, the dynamically generated output would still have to be stored in a structured mode.

modeling the different levels mentioned above. The plan followed by the student can be accommodated each time. That is easy to be done in TRINDIKIT as it provides a way of separating procedural and declarative knowledge through modules called resources. All declarative domain knowledge can be stored there and called by update rules and algorithms as desired.

That, of course, presupposes that the system itself can reason about when more detailed guidance is required and what level is appropriate. That is not provided as such. There are ways, though, to easily achieve it. To start with, a combination of *context accommodation* and the structured plans can deliver the desired effect. The system accommodates a plan for completing the current task. The student's actions must be matched with one of the steps in that plan. Steps that are part of the plan but appear later on can be accommodated, provided there are no ordering restrictions. When the last step of the plan has been matched, the task can be considered completed. The notion of *accommodation* has already been implemented extensively using TRINDIKIT (Larsson (2002a)).

Plan recognition and partially ordered steps

In the same context, an aspect of tutorial dialogues is that students may choose different ways of dealing with the same task. To use a straight-forward example from the domain of mathematics, proving the same theorem can be done equally well by completely different proving strategies. For that reason, different plans for performing the same task can be available and the one that the student chooses can be accommodated. A technic for doing that based on the present step is available in GODIS. It looks up the step in all the plans and accommodates the plan that includes it (Larsson *et al.* (2000b)). Making use of the different plans in the database and that technic, covers to a satisfactory extent the need for freedom in task initiative. It does, of course, by no means solve the general problem of the incommensurability of plan recognition.

Another issue to consider is allowing as much freedom as the domain and the task themselves allow in terms of the order in which steps in the task plan are performed. A possibility for dealing with this exists in IMDiS as well. In the context of instructional dialogues, IMDiS models postconditions for the plan that the system has to find out whether they hold or not, in order to determine what the following instruction will be. For tutorial dialogues, the system needs to be able to disallow the performing of certain steps when these are bound by preconditions not yet satisfied. If we have that, we can achieve a partial order in the performance of steps in the plans. Every time a step is attempted, and there is a precondition associated with it, the system will check if the precondition holds. Based on the result of that check, it will allow or disallow the performing of the step. If the student attempts to carry out a step the preconditions for which are not satisfied, the system can restrict him. If the performance of steps is discussed and not attempted, the same will be applied to hinting. The system can judge whether to accept an answer by the student or not.

Planning of hints

Regarding the planning of hints, for every step that exists in the plans there can be a hinting strategy related to it that can be invoked if needed. That can be used instead of an automated way for dynamically choosing the right hint to generate each time. On the other hand, dynamic realisation of hints might prove more important than their dynamic planning, not least because it has to take into account dialogue aspects as well. IMDiS can handle reasoning about the production of dialogue moves based on context via the introduction of a *dialogue history* field, as we suggested in Section 6.4.1. However, it needs to be changed further for the realisation of hints. More issues have to be taken into account that pertain to the discourse structure, such as discourse markers, repeating words and phrases used by the student and using useful referring expressions. Not including those issues in the implementation might make the solution of hint-step association less optimal.

Both a taxonomy of hints and an algorithm are necessary even if the generation takes place via the static plan construction and hint-step association. Without those, we cannot model the hinting process at all. The taxonomy will consist of hints that are used in tutorial dialogues. These should capture the way tutors prompt students to do the right thing. This can be thought of as prompting to follow the particular plan that the tutor has in mind (or has accommodated for handling the task at hand). It is the implementor's job to define the specific form of each hint in the taxonomy for a particular domain.

In order to use the taxonomy for a dynamic generation of hints, an algorithm for choosing the right hint to be produced is also needed. This should be based on data from existing tutorial dialogues and the theory of the teaching strategy that is to be modeled. It should take into account the student model (how well or badly the student is doing), and the latest move by the student. Update rules in the resource interface of TRINDIKIT could model the algorithm.

Student answer evaluation

Another important aspect of tutorial dialogues is that we need to have a way of evaluating the content of the student's answer, i.e., whether it is correct, wrong, near miss e.t.c. (Glass (2001)). We need a sophisticated answer evaluation in order to be able to categorise the student answer and to produce relevant, useful and pedagogically legitimised feedback.

This is a complex issue on its own. There is currently not much in GODIS/ IMDiS that can be reused towards this direction. Student answer evaluation is, however, very domain specific. It cannot be modeled without strict reference to domain knowledge, at best organised in a comprehensive ontological order. Even if one was to try some shallow parsing instead of a fully fleshed evaluation, that would still have to take into account the expected correct an-

swer. Hence, it is again domain specific. For some relevant work see (Aleven *et al.* (2001); Glass (2001)).

Towards this direction, there could also be a set of expected wrong answers. If they are linked to some misconception that gives rise to them, then the right hint to produce will depend on that (Core *et al.* (2000)). Core *et al.* have done this by combining a problem solver (matches students steps to plans), dialogue context, an expectations module and a curriculum module (what the student is supposed to know) (Core *et al.* (June 2001)).

Task vs. discourse planning

The issue of separating discourse and task planning is relevant for a number of things in planning. One can immediately see that this two level approach is necessary even in order to have a fine-grained and multi-level move classification. DAMSL (Core and Allen (November 1993)) is an example of a move classification of that sort. According to DAMSL, one has to allow for being able to acknowledge at one level and reject at another. There has been very detailed work done in GODIS regarding the different levels of understanding (Larsson (2002a)). For tutorial dialogues we also need to allow for this behaviour between discourse level and task level. In other words, we must be able to show understanding at the discourse level, where we can make use of the different levels of understanding already implemented. We also need to reject an answer, even though we understand it, and indeed only if we understand it, at all discourse levels. This issue is further discussed in Section 6.4.4.

There are additional reasons for the separation of discourse and task planning. For instance, it is difficult to improve one level because all the information from the other level obscures their separate functions that makes it difficult to check what level is not doing well. Incremental parsing is difficult as well, as all input goes through the Dialogue Manager module. For example, that makes it difficult to deal with an *acknowledgement* (“OK”), separately from the rest of the sentence that contains information about the task, which is irrelevant to it (Allen *et al.* (2001)). By maintaining different dialogue and task plans, the task plan remains the same even when the dialogue plan, which models the interaction rather than the performance of the task, needs to change. For a genre that demands such complicated task planning as tutorial dialogues, these issues become very important.

In the following section we will argue the point of the separation from the perspective of manipulating the discourse information effectively, through specific suggestions of how to achieve that in IMDiS.

Making use of IMDiS

There are things in IMDiS that can be slightly altered and then reused to adjust the planning for tutorial dialogues further. There is a field *plan* in the *information state* which represents the list of dialogue acts that need to be performed. This is not a totally static plan. Actions in it can be skipped based on the task plan. That means, for example, that an instruction on how to perform a step can be skipped if the user knows how to perform it. However, actions in *plan* are either instructions to perform a step, or requests for confirmation on whether a step has been performed. For tutorial dialogues we need to proceed to a further step in order to facilitate the manipulation of task and discourse planning.

We can introduce a new field in the *information state* as it stands in IMDiS to represent the task plan separately. *plan* will now only be used for reasoning on which dialogue acts to perform. For tutorial dialogues the task plan field can hold the actions that the student is expected to perform. In other words, the structured plan that represents the different levels of abstraction with steps and substeps. The topmost step in the new task plan field will be the one expected to be performed next. Accommodation of plans or steps, in one of the ways already described in this section, will take place within this new field.

If we choose to associate each of these steps with a hint, being, if we are, unable to produce hints in a more automated fashion, we can send the hint chosen every time to the generation module. That means that we already have a place-holder for *hint* in the short term dialogue plan, i.e., in the *agenda*. *agenda* in turn, can be informed by *plan*, now responsible only for the discourse planning.

As we saw in Sections 6.2.5 and 6.3, there are a lot of pedagogical issues at the discourse level that need to be handled independently of any domain knowledge, such as acknowledging and issues of collaboration. Representing the task plan in the *information state* and using it to manipulate all the knowledge related to the domain, will allow clearer manipulation of the pedagogical knowledge, as there will be no interference of any information about the domain. Notice, also, if we represent discourse and task plans separately, there need not be any changes in the manipulation of the discourse plan, which controls the pedagogical knowledge, for the implementation of a different domain. Pedagogical knowledge is common across domains within the tutorial dialogue genre. Only the task plan will have to be adapted for different domains.

As far as following the student's plan is concerned, we already suggested in Section 6.4.1 a somewhat non-intuitive but easy way of producing this effect. That is, involving the *confirm* move and asking the student to report on what actions they have taken. We can maintain that only to the level of performable steps, or extend it to the reasoning behind those. The latter is more intuitive than the former. It is very common in tutorial dialogues that the tutor asks the student to explain their reasoning. It is a form of *aligning*. *Confirm* may cause the topmost element of the *shared.actions* field to be popped, if it matches it. The *shared.actions* field is

used in IMDiS. It is a stack that holds the actions the user has been instructed to perform by the system and have not yet been confirmed. When the topmost element in *shared.actions* is popped it will, in turn, cause the topmost element of the task plan field to be popped. Alternatively, if the student's answer is not correct and the confirmed action does not match with anything in the task plan, the topmost element, i.e., the expected step, will not be popped and a hint, an explanation, or whatever is necessary based on the state of tutoring will be produced.

6.4.4 Collaborative Responses and Planning

This section investigates ways to handle the seemingly non-collaborative behaviour that was introduced in Section 6.2.5. In GODIS the relevance of the propositional content of an utterance which is meant to resolve a question in *qud* is judged before an answer by the user can be integrated. Only answers whose propositional content is judged to be relevant to the issue raised will be integrated. After integration, the system judges if the answer resolves the issue raised. If it does resolve it, the issue in *qud* that the answer was supposed to be resolving will be popped. Otherwise the issue might be raised anew.

In tutorial dialogues any answer within the domain will be integrated irrespectively of its relevance, due to the unequal roles and knowledge the dialogue participants have. The answer should be accepted as an attempt to answer the question and integrated without any relevance conditions. After integration, the propositional content of the answer should be assessed for its degree of relevance, based on whether and to what extent the content matches a step in the plan. The first acceptance occurs at the discourse level, the second occurs at the task level. Only when the propositional content of an answer matches a step in the plan can it be accepted. The tutor will produce an explicit *accept* move to let the student know that the propositional content of the answer was correct with regard to the task plan.

Furthermore, system utterances in GODIS are integrated automatically when there is no negative feedback. This is motivated by the simplicity of the genre of instructional dialogues and the belief that the human user does not run into problems interpreting given utterances in the given simple context. It is also in accordance with the expectations in the genre, as there is no reason to assume that the user does not understand.

In tutorial dialogues, on the contrary, because of the participants' unequal roles, any optimistic grounding is not allowed on either side. The tutor cannot assume that the student understands her hints. That kind of optimistic behaviour is not legitimate since the main aim of the task is exactly making sure that understanding and correct answers are achieved. Before integration, the system would have to make sure that the student understands and if not, provide more guidance. As far as user utterances are concerned, GODIS provides both optimistic and pessimistic grounding. We can easily just run the system under pessimistic mode. In addition, since all answers are accepted before their propositional content is assessed, the student cannot assume acceptance at

the task level unless explicitly stated.

The difference between GODiS and tutorial dialogues can be seen in terms of the levels of acknowledgement that Brandle and Evens identify (Brandle and Evens (1997)). When there is an “OK” or a “yes” in GODiS, they are taken as *accept* moves that simultaneously signal that the answer is taken into account, i.e., as an *acknowledge*, by virtue of the parent/child relationship that holds between the latter and the former. The system shows understanding and accepts the answer for relevance, based on its propositional content. In tutorial dialogues, however, these markers will only be integrated as an *acknowledge* by the tutor. Then, if the answer matches a step in the plan, the tutor is under the obligation to perform an explicit *accept* to let the student know that the propositional content of the answer has been accepted. If not, a hint will be provided.

The demand to clearly distinguish between *acknowledge* at the understanding level and *accepting* the propositional content makes obvious the need for plan recognition in tutorial dialogues to take place independently of grounding at the discourse level. Grounding takes place, as long as the student’s answer is understood at the discourse level. Then, the answer is assessed based on domain knowledge about the task. We saw in Section 6.4.3 how we can use the IMDiS implementation as a starting point for the separation of task and discourse planning and we suggested additional modifications towards that aim. After the separation, the issues we just saw in this section will be easy to deal with.

6.4.5 Mixed Initiative

IMDiS has full task initiative and the dialogue initiative only changes to the user in order for them to point out that the current plan is over-detailed for their needs. This does not satisfy the need to have mixed task initiative that was argued for in Section 6.2.1. Likewise, the dialogue initiative needs to be entirely mixed, as it has been already pointed out. Chu-Carroll and Brown (1998) argue that the dialogue initiative necessarily switches to the dialogue participant that has the task initiative as well. In the case of tutorial dialogues, we have seen additional indicative reasons to allow for mixed initiative (Section 6.2.1). Before any reasonable tutoring method can be implemented, the initiative has to be adapted. We need to allow the student to take the task initiative, which currently lies only with the system.

GODiS caters for mixed dialogue initiative but needs to be modified before it can simulate mixed task initiative. More specifically, we need to modify the domain reasoner. For one, as we saw in Section 6.4.3, we have to introduce a more sophisticated plan recognition method that would include the possibility to evaluate the student’s answer. Ideally, the system should also be able to dynamically produce hints relevant to the current state in the task planning, so that it can make useful suggestions.

6.4.6 Obligation Modeling in TRINDIKIT

Regarding the analysis of obligations in Section 6.3, there are a number of consequences for implementation. Because of their causal relation to intentions, they have a bearing on interpreting dialogue moves and deciding what dialogue act must be performed. They may be used to create protocols of behaviour. Taking advantage of that in the implementation involves having a taxonomy of moves. This way the system can be adjusted to the needs of every different genre. Only the relevant rules would have to be adapted, that is the ones that manipulate the field of obligations. Currently all the information about the moves is hard coded.

A move recognition algorithm which allows for a more sophisticated approach than the simple *qud* would further enhance the performance of the system. Kreutel and Matheson (2000) have implemented a similar system that operates based on scenarios and inference rules in the context of obligations. That also allows for easily deriving a move generation algorithm based on context, i.e., previously performed dialogue acts. As we have already seen, that would be very useful for the genre of tutorial dialogues, both because of the clear obligations observed and because of the necessity to work with as much context as possible. Obligations have also been implemented in the Trindi project, the predecessor of Siridus (Bos *et al.* (1999)).

The current system can be modified to provide the above possibilities. In order to allow for that, the obligations of the genre have to be formalised. Since obligations differ between genres, that cannot really be seen as a drawback of the particular system. The realisation of obligations also have to be specified in the semantics of the system. That is again genre dependent. To give an example, we already saw how different surface level forms cannot be interpreted uniformly between genres; For example, question forms in tutorial dialogues are really directives for the student and are interpreted as directives.

Having a field for obligations in the *information state* means that they can be manipulated separately in the update rules for different genres. Only the specifications regarding obligations in a particular genre would have to be changed/adapted and not the whole system. The latter would be the case if obligations were not modeled explicitly in the system, that is, if they are hard coded. The update rules relevant to that manipulation have to be genre dependent in order to manipulate fields in a different way for different genres.

6.4.7 Miscellaneous

A couple of minor things that are already provided by GoDIS/IMDiS are worth mentioning.

When the user asks an elliptical ‘how’ question, the system interprets it as applying to the top-most element in the *shared.actions*. That is an important feature for a natural dialogue in the

context of tutoring, as the elliptical ‘how’ question form is commonly used. Moreover, all kinds of short answers, which are also particularly common, can be dealt with by GODIS. *qud* and *accommodation* is used for that purpose in order to accommodate a question in the plan to which the short answer is relevant (Larsson *et al.* (2000b)).

Another interesting feature that has been implemented is that the system saves the state before integration every time in the *tmp* field. This is used for backtracking to the previous state in cases of over-optimistic grounding. As a lot of misunderstanding can be expected in the tutorial dialogue genre, recovering a previous information state could prove necessary.

6.5 Conclusion

In this chapter we looked into the possibility of using TRINDIKIT in order to implement tutorial dialogues. We first examined the different aspects of tutorial dialogues that need to be addressed and then gave an account of how those aspects can be dealt with using TRINDIKIT. As a starting point for the evaluation of the reconfigurability of TRINDIKIT for tutorial dialogues, we looked into the implementations of GODIS and IMDiS for action oriented and instructional dialogues, respectively. We concluded that there are a number of changes that need to take place in the above implementations.

To begin with, in order to implement tutorial dialogues we need to separate the discourse and the task planning. That will facilitate better dialogue management at both levels. Full mixed initiative is also instructed by the nature of the genre both at the discourse and task level.

For the discourse planning a full classification of dialogue acts is needed for fine-grained manipulation of discourse behaviour in tutorial dialogues. A formal analysis of obligations in the genre and additional fields in the *information state* for easier manipulation of obligations, will further assist accomplishing the behaviour characteristic in tutorial dialogues.

The task planning level requires more extensive augmentation. A way of following the student’s steps and reasoning as well as evaluating students’ answers are necessary for the tutor to tailor their guidance. For production, we need a general hint taxonomy. We also need further refined specifications of the form of every category of hint for each domain. A generic algorithm for the genre stipulating conditions of the appropriate hint production in a given context is also necessary, as well.

TRINDIKIT can serve as a basis for more or less sophisticated ways to achieve these properties. For example, various degrees of change in the way planning is handled are possible. We can use a large database of manually created plans. Alternatively, we can use other resources available, like theorem provers to aid the tutor’s reasoning and planning at the task initiative level. Besides

planning, we have seen suggestions for effecting other properties of tutorial dialogues. It is up to the implementor to decide upon a particular implementation according to the exact effect they want and the time they are willing to invest.

Part II

Issues in Flexible Dialogue

Chapter 7

Flexibility and Cooperative Behaviour in Natural Command Language Dialogues

7.1 Introduction

Work package 1 of Siridus is concerned with *Dialogue Moves: extensions and specifications*. In particular, it aims at expanding the range of types of dialogue to which the Information State Update approach is applicable. One of the new types of dialogue considered is Natural Command Language dialogues (NCLD). Natural command languages and Natural Command Language dialogues have been defined and theoretically motivated in a previous deliverable within this Work Package Amores and Quesada (2000). Deliverable 1.4 is concerned with *Flexible Dialogue*, with a goal to show how the information state update approach covers a wide range of flexible dialogue phenomena.

Accordingly, this chapter discusses flexibility in NCLDs. We first introduce Natural Command Languages and Natural Command Language Dialogues. Section 3 compares NCLDs to other types of dialogue. Next, section 4 discusses how the nature of the dialogue itself, in conjunction with other dialogue modules allow the incorporation of new commands which may improve the overall performance, naturalness and flexibility of the system. Section 5 describes how dialogue flexibility may be achieved by ensuring a complex level of linguistic coverage in the system. Section 6 concentrates on cooperative and collaborative behaviour in NCLDs. Next, a typology of conflicts in NCLDs is proposed in section 7. Section 8 proposes some advanced cooperative behaviour in NCLDs. Finally, section 9 discusses whether DISC guidelines on cooperative behaviour are applicable to NCLD systems. Sample dialogues are taken from research carried out under the Siridus and D'Homme European projects.

7.2 Natural Command Language Dialogues

A Natural Command Language (NCL) is a command language expressed through the medium of natural language. 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) supported by the system, and the natural language vocabulary which may be used to express those commands. In addition, NCLs should contain expressions typical of human-like interaction. Consequently, a Natural Command Language Dialogue is a dialogue (usually between a human and a computer) using a natural command language. This conceptualization of NCLs has been applied to two sample domains under the Siridus¹ and D’Homme² Projects.

7.2.1 Siridus

In Siridus, we are building a Spanish demonstrator jointly with Telefónica I+D, as specified in Work Package 3 *Implementing a “Natural Command Language” Dialogue System*. The system consists of an Automatic Telephone Operator System (ATOS). The user should be able to naturally issue the commands which perform the following tasks Torre and others (2001):

- Call an extension by name or office
- Redial
- Transfer calls to extension or office
- Cancel transference
- Set up a conference call
- Look up an office in the company’s directory
- Look up an e-mail address in the company’s directory

7.2.2 D’Homme

In the D’Homme project DHomme (2001), the implemented system was able to perform the following functions in a home environment:

¹<http://www.ling.gu.se/projekt/siridus>

²<http://www.ling.gu.se/projekt/dhomme>

- Switch on/off a device/set of devices
- Dim/Bright a device/set of devices
- Consult the State/Location of a device/set of devices

7.3 Comparing NCLDs with other types of dialogue

Before we analyze flexibility and cooperative behaviour, it is worth pointing out some characteristics of NCLDs.

As a consequence of their own nature, NCLDs involve just two participants: the user and the machine. One of the first decisions to be made concerns whether we should model the participants' internal beliefs, 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. So, 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 their interactions were produced.

NCLDs are different from other types of dialogue such as Information Seeking and Negotiative Dialogues. In Information Seeking dialogues, one participant requests information from the other. In Negotiative Dialogues, the goal of both participants is to come to an agreement about some conflict of interests. Another aspect which differentiates NCLDs 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 have a clear static character from the point of view of the user. That is, the data 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 NCLDs is the presence of a command execution module, which is capable of dynamically modifying the contents and state of resources external to the dialogue, such as the knowledge bases associated to the domain.

7.3.1 Functional Embedding

An important aspect of NCLDs is that they usually exhibit *functional embedding* Walton and Krabbe (1995); Reed and Long (1997). Functional embeddings occur when the goal of a subdialogue shifts to another dialogue type.

As pointed out above, task identification subdialogues are possible in NCLDs. As opposed to other dialogue types in which the overall task of the dialogue is predefined (seeking information

about flights in a travel agency, etc.), the system in a NCLD does not know *a priori* which function the user desires to perform. The first task, therefore, involves identifying the speaker's intentions. Once the task has been identified, the corresponding plan may be unfolded. Task identification may be considered a sort of *deliberation* subdialogue in the sense of Walton & Krabbe Walton and Krabbe (1995). In deliberation dialogues, the participants jointly aim to reach a decision or form a plan of action. Deliberation is goal-directed, in contrast to the more abstract reasoning characteristic of negotiation. Clarification subdialogues are also common in NCLDs, when the name of the destination of a call or some other parameter required for the successful completion of a command was misunderstood or missing. If some portion of the desired action was understood, it will be used as indirect feedback to the user:

- **U(1):** Please, transfer my calls to BAD-RECOGNITION-CHUNK
(function was understood, but not the destination)
- **S(1):** Could you repeat where I should transfer your calls?

Clarifications may be viewed as negotiation stages within the overall dialogue. Another instance of negotiation in NCLDs occurs when the system proposes alternatives in cases of conflict. This will be discussed in more detail below.

Information seeking subdialogues are common in NCLDs when the user wishes to know the state of specific devices, or consult some information (e-mail, office number) stored in the system, as pointed out above.

Figure 7.1 below shows a possible cascade of functional embedding in NCLDs.

As has become apparent in this section, NCLDs exhibit much more complexity than one could originally envisage given the apparent simplicity of the task at hand.

7.4 Adding flexibility through dialogue commands

This section discusses some ways in which NCLDs may exhibit a more flexible behaviour than that expected from the basic functionality of the system. A first level of flexibility may be achieved through the combination of domain-specific and interaction-oriented dialogue moves.

As outlined in Amores and Quesada (2000), Dialogue Moves in NCLs were classified as follows:

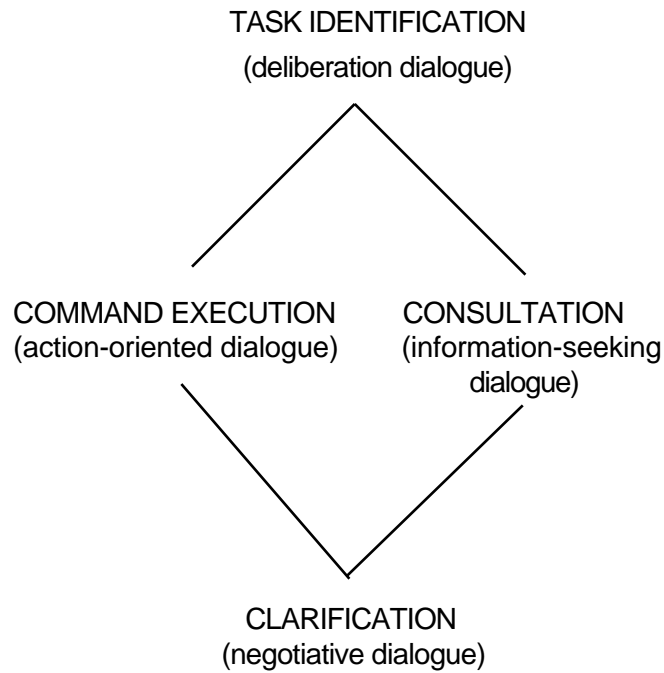


Figure 7.1: Cascade of functional embeddings in NCLDs

	<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

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, improv-

ing the overall performance, naturalness, and flexibility of the system.

A very simple example will illustrate the idea. Let us consider a telephone system that allows only the following set of commands:

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

The combined use of a dialogue manager, a dialogue history and a personnel directory (which stores the number, name and office of a person) makes some new dialogue commands possible.

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 the user, a new block of dialogue commands will appear at the dialogue level:

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

The relevant issue is that from the point of view of the user, these functions are seen as real commands, both from a functional and an operational perspective, when they are actually built up compositionally from more primitive basic functions.

7.5 Adding flexibility by expanding the linguistic coverage of the system

Another way in which more flexibility and naturalness in the dialogue system may be achieved is by expanding the linguistic coverage of the Natural Language Understanding module. That is, if this module is capable of handling complex—but natural—ways in which humans usually interact with machines, and which go beyond the expected basic functionality, the resulting system will be more natural and flexible. The following linguistic phenomena illustrate the kind of complexity which NCLDs should be able to cover. Most of the examples are taken from real accomplished interactions in the D’Homme project.

- Coordination of parameters
Switch on the kitchen light and the radio
- Issuing multiple commands
Switch the bathroom light on and the kitchen light off
- Quantification over a set of devices
Switch on all the indoor lights
- Anaphora resolution
Switch on the kitchen light and the radio
Switch them off
- Handling of exceptions
Switch on all indoor lights except for the bathroom
- Error repairs
Switch on the kitchen light ... No, the one in the living room

7.6 Cooperative behaviour in NCLDs

Let us now turn to the question of cooperative behaviour. The linguistic definition of cooperation was first proposed by Grice (1975). Reed & Long (1997) propose a notion of cooperation which is similar to that of Grice, but acts at a higher level. Gricean cooperation is speaker-centered, whereas in Reed & Long’s view, cooperation is more objectively discourse-centered. In their opinion all dialogue is inherently cooperative since any instance of true dialogue involves the participants accepting a common goal and working towards that goal within a given set of rules. This concept is also similar to the *Conversational Contract* of Fraser (1990).

Another relevant aspect regarding cooperation is that proposed by Allwood (1976). According to Allwood, two or more parties interact cooperatively to the extent that they:

1. take each other into cognitive consideration,
2. have a joint purpose,
3. take each other into ethical consideration,
4. trust each other to act in accordance with 1–3.

Of these, the most relevant requirement as regards NCLDs is 4 since, in fact, one of the agents in NCLDs is just *pretending* to act according to the principles. This idea reinforces our previous analysis of NCLDs as an inherently cooperative activity since, as long as the agents trust each other to act according to a set of principles, communication will be cooperative.

However, there is a simple fact about NCLDs which may change our perspective. Namely, that the system in NCLD applications is usually *dispensable*. Its role is to make our lives easier. If it fails to achieve that goal, the user may just ignore it and proceed to perform the desired function in the ordinary way (by pressing a sequence of digits in the telephone pad, light switches, etc.).

In this kind of scenario the system should be as cooperative as possible, trying to avoid a situation of frustration on the other participant which would indeed lead to interrupting any communication between them. So, what is usually modelled in NCLD systems is a *collaborative* behaviour in which the participants are in some sense working together to reach the desired outcome. The presence or absence of conflict will determine the level of collaboration in a dialogue. The next section describes the types of conflict which may arise in NCLDs and the expected level of collaboration which should be incorporated.

7.7 Sources of conflict in NCLDs

Conflict arises from different sources in NCLDs.

1. First, the user may just not know exactly what functionality is available, or the parameters which they require. Providing help is one way of solving this conflict. The corresponding *Dialogue Moves* (askHelp and answerHelp, both for generic and specific help) have been incorporated in our systems.

2. Second, given the dynamic nature of the domains in which we are working, one of the agents (the user) may just ignore the exact state of the world at that precise moment. Requesting information is then a collaborative activity which the system should model. Several dialogue moves have also been incorporated to take care of this kind of interaction.
3. A third type of conflict arises when one of the agents wrongly believes that a specific goal is possible, but the state of the world does not permit its accomplishment. Solving this conflict involves proposing alternatives, as the following dialogue illustrates:

- **U(1):** Call Mr Fernández
- **S(1):** This extension does not respond. Do you want to leave a message or shall I try his mobile?

A more cooperative scenario is proposed in the following example:

- **U(1):** Call Mr Fernández at extension 311
(Extension and name do not match in the database)
- **S(1a):** Mr Fernández does not have this extension; shall I put you through his current extension?
(or)
- **S(1b):** Extension 311 belongs to Mr González; do you want him instead?

In some other contexts, the alternatives proposed by the system may be based on what the user *usually* does, that is, her most frequent destinations or actions in the past. Behaving cooperatively in this context occurs, for example, when the user underspecifies a destination of a call by a last name, and that last name turns to be ambiguous in the telephone directory. In such an event, the system may consult the most common destinations (chosen in the past by the user) and propose one who matches the last name provided.

- **U(1):** Call Fernández
(after spotting a frequent Pedro Fernández in the user's profile)
- **S(1):** Do you mean Pedro Fernández?
- **U(2):** Yes, please

This form of user adaptation is related to advanced modes of cooperation to be discussed in section 8 below.

4. *Reaccommodation* may be seen as a kind of cooperative behaviour as well since it involves solving an elliptical utterance. Question/Issue Reaccommodation is defined by Larsson (2002a) as “adjustments of common ground required to understand an utterance addressing an issue which has been resolved and does not influence any other resolved issue, influences another resolved issue or concerns grounding of a previous utterance.” Two instances of reaccommodation have been implemented in our system. The first example exemplifies

a case in which there is a previous history in the dialogue, and the user issues an incomplete command.

- **U(1):** Switch on the light in the kitchen
(the light in the kitchen goes on)
- **U(2):** Now the one in the living room

A more complex type of reaccommodation occurs when, after a function has been executed, the user retracts and wishes to undo the previous action:

- **U(1):** Switch on all indoor lights
(all indoor lights go on)
- **U(2):** No, the one in the garage
(the system must undo the previous action first)

5. The last (and more productive) source of conflict arises from misunderstandings stemming from misrecognition. In our systems, some degree of cooperative behaviour has been implemented, depending on the consequences of adopting an aggressive attitude or whether explicit confirmation was seen as necessary.

- **U(1):** I want to call 12 15 collect
- **S(1):** Do you really want to call 12 50 collect?
- **U(2):** no, 12 15.
- **S(2):** OK, shall I call 12 15 collect?
- **U(3):** yes, please.

Table 7.1 below summarizes the types of conflict which may arise in NCLDs, the relevant Dialogue Moves (either from the point of view of the user or the system's) and the associated cooperative/collaborative action proposed for each case.

7.8 Advanced Cooperation

This section outlines some modes of advanced cooperation in NCLDs. Some of them may be considered cases of advanced flexibility or even robustness, which, in some sense, may be viewed as other aspects of cooperative behaviour. This functionality has not been fully implemented in our systems yet, partly because it relies on the technical limitations of the specific hardware being used.

Type of Conflict	Related Dialogue Move	Proposed Action
user ignores overall functionality	askHelp	provide general help
user ignores how to carry out a specific command	askHelp	provide specific help
user ignores current state of the world	specifyCommand (TYPE=Request)	provide requested information
desired command cannot be accomplished	informExecution askCommand (proposedAction)	propose alternatives
elliptical command	specifyCommand specifyParameter	try reaccommodation
Command and/or parameter misunderstanding	askRepeat askConfirmation	clarification subdialogue

Table 7.1: Types of conflict in NCLDs and cooperative action proposed

- **User profiling:** In the automatic telephone domain, the command *Call my wife at her mobile* is only possible if a personal directory for each person making use of the system is made available, in addition to the general one in which all the personnel is stored.

In addition, the system might allow the user to set *modes* of behaviour. For example, in the home domain, it could be possible to issue the command *set night mode*, and automatically the system would perform a series of tasks such as setting some external lights on, all indoor lights off, switching the alarm on, etc.

Similarly, in the telephone domain, the user might want to set a *non-disturb* mode, and the system should transfer all incoming calls to the answering machine or to the secretary.

- **Default reasoning** may be considered a form of cooperative behaviour, at least in the telephone scenario. In our system a default action has been specified, whereby if no other action has been recorded in the dialogue history, a *PhoneCall* action will be executed. This is useful in the (frequent) cases in which the user just picks up the telephone and utters,

– U(1): Carlos García, please

- **Proactive Behaviour** is the most extreme case of cooperativeness. In the home environment, proactive behaviour has already been proposed for cases of fire, gas or water alarm, but they rely more on the technical capabilities of the specific setting than on dialogue capabilities.

In the telephone scenario, however, a proactive cooperative behaviour stemming from the dialogue occurs when the destination is busy, and the system proposes to trigger a call-back when she is done:

- **S(1)**: Calling Juan Fernández ...
- **S(2)**: Mr. Fernández is busy. Shall I have him call you back when he is done?
- **U(1)**: Yes, please.

7.9 DISC Guidelines

Finally, let us briefly examine to what extent our systems comply with the DISC guidelines for cooperative dialogue Consortium (1999). In particular, we will focus on the *specific guidelines* proposed.

- **SG1** *Summarising feedback* is achieved through direct or indirect confirmation *Shall I transfer your calls to 0 1 2 3? or You want to transfer your calls; to which number, please?*
- **SG2** *Provide immediate feedback*: in some tasks, such as conference calls, it is better to confirm one destination party at a time, given the ambiguities that would result from the combinations of first names, last names, second last names, etc.
- **SG3** *Ensure uniformity*. In the telephone scenario this is achieved through ‘canned expressions’ in the natural language generator.
- **SG4** *State your capabilities*: is achieved through general help. A non recognized command will only turn into *I cannot do that* or *that’s beyond my current capabilities* if the expression was understood as a command, and this command is not in the set of those supported by the current system. Otherwise, misinterpretation will arise.
- **SG5** *State how to interact*: is achieved through specific help.
- **SG6** *Be aware of user inferences*. In the domains under consideration, this recommendation is applicable to those cases in which an inconsistency takes place between the model represented in the system and the beliefs of the user with regard to that state. The system detects these inconsistencies when, for example, the user tries to execute a command which makes no sense in the current situation. Or else, when the user assumes a specific behaviour for a command, different from the one actually implemented in the system. For example, if the user wrongly believes that when activating a call transfer and there is a previous transfer active, the system should automatically cancel the active one at that point, the system informs the user about this interpretation and asks her confirmation.

- **SG7** *Adapt to target group, novice/expert users*: This kind of cooperative behaviour is left for future work. In order to separate the novice from the expert user, the system might ask at the beginning if the user knows the functionality of the system, suggesting that she asks for specific help in case of a negative answer. If the user requests this initial help, the system may enter in novel user mode, which will affect the subsequent messages generated; otherwise, a more agile behaviour is followed for expert users.
- **SG8** *Cover the domain*: some degree of flexibility is achieved through advanced cooperative functions such as user profiling.
- **SG9** *Enable system repair*: is achieved through a specific `ErrorInput` strategy which generates ‘misunderstanding’ messages, and even orders these messages depending on whether it is the first, second, etc. time the input wasn’t understood. In addition, a special `DIALOGUE RECOVERY` state may be activated in case of repeated misunderstanding.
- **SG10** *Enable inconsistency clarifications*: is achieved by checking the consistency of the arguments with respect to their commands. For example, calls can only be transferred to extensions, so that if a transfer to an external number is tried, the system triggers a clarification subdialogue. In the home domain, might violate restrictions regarding security, infrastructure (maximum level of power consumption) which would require the corresponding inconsistency clarification subdialogues. Although this functionality was not addressed in D’Homme, the design of the system would certainly permit its incorporation.
- **SG11** *Enable ambiguity clarification*. Different ambiguity clarification strategies have been implemented. In the D’Homme domain, this phenomenon shows up when the user utters an expression which may affect several devices without an explicit use of universal quantification (*Switch on the light in the living-room* and there are several lights in the living-room). In Siridus, this phenomenon is especially critical since possible call destinations are identified by names or surnames. Specific strategies have been implemented in order to clarify the ambiguous expression.

7.10 Conclusion

This chapter has analyzed the complexity of NCLDs, and the different perspectives from which such dialogues may be cooperative and collaborative. In particular, several types of conflict have been identified, for which an adequate solution has been implemented in two spoken dialogue prototype systems under the D’Homme and Siridus projects. As we have shown, the systems comply with most of the DISC guidelines for cooperative dialogue.

Chapter 8

Over informative answers and clarification questions

In menu driven dialogue systems the user is guided to make a choice from a given set of alternatives. This ensures that they pick a suitable value which is at the correct level of granularity. However, without guidance, users may well supply more or less information than required. Consider the following examples where a user supplies more information than expected:

- (80) S: Where do you want to leave from?
U: Boston at 8am.

- (81) S: What service do you require?
U: Credit card payment

The first example has become a standard one to illustrate the need for one particular kind of “mixed initiative” where the user supplies more information than expected. This is handled, in VoiceXML forms by allowing more than one form value to be supplied at once, and in Godis via question accommodation. The second example is perhaps more interesting. Here the expected answer was banking, insurance or travel. However, the user has supplied more information, and will not be pleased if the reply is ignored. Moreover, once in the banking service dialogue, the user will not expect to answer a question about whether they want to make a withdrawal or payment.

The next example illustrates a user supplying less information than required.

- (82) S: Which account do you want to transfer into
U: My bank account

In this case the system expected a particular account e.g. deposit, or current. Again, although “bank account” was not a fully acceptable answer to the question, the system should not repeat the original question. The answer supplied some information, and the system needs to acknowledge this and give an indication of the granularity it requires e.g.

(83) S: Do you mean your current account or your deposit account

So far, we have looked at cases which can be handled in terms of an “is-a” hierarchy or correspond to supplying extra parameters. However, clarification questions are also required in other cases where the reply is under specified. Consider the following (constructed) example:

(84) S: Where should the new light go?
U: Upstairs
S: Which room upstairs: the front bedroom, the back bedroom or the bathroom?

Here, the rooms are part-of upstairs rather than being in an is-a relationship with upstairs.

In ontologically rich domains such as medicine, granularity issues are much harder to ignore. For example, in the medical domain, Martin Beveridge and David Milward (Ceusters *et al.* (2002)) discuss examples such as the following:

(85) S: Where does it hurt?
U: In my arm
S: Where in your arm?
U: My elbow

In this exchange, the necessary level of granularity is determined by the decision support system underlying the dialogue. For certain decisions, the system may require `location = arm`, but for others it may require `specific_location = forearm`.

How can we build this kind of flexibility into a spoken dialogue system without hard coding these kinds of interactions? In the Linguamatics home control demonstrator, the information state contains ontological knowledge. This encodes, for example, that a light is a kind of device, a kitchen is a kind of room, and a kitchen is part-of the downstairs. Dialogue interactions are not defined directly, but derived more indirectly using the ontological knowledge as well as the rest of the information state. The main need for clarification questions in the home control domain come from underspecified definite descriptions. The system handles cases such as:

(86) U: Turn on the bedroom light
S: Do you mean the back bedroom light or the front bedroom light?

These cases are treated as part of a generic strategy for treating under specified responses which tries to match up the required level of specificity with what the user provided.

Part III

Summary and conclusions

Chapter 9

Summary and conclusions

9.1 Overall summary and conclusions

This deliverable has presented various strains of work on flexible dialogue management conducted in SIRIDUS¹. The first part collected contributions dealing with flexible issue-based dialogue management in GoDiS, and the second collected work based on other approaches to dialogue management. Dialogue phenomena covered with include grounding and feedback, addressing unraised issues, flexible menu-based dialogue, negotiative dialogue, tutorial dialogue, conditional responses, clarification subdialogues, and cooperation and collaboration.

In Chapter 2 we discussed general types and features of feedback as it appears in human-human dialogue. Next, we discussed the concept of grounding from an information state update point of view, and introduced the concepts of optimistic, cautious and pessimistic grounding strategies. We then related grounding and feedback to dialogue systems, and discussed the implementation of a partial-coverage model of feedback related to grounding in GODIS2. This allows the system to produce and respond to feedback concerning issues dealing with the grounding of utterances.

In Chapter 3, we made a distinction between a local and a global QUD (referring to the latter as “open issues”, or just “issues”). The notions of question and issue accommodation were then introduced to allow the system to be more flexible in the way utterances are interpreted relative to the dialogue context. Question accommodation allows the system to understand answers addressing issues which have not yet been raised. In cases of ambiguity, where an answer matches several possible questions, clarification dialogues may be needed.

In Chapter 4, we first extended the issue-based approach to action-oriented dialogue, and im-

¹This chapter contains material from Chapter 6 of Larsson (2002a).

plemented a dialogue interface to a VCR where dialogue plans were based on an existing menu interface. We then proposed a view of negotiation as discussing several alternative solutions to an issue under negotiation. On our approach, an issue under negotiation is represented as a question, e.g. what flight the user wants. In general, this means viewing problems as issues and solutions as answers.

In Chapter 5, we described the analysis of CRs and their implementation in the GoDiS system. We argued that CRs are collaborative responses which help the user to determine a set of parameters of a possible journey (in the TA domain). We described in detail the production of negative CDCRs as collaborative responses after failed database search and of positive NDCRs as collaborative responses after successful search in GoDiS. We also provided detailed proposals for the production of positive CDCRs and negative NDCRs as well as for the interpretation of user CRs by the system. Implementing these proposals is left for future work.

In Chapter 6 we evaluated the TrindiKit toolkit and the GoDiS system by assessing the possibility of developing a tutorial dialogue system on their basis. The motivation to consider tutorial dialogues was twofold: they demand complex dialogue phenomena to be cared for and they represent a genre that has not been aimed at in any of the existing implementations within the TrindiKit/GoDiS.

There are several phenomena that justify the choice of tutorial dialogue as an evaluation measure. In particular, tutorial dialogues present a challenge for dialogue management, because they exhibit more complex patterns of mixed dialogue- and task-initiative. This is necessary, for the tutor to both allow the student to suggest ways of resolving the problem at hand (to achieve active learning), and to carefully guide the student through the task without giving answers away. We concluded that there is already sufficiently mixed dialogue initiative in GoDiS, but that task initiative needs to be augmented.

We also explored the seemingly uncooperative behaviour of a tutor, which is manifested when she avoids to reveal a solution to the task directly and does not answer student's questions to the best of her ability, thus withholding information. We suggested a way of simulating this behaviour by making use of the modularity of the structure of the information state. More specifically, we suggested implementing a field for obligations, as it has been done before in TrindiKit, to capture the particular behaviour. We also suggested to representat discourse- and domain-plans separately, in order to facilitate the implementation of the desired dialogue behavior, and the possibility of changing or augmenting it for different genres.

We also unraveled the challenge of meeting the needs of plan recognition and reasoning in tutorial dialogues, in order to provide effective feedback for the student's answers. We recommended using the idea of *accommodation* that has been implemented before in GoDiS and enables the tutor to follow the student's reasoning rather than force her own preconceptions on the student. We also saw that having achieved that, we can further explore the possibilities in GoDiS to produce suitable hints.

For the generation of hints and the tailoring of the tutor's responses to the given context and the student model, we pointed to the need to extend the dialogue move taxonomy and make use of the dialogue history, as a means of enhancing the decisions about the information to be communicated and the way it should be structured.

The overall conclusion we draw from this assessment is that the modularity of the TrindiKit architecture and the philosophy of the information state update form a sufficient environment for modeling tutorial dialogues.

Chapter 7 analyzed the complexity of NCLDs, and the different perspectives from which such dialogues may be cooperative and collaborative. In particular, several types of conflict have been identified, for which an adequate solution has been implemented in two spoken dialogue prototype systems under the D'Homme and Siridus projects. Future work will concentrate on advanced modes of cooperation suggested in the DISC guidelines. In particular, we plan to cover issues regarding user adaptation.

Finally, Chapter 8 briefly explored the use of domain ontologies for dealing with over informative answers and clarification questions.

9.2 Dialogue genres

In this section, we will use some distinctions made in previous chapters as a basis for a partial and preliminary classification of dialogues and dialogue segments along various dimensions². This classification is primarily based on issue-based dialogue management as implemented in GoDiS. While these dimensions can to some extent be used to classify dialogue systems according to the kinds of dialogues they can handle, they are not intended as a classification of human-human dialogues. Rather, they should be regarded as describing properties of dialogue segments.

As we have previously stated, we make a distinction between Inquiry-oriented and Action-oriented dialogue according to whether the dialogue concerns non-communicative actions to be performed by a DP. Usually, but not necessarily, Action-Oriented Dialogue (AOD), a.k.a. Natural Command Language Dialogue (NCLD), subsumes Inquiry-Oriented Dialogue (IOD), a.k.a. Information-Seeking Dialogue (ISD). One example of "pure" action oriented dialogue, where no questions are asked, is Wittgenstein's simple "slab" game in Wittgenstein (1953). Another example is simple voice command systems. AOD and IOD are shown with their corresponding GoDiS dialogue moves and information state components in Table 9.1.

We can also classify dialogues according to the presence or absence of general dialogue features

²This section contains material from Chapter 6 of in Larsson (2002a).

Dialogue type	Moves	IS components
IOD	ask answer	QUD ISSUES
AOD	request confirm	ACTIONS

Table 9.1: Dialogue types

such as grounding, question accommodation, and negotiation. This is done in Table 9.2. While grounding and accommodation is probably present in all human-human dialogue, negotiation may be less frequent.

Feature	Moves	IS component
Grounding	icm	TMP, grounding issues
Accommodation	accommodateX (tacit)	-
Negotiation	propose	Question●Set(Answer)

Table 9.2: Dialogue features

Finally, we can also classify activities according to various aspects of dialogue, as in Table 9.3. Note that this classification is independent of that in Table 9.2. We believe that dialogue in all these activities may be negotiative or non-negotiative, and negotiation may be argumentative or non-argumentative.

Activity type	Dialogue type	Result type	External process	Decision rights
Database search	IOD	simple: price etc. complex: itinerary	passive	user
Ticket booking	AOD	simple: book ticket	passive	user
Simple device control	AOD	simple: action	passive or active	user shared
Offline planning, incl. itinerary planning, complex device control	AOD	complex: plan	passive	shared
Online planning, incl. rescue planning (TRIPS)	AOD	complex: plan	active	shared
Explanation	IOD	complex: explanation	passive	shared
Tutorial	IOD or AOD	complex	?	tutor

Table 9.3: Activities

We will now relate the taxonomy above to the taxonomies in Dahlbäck (1997) and Allen *et al.* (2001). It should be stressed that neither Allen nor Dahlbäck have the same goals with their

classifications as we do here, and though some formulations may appear critical they are mainly intended to clarify the relation between these classifications and ours.

9.2.1 Relation to Dahlbäck's dialogue taxonomy

Dahlbäck (1997) taxonomizes dialogue according to seven criteria:

- modality: spoken or written
- kinds of agents: human or computer
- interaction: dialogue or monologue
- context: spatial, temporal
- number and type of possible/simultaneous tasks
- dialogue-task distance: long or short
- kinds of shared knowledge used: perceptual, linguistic, cultural

Our typology appears to be on a different level and is independent of many of Dahlbäck's criteria, and both cover important (but for the most part distinct) dimensions of classification. In general, the interaction between the dimensions covered by Dahlbäck and the ones covered in our typology is an interesting area for future research.

Modality is not included in our typology; however, GODIS is able to use both written and spoken language. Regarding kinds of agents, we have of course been dealing mainly with human-computer interaction; however, we have based both theory and implementation on observations of human-human dialogue.

Our dialogue typology should be regarded as primarily concerning dialogue interaction; however, a version of GoDiS (the predecessor of GODIS) has been used to produce monologue output from a domain plan specification which was also used for generating dialogue plans (see Larsson and Zaenen, 2000).

We have not included aspects of spatial and temporal context in our typology; for our theory and system we have not explored the impact of any other kind of context than (pre-stored information about) the domain (activity) and the dialogue itself.

Regarding the number and type of possible and/or simultaneous tasks, the use of the ISSUES and ACTIONS stacks allows, at least in principle, an arbitrary number of simultaneous tasks. Since the

simplest version of our theory and system can handle this, we have not used this as a dimension of classification.

The dialogue-task distance dimension is perhaps less obvious than the others. This is based on the observation that some kinds of dialogue have a structure closely corresponding to the task structure (e.g. planning or advisory dialogue), while some have a “longer distance” between these two structures (e.g. information retrieval dialogue). Dahlbäck argues that for dialogues with a short dialogue-task distance, intention-based methods for dialogue act recognition is both more useful and easier than for dialogues with a long dialogue-task distance. For the latter, surface-based act interpretation is easier and more appropriate, whereas intention-based methods are less useful and more difficult. Regarding this dimension, we have been mostly concerned with dialogues with a long dialogue-task distance, and if Dahlbäck is right an intention-based and context-dependent interpretation module will be needed when extending the issue-based approach to e.g. collaborative planning dialogue. While this may affect how dialogue moves are defined, we believe (although we cannot be sure) that the set of dialogue moves we have proposed in our taxonomy can still be maintained.

Finally, regarding the kinds of shared knowledge that are used, our taxonomy does not say much. We have not been concerned with the perceptual and cultural context, except to the extent that these are encoded in the static domain knowledge resources. The use of domain-specific lexicons can perhaps be regarded as a simplistic form of linguistic context.

9.2.2 Relation to Allen et. al.’s dialogue classification

The classification by Allen *et al.* (2001) appears to be closer in spirit to the one proposed here. Dialogues are classified according to the dialogue management technique (minimally) required by a dialogue system capable of handling the respective kinds of dialogue. Each class is further specified by example tasks, a degree of task complexity (ranging from least to most complex), and a set of dialogue phenomena handled.

- finite-state script
 - example task: long-distance calling
 - dialogue phenomena: user answers questions
- frame-based
 - example tasks: getting train timetable information
 - dialogue phenomena: user answers questions, simple clarifications by system
- sets of contexts

- example tasks: travel booking agent
- dialogue phenomena: shifts between predetermined topics
- plan-based models
 - example tasks: kitchen design consultant
 - dialogue phenomena: dynamically generated topic structures, collaborative negotiation subdialogues
- agent-based models
 - example tasks: disaster relief management
 - dialogue phenomena: different modalities (e.g. planned world and actual world)

The first thing to note about this classification is that it does not distinguish separate dimensions of classification, but rather reduce several dimensions to one; this kind of simplification and generalization does of course have its merits, but may also be confusing.

Regarding the taxonomy of technologies used in this classification, it appears that the closest corresponding dimensions in our typology is the different kinds of information states and dialogue moves used for various dialogue types, dialogue phenomena, and activities. However, the classifications are also quite different; for one thing, the finite-state-based and form-based techniques usually do not even use dialogue moves. By contrast, our classification relies on specifying dialogue moves even for very simple dialogues. We will not go into a discussion of the relative merits of these grounds of classification; suffice to say that a theory-dependent classification (which ours to some extent is) allows a greater level of detail in the classification, but its usefulness is of course dependent on the acceptance of the basic theoretical assumptions that are made.

The first two technologies listed by Allen *et al.* were discussed in Larsson *et al.* (2002), and the distinction between them are pretty much standard. However, the classification of the remaining three technologies is more problematic.

Regarding the “sets of contexts” technology, further specified as the use of several forms, it can be regarded as ambiguous between the use of several forms of the same type and the use of several forms of different types. The example task provided is “travel booking agent”, or more specifically, itinerary booking. This seems to indicate that the intended meaning of “sets of contexts” is the use of several forms of the same type (e.g. one for each leg of the itinerary). In our typology of activities in Table 9.3, this would correspond to a dialogue with a complex result. However, the use of several forms of different types seems rather to the possibility of several simultaneous tasks (e.g. asking about which channel is on while programming the VCR).

The level of plan-based technology is further specified as “interactively constructing a plan with the user” (Allen *et al.*, 2001, p.30). This specification thus says something about the result of the dialogue (a plan) and how this result is constructed (interactively). Note that this is not exactly what we referred to as the plan-based approach in Larsson *et al.* (2002); at least in principle (and perhaps also in practice; this is an empirical issue related to Dahlbäck’s concept of dialogue-task distance) it appears to be possible for a dialogue system to engage in this kind of dialogue even if the system itself does not use complex planning and plan recognition (e.g. for dialogue act recognition). Relating this to our classification of activities, it appears that the plan-based level corresponds roughly to dialogues with complex results (plans) and distributed decision rights (interactivity). As is indicated by Table 9.3, the techniques needed to handle the “plan-based” level would also be needed for e.g. explanatory and tutorial dialogue.

Finally, regarding the level of agent-based technology, further specified as possibly involving execution and monitoring of operations in a dynamically changing world, it appears that the main difference to the plan-based model is what we refer to as (pro)activeness of the external process.

To conclude, it appears from the point of view of our typology that the classification by Allen *et al.* (2001) is based on a mix of criteria, including information state components (e.g. forms) but also activity type, result type, pro-activeness of external process, decision rights, and dialogue features such as grounding (“simple clarifications”) and negotiation (“collaborative negotiation subdialogues”). The AOD/IOD distinction appears not to be included at all.

9.3 Future research areas in flexible issue-based dialogue management

In this section, we briefly mention some additional future areas for research using the issue-based approach to dialogue management. We also mention some desirable improvements to GODiS.

9.3.1 Developing the issue-based approach to grounding

Representation of utterances Starting from Ginzburg’s grounding protocols, we have formalized and implemented a basic version of issue-based grounding and feedback. However, some aspects of the current solution are not completely satisfactory, and it appears that a better solution could be obtained by explicitly representing utterances in various stages of grounding to a larger extent than in the current system.

Grounding issues Also, to increase the coverage of the theory and the abilities of the system it would be useful to represent grounding issues explicitly on several levels of grounding to a larger extent than is currently done. Some of the possible grounding issues that could be represented are the following (*S* is the speaker, *A* is the addressee, *u* is an utterance by *S*).

- Contact level
 - *S* and *A*: Do I have contact with other DP?
- Perception level
 - *S*: Did *A* perceive *u* correctly? If not, what did *A* perceive?
 - *A*: What did *S* say? Did *S* say *X*? Which of X_1, \dots, X_n did *S* say?
 - *S* and *A*: Is *u* grounded on the perception level?
- Semantic understanding level
 - *S*: Did *A* understand the literal meaning of *u*? If not, what does *A* think I meant (literally)?
 - *A*: What does *u* mean (literally)? Does *u* mean *X*? Which of X_1, \dots, X_n does *u* mean?
 - *S* and *A*: Is *u* grounded on the semantic understanding level?
- Pragmatic understanding level
 - *S*: Did *A* understand the pragmatic meaning of *u*? If not, what does *A* think I meant (pragmatically)?
 - *A*: What did *S* mean by *u* mean, given the current context? How is *u* relevant in the current context? Did *S* mean *X*? Which of X_1, \dots, X_n did *S* mean?
 - *S* and *A*: Is *u* grounded on the pragmatic understanding level?
- Reaction level
 - *S*: Will *A* accept (the content of) *u*?
 - *A*: Should I accept (the content of) *u*? If *u* is an assertion, should I accept *u* as a fact or only as a topic for discussion? If I don't accept *u*, how should I indicate this? Should I accept an altered version of *u*? Should I accept only a part of *u*?

Increased coverage Our account of grounding and ICM is so far only partial in coverage; phenomena that remain to be accounted for and/or implemented include clarification ellipsis, semantic ambiguity resolution, collaborative completions and repair, and turntaking ICM. While we have included some rudimentary sequencing ICM, further investigations of the appropriateness and usefulness of these utterances are needed; here, research on discourse markers (e.g. Schiffrin, 1987) and cue phrases (e.g. Grosz and Sidner, 1986, Polanyi and Scha, 1983, and Reichman-Adar, 1984) can be of great use. We also want to explore turntaking in asynchronous dialogue management, and how this relates to turntaking ICM.

Own Communication Management has so far not been handled at all, and this is clearly an area where the system could be improved both on the interpretation and generation side. We hope that the issue-based approach could help clarify the relation between ICM and OCM aspects of grounding-related utterances.

Methods for choosing grounding and ICM strategies We have used a very basic method for choosing grounding and ICM strategies; this could be developed to include context-related aspects of the utterance to be grounded. This also goes for the strategies for choosing between several competing interpretation hypotheses on the perception and understanding levels.

Implicit issues We believe that the modelling of implicit issues, both grounding-related and others, can be very useful for accounting for the relevance of many utterances. We therefore need to develop a general way of dealing with implicit questions and the accommodation of these. We believe that such an account should be based on dynamic generation and accommodation of implicit issues when they are needed, rather than calculating all possible implicit issues available at any stage of the dialogue.

9.3.2 Other dialogue and activity types

Of course, an obvious continuation of the work presented here is to continue extending the coverage of issue-based dialogue management to other kinds of dialogues. In this section we will discuss some possibilities.

Pro-active devices To handle dialogue with pro-active devices, it is not sufficient to model the device only as a resource, since the latter are by definition passive. What is needed is a module which is connected to the active device; we can call such a module an *action manager*. Dialogue with pro-active devices will also require asynchronous dialogue processing, at least on some level. The simplest solution is to check for messages from the device when the system has the

turn. To handle this, it is probably sufficient to have the whole system except the action manager running as a single process. However, it may also be necessary for the system to interrupt the user (or indeed itself) in mid-turn, to give a report on the state of some action or plan being executed. This is likely to require a more advanced asynchronous setup, where several processes are needed.

Complex results We have so far only been dealing with dialogues where the “result” (answer, action) is not very complex. In e.g. itinerary information dialogue, the result may be a more complex structure. In collaborative planning dialogue the result is a potentially complex plan with several actions related in various ways. Similarly, explanatory dialogue may require representation of complex explanations or proofs. To deal with dialogue with complex results, we need to be able to represent these complex structures, and perhaps also to incrementally construct them by successive additions and modifications. However, we believe that the essential features of inquiry-oriented, action-oriented, and negotiative dialogue are the same regardless of whether the results are complex or simple.

Argumentation To handle argumentation, which is most likely to appear in negotiative dialogue, we hope to be able to exploit previous work in this area, e.g. Mann and Thompson (1983) and Asher and Lascarides (1998), and relate it to the issue-based approach.

Use of obligations Traum and Allen (1994b) propose *obligations* as a central social attitude driving dialogue. For example, if DP *A* asks a question *Q* to DP *B*, *B* will have an obligation to address *Q*; typically, this obligation will then give rise to an intention to address *Q*. In GODIS, we instead add *Q* to QUD (global and local), and if the system can answer a question on QUD it will do so. It has been noted that the job done by obligations and QUD overlap to a large extent (Kreutel and Matheson, 1999), and in many kinds of dialogue the choice between QUD and obligations will not result in any differences in behaviour. However, there are also differences between QUD and obligations.

For one thing, QUD does not represent who raised the question, or who should respond to it. An interesting question then becomes: given that we include a global QUD in our information state, when does it become necessary to also include obligations? It appears that one type of dialogue where QUD on its own is insufficient is tutorial dialogue, where the tutor asks the student a so-called “exam question” to which the tutor already knows the correct answer. Given the strategy used by GODIS, the system would raise the question and then immediately answer it, which is probably not a very good teaching strategy. However, in many other kinds of dialogues it appears that the strategy of answering a question regardless of who answered it is a useful strategy. For example, if a DP *A* (a human or perhaps a robot equipped with vision) asks another DP *B* where some object is located and then finds the object, it appears more felicitous for *A* to answer the

question (“Ah, there it is!”) than to wait for *B* to do so. More importantly, we have in the preceding chapters demonstrated several uses of a global QUD (modelling grounding issues, handling issue accommodation, representing issues under negotiation) which it may or may not be possible to handle using obligations. For these reasons, we are interested in further exploring the similarities, differences, and interaction between QUD and obligations, and possibly extend the issue-based dialogue model by adding obligations, at least for modelling some complex kinds of dialogue.

General planning A similar case applies to generalized planning. We have so far only used pre-scripted dialogue plans which are used in a flexible way by the dialogue manager to enable some degree of rudimentary replanning, but it can be expected that for sufficiently complex dialogues the number of dialogue plans that are needed will become so large that pre-scripting is no longer feasible. At this point, dynamic planning will be needed. We are interested in finding out for which kinds of dialogues dynamic general-purpose planning is needed, and in integrating dynamic planning in the issue-based approach to dialogue management.

9.3.3 Semantics

The semantics currently used in GODIS is obviously very simple, and integrating the issue-based approach to dialogue management with a more powerful semantics is likely to improve both the theoretical depth of analysis, especially regarding the semantics of questions, and the performance of the system. A relevant issue in this context is the connection between dialogue features and requirements on the semantic representation used - when does more complex semantics become necessary?

Specifically, we would like to explore and implement semantics using dependent record types (Cooper, 1998), and integrate this with issue-based dialogue management. One reason for this is that dependent record types appears to provide a computationally sound framework for implementing ideas from situation semantics; the latter has been used by Ginzburg in formulating the semantics of questions on which the issue-based approach to dialogue management is based.

It should be noted that the system itself is independent of which semantics is used; this is rather a feature of the domain-specific resources and (to some extent) the resource interfaces. Adding a more powerful semantics will therefore not require any significant modifications of update rules etc.

9.3.4 General inference

While update rules can be regarded as specialized (forward-chaining) inference rules, we have so far not dealt with general inference and backward-chaining inference. Inference could be useful even in database search dialogue to reason about the best way of dealing with search results in the form of conditionals (see Chapter 5). One idea here is to introduce a private issue-structure representing questions that the system is interested in resolving; this could be regarded as modelling the “wonder” attitude. A *findout*(*Q*) action on the agenda could then result in *Q* being added to a field /PRIVATE/WONDER, which can either lead to a database search, backward-chaining reasoning from available information, or asking the user for an answer. As an example of backward-chaining reasoning using the “wonder” attitude, if the system believes $p \rightarrow r$ and wonders about $?r$, a rule could add $?p$ to the WONDER field; this rule would then implement backward-chaining modus ponens.

9.3.5 Natural language input and output

Parsing and generation In GODIS we have so far concentrated on dialogue management and used very rudimentary modules for interpretation and generation of natural language. We need to explore the possible use of robust parsing techniques (see e.g. Milward, 2000) and “real” grammars. It would also be useful to be able to automatically generate speech recognition grammars (which are usually finite-state) from the parsing grammar. Using the same grammar for parsing and generation would further decrease the amount of work needed for porting the system to a new domain or language.

Dialogue move interpretation Since we have been dealing with simple dialogue in toy domains, we have so far been able to get away with doing dialogue move interpretation independently of the dynamic context. Instead, context dependent interpretation is performed by the dialogue move engine as a subtask of integrating moves. While we believe that this is a good strategy to use as long as it works well, it may eventually become necessary to be able to recognize indirect speech acts (in our case, indirect dialogue moves), which probably requires some form of context-dependent intention recognition to decide what move has been performed.

Focus intonation One area of research that we have not mentioned so far, but where the issue-based approach shows great promise, is the generation and interpretation of focus intonation with respect to the information state. Some work was done on this in the TRINDI project (Engdahl *et al.*, 1999), and is currently being developed further in the followup SIRIDUS project.

Speech recognition for flexible dialogue One problem for any dialogue system allowing for user initiative and flexibility is that a larger speech recognition lexicon is needed, which negatively affects the quality of speech recognition. We want to explore the use of the information state, and especially QUD, for improving recognition, e.g. by running a “global” recognizer listening for anything that the system can understand, and a “local” recognizer, listening e.g. for answers to questions on QUD, in parallel.

9.3.6 Applications and evaluation

To properly test the issue-based approach to dialogue management, we believe it is necessary to build full-scale prototype applications and evaluate these based on interactions with naive users. One possible such application is VCR control; another is local travel information.

Although we have not said much about it here, we have previously explored various ways of acquiring dialogue plans appropriate for a given domain or application. Among the options we have used is dialogue distillation (see Larsson *et al.*, 2000c), conversion of domain plans to dialogue plans (see Larsson, 2000), and conversion of menu interfaces to dialogue plans.

A further option we want to explore is the use of VoiceXML (McGlashan *et al.*, 2001) dialogue specifications as a basis for dialogue plans. We hope to be able to automatically or semi-automatically convert VoiceXML scripts into complete domain and lexicon specifications for GODIS, which we hope would allow the use of general dialogue mechanisms (e.g. grounding, accommodation, negotiation) to enable flexible dialogue given fairly simple VoiceXML scripts. This would decrease the amount of work on the part of the dialogue designer, and thus enable rapid prototyping.

Bibliography

Abella, A.; Brown, M.K.; and Buntschuh, B. 1996. Development principles for dialogue-based interfaces. In *Proceedings of ECAI'96 Workshop on Dialogue Processing in Spoken Language Systems*. 1–7.

Aleven, Vicent; Popescu, Ocav; and Koendinger, Kenneth R. 2001. A tutorial dialogue system with knowledge-based understanding and classification of student explanations. In *Working Notes of 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*., Seattle, USA.

Alexandersson, Jan and Becker, Tilman 2000. Overlay as the basic operation for discourse processing in a multimodal dialogue system. In *Proceedings of the IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*. 8–14.

Alexandersson, Jan and al., et 1998. Dialogue acts in Verbmobil-2. Technical report, Verbmobil-Report.

Allen, James and Core, Mark 1997. Draft of damsl: Dialogue Act Markup in Several Layers. <http://www.cs.rochester.edu/research/cisd/resources/damsl>.

Allen, James F.; Byron, Donna K.; Dzikovska, Myroslava; Ferguson, George; Galescu, Lucian; and Stent, Amanda 2001. Toward conversational human-computer interaction. *AI Magazine* 22(4):27–37.

Allwood, Jens; Nivre, Joakim; and Ahlsen, Elisabeth 1992. On the semantics and pragmatics of linguistic feedback. *Journal of Semantics* 9:1–26.

Allwood, Jens 1976. *Linguistic Communication as Action and Cooperation*. Ph.D. Dissertation, Göteborg University, Department of Linguistics.

Allwood, Jens 1995. An activity based approach to pragmatics. Technical Report (GPTL) 75, Gothenburg Papers in Theoretical Linguistics, University of Göteborg.

AMEX, 1989. SRI's Amex Travel Agent Data. <http://www.ai.sri.com/commu-nic/amex/amex.html>.

- Amores, J. G. and Quesada, J. F. 2000. Dialogue moves in natural command languages. Deliverable 1.1, Siridus Project.
- Asher, N. and Lascarides, A. 1998. The semantics and pragmatics of presupposition. *Journal of Semantics* 15(3):239–299.
- Aust, H.; Oerder, M.; Seide, F.; and Steinbiss, V. 1994. Experience with the Philips automatic train table information system. In *Proc. of the 2nd Workshop on Interactive Voice Technology for Telecommunications Applications (IVTTA)*, Kyoto, Japan. 67–72.
- Barwise, J. and Perry, J. 1983. *Situations and Attitudes*. The MIT Press.
- Berman, Alexander 2001. Asynchronous feedback and turn-taking. ms.
- Bohlin, Peter; Bos, Johan; Larsson, Staffan; Lewin, Ian; Matheson, Colin; and Milward, David 1999. Survey of existing interactive systems. Technical Report Deliverable D1.3, Trindi.
- Bos, Johan; Bohlin, Peter; Larsson, Staffan; Lewin, Ian; and Matheson, Colin 1999. Dialogue dynamics in restricted dialogue systems. Technical Report Deliverable D3.2, Trindi.
- Boye, J.; Larsson, S.; Lewin, I; Matheson, C.; Thomas, J.; and Bos, J. 2001. Standards in home automation and language processing. Technical Report Deliverable D1.1, D’Homme.
- Brandle, Stefan and Evens, Martha 1997. Acknowledgements in tutorial dialogues. Technical report, AAAI CF-97-01, p.p.13-17. AAAI Press.
- Bäuerle, Rainer; Reyle, Uwe; and Zimmermann, Thomas Ede, editors 2002. *Presuppositions and Discourse*. Current Research in the Semantics/Pragmatics Interface. Amsterdam (Elsevier).
- Ceusters, Werner; Beveridge, Martin; Milward, David; and Falavigna, Daniele 2002. Specification for semantic dictionary integration. Technical report, HOMEY Deliverable D9.
- Chi, Michelene T. H.; Lewis, Matthew W.; Riemann, Peter; and Glaser, Robert 1989. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science* 13:145–182.
- Chi, Michelene T. H.; Nicholas, de Leeuw; Mei-Hung, Chiu; and Christian, Lavancher 1994. Eliciting self-explanation improves understanding. *Cognitive Science* 18:439–477.
- Chu-Carroll, Jennifer and Brown, Michael K. 1997. An evidential model for tracking initiative in collaborative dialogue interactions. *Meeting of the Association for Computational Linguistics* 262–270.
- Chu-Carroll, Jennifer and Brown, Michael K. 1998. An evidential model for tracking initiative in collaborative dialogue interactions. *User Modelling and User-Adapted interaction, special issue on Computational Models of Mixed Initiative Interactions* (3+4)(8):215–253.

- Chu-Carroll, Jennifer and Carberry, Sandra 1994. A plan-based model for response generation in collaborative task-oriented dialogues. In *Proceedings of AAAI-94*. 799–805.
- Chu-Carroll, Jennifer 2000. Mimic: An adaptive mixed initiative spoken dialogue system for information queries. In *Proceedings of the 6th Conference on Applied Natural Language Processing*. 97–104.
- Clark, H. H. and Schaefer, E. F. 1989a. Contributing to discourse. *Cognitive Science* 13:259 – 94.
- Clark, Herbert H. and Schaefer, Edward F. 1989b. Contributing to discourse. *Cognitive Science* 13:259–294. Also appears as Chapter 5 in Clark (1992).
- Clark, Herbert H. 1992. *Arenas of Language Use*. University of Chicago Press.
- Clark, H. H. 1996. *Using Language*. Cambridge University Press, Cambridge.
- Cohen, P. 1981. The need for referent identification as a planned action. In *Proceedings of the 7th International Joint Conference of Artificial Intelligence, Toronto*. 31–36.
- Consortium, The DISC 1999. Disc dialogue engineering model. Technical report, DISC, <http://www.disc2.dk/slds/>.
- Cooper, Robin and Ginzburg, Jonathan 2001. Resolving ellipsis in clarification. In *Proceedings of the 39th meeting of the Association for Computational Linguistics, Toulouse*. 236–243.
- Cooper, Robin and Larsson, Staffan 2002. Accommodation and reaccommodation in dialogue. In Bäuerle et al. 2002.
- Cooper, Robin; Engdahl, Elisabet; Larsson, Staffan; and Ericsson, Stina 2000. Accommodating questions and the nature of qud. In Poesio and Traum 2000. 57–61.
- Cooper, Robin; Ericsson, Stina; Larsson, Staffan; and Lewin, Ian 2001. An information state update approach to collaborative negotiation. In Kühnlein, Peter; Rieser, Hannes; and Zeevat, Henk, editors 2001, *BI-DIALOG 2001—Proceedings of the 5th Workshop on Formal Semantics and Pragmatics of Dialogue*, <http://www.uni-bielefeld.de/BIDIALOG>. ZiF, Univ. Bielefeld. 270–9.
- Cooper, R. 1998. Information states, attitudes and dependent record types. In *Proceedings of ITALLC-98*.
- Core, Mark G. and Allen, James F. 1993. Coding dialogues with damsl annotation scheme. In *AAAI Fall Symposium on Communicative Action in Humans and Machines*, Boston, MA. 28–35.
- Core, Mark G.; Moore, Johanna; and Zinn, Claus 2000. Supporting constructive learning with a feedback planner. Technical report, Human Communication Research Center, University of Edinburgh, 445 Burgess Drive, Menlo Park CA 94025.

- Core, Mark G.; Moore, Johanna; Zinn, Claus; and Wiemer-Hastings, Peter 2001. Modelling human teaching tactics in a computer tutor. In *Proceedings of the NAACL-2001 Workshop on Adaptation in Dialogue Systems*, Pittsburgh, PA.
- Dahlbäck, Nils 1997. Towards a dialogue taxonomy. In Maier, Elisabeth; Mast, Marion; and LuperFoy, Susann, editors 1997, *Dialogue Processing in Spoken Language Systems*, number 1236 in Springer Verlag Series LNAI-Lecture Notes in Artificial Intelligence. Springer Verlag.
- DHomme, 2001. <http://www.ling.gu.se/projekt/dhomme/>.
- Di Eugenio, B.; Jordan, P.W.; Thomason, R.H.; and Moore, J.D. 1998. An empirical investigation of proposals in collaborative dialogues. In *Proceedings of ACL-COLING 98: 36th Annual Meeting of the Association of Computational Linguistics and 17th International Conference on Computational Linguistics*. 325–329.
- Engdahl, Elisabet; Larsson, Staffan; and Bos, Johan 1999. Focus-ground articulation and parallelism in a dynamic model of dialogue. Technical Report Deliverable D4.1, Trindi.
- Fraser, Bruce 1990. Perspectives on politeness. *Journal of Pragmatics* 14(2):219–236.
- Gärdenfors, P. 1988. *Knowledge in Flux: Modeling the Dynamic of Epistemic States*. The MIT Press, Cambridge, MA.
- Ginzburg, J. 1997. Structural mismatch in dialogue. In Jaeger, G. and Benz, A, editors 1997, *Proceedings of MunDial 97*, Technical Report 97-106. Universitaet Muenchen Centrum fuer Informations- und Sprachverarbeitung, Muenchen. 59–80.
- Ginzburg, J. forth. Questions and the semantics of dialogue. Forthcoming book, partly available from <http://www.dcs.kcl.ac.uk/staff/ginzburg/papers.html>.
- Glass, Michael 2001. Processing language input in the circsim-tutor intelligent tutoring system. In al., J. D. Moore et., editor 2001, *Artificial Intelligence in Education*. IOS Press.
- Goffman, E. 1976. Replies and responses. *Language in Society* 5:257–313.
- Green, Nancy and Carberry, Sandra 1999. A Computational Mechanism for Initiative in Answer Generation. *User Modeling and User-Adapted Interaction* 9(1/2):93–132.
- Grice, Herbert Paul 1975. Logic and conversation. In Cole, P. and Morgan, J. L., editors 1975, *Speech Acts*, volume 3 of *Syntax and Semantics*. Seminar Press, New York. 41–58.
- Grosz, B. J. and Sidner, C. L. 1986. Attention, intention, and the structure of discourse. *Computational Linguistics* 12(3):175–204.
- Grosz, B. J. and Sidner, C. L. 1987. Plans for discourse. In *Symposium on Intentions and Plans in Communication and Discourse*.

- Hochberg, Judith; Kambhatla, Nanda; and Roukos, Salim 2002. A flexible framework for developing mixed-initiative dialogue systems. In *Proceedings of the Third SIGdial Workshop on Discourse and Dialogue*, Philadelphia, PA, USA. University of Pennsylvania.
- Hume, Gregory D.; Joel, Michael A.; Allen, Rovick A.; and Evens, Martha W. 1996. Journal of the learning sciences. *Hinting as a Tactic in One-On-One Tutoring* 5(1):23–47.
- Hume, Gregory D. 1995. *Using Student Modelling to Determine When and How to Hint in an Intelligent Tutoring System*. Ph.D. Dissertation, Illinois Institute of Technology. 150 pp.
- Jennings, N. and Lesperance, Y, editors 2000. *Proceedings of the 6th International Workshop on Agent Theories, Architectures, and Languages (ATAL'1999)*, Springer Lecture Notes in AI 1757. Springer Verlag, Berlin.
- Kaplan, D. 1979. Dthat. In Cole, P., editor 1979, *Syntax and Semantics v. 9, Pragmatics*. Academic Press, New York. 221–243.
- Kreutel, Jorn and Matheson, Colin 1999. Modelling questions and assertions in dialogue using obligations. In Van Kuppevelt et al. 1999.
- Kreutel, Joern and Matheson, Colin 2000. Incremental information state updates in an obligation-driven dialogue model. *Language and Computation* 0(0):1–32. To appear.
- Larsson, Staffan and Zaenen, Annie 2000. Document transformations and information states. In *Proceeding of the 1st SigDial Workshop, Hong Kong*. ACL. 112–120.
- Larsson, Staffan; Cooper, Robin; Engdahl, Elisabet; and Ljungloef, Peter 2000a. Information states and dialogue move engines. *Electronic Articles in Computer and Information Science* 3(7).
- Larsson, Staffan; Cooper, Robin; Engdahl, Elisabet; and Ljungloef, Peter 2000b. Information states and dialogue move engines. *Electronic Articles in Computer and Information Science* 3(7).
- Larsson, Staffan; Santamarta, Lena; and Jönsson, Arne 2000c. Using the process of distilling dialogues to understand dialogue systems. In *Proceedings of 6th International Conference on Spoken Language Processing (ICSLP2000/INTERSPEECH2000), Volume III*. 374–377.
- Larsson, Staffan; Amores, Gabriel; Jonson, Rebecca; and Qesada, Jose 2002. Siridus system architecture and interface report (enhanced version). Project deliverable 6.3, SIRIDUS.
- Larsson, Staffan 1998. Questions under discussion and dialogue moves. In *Proceedings of the Twente Workshop on Language Technology*. 163–171.
- Larsson, Staffan 2000. From manual text to instructional dialogue: an information state approach. In Poesio and Traum 2000. 203–206.

- Larsson, Staffan 2002a. *Issue-based Dialogue Management*. Ph.D. Dissertation, Göteborg University.
- Larsson, Staffan 2002b. Issues under negotiation. In Kristinna, Jokinen; and McRoy, Susan, editors 2002b, *Proceedings of the Third SIGdial Workshop on Discourse and Dialogue*. 103–112.
- Lewin, Ian; Cooper, Robin; Ericsson, Stina; and Rupp, C.J. 2000. Dialogue moves in negotiative dialogues. Project deliverable 1.2, SIRIDUS.
- Lewin, I.; Larsson, S.; Ericsson, S.; and Thomas, J. 2001. The d’homme device selection. Technical Report Deliverable D5.1, D’Homme.
- Lewis, D. K. 1979. Scorekeeping in a language game. *Journal of Philosophical Logic* 8:339–359.
- Mann, W. C. and Thompson, S. A. 1983. Relational propositions in discourse. Technical Report ISI/RR-83-115, USC, Information Sciences Institute.
- Matheson, Colin; Poesio, Massimo; and Traum, David 2000. Modelling grounding and discourse obligations using update rules. In *Proceedings NAACL 2000*, Seattle.
- McGlashan, S.; Burnett, D.; Danielsen, P.; Ferrans, J.; Hunt, A.; Karam, G; Ladd, D.; Lucas, B.; Porter, B.; and Rehor, K. 2001. Voice extensible markup language (voicexml) version 2.0. Technical report, W3C. W3C Working Draft, 23 October 2001.
- Microsoft, 2000. *Universal Plug and Play Device Architecture Version 1.0*. URL: http://www.upnp.org/download/UPnPDA10_20000613.htm.
- Milward, D. 2000. Distributing representation for robust interpretation of dialogue utterances. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics, ACL-2000*. 133–141.
- Moore, Johanna 1993. What makes human explanations effective? In *Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society*, Hillsdale, NJ.
- Pieraccini, R.; Levin, E.; and Eckert, W. 1997. AMICA: The AT&T mixed initiative conversational architecture. In *Proceedings of EUROSPEECH*.
- Poesio, Massimo and Traum, David 1998a. Towards an axiomatisation of dialogue acts. In Hulstij, J. and Nijholt, A., editors 1998a, *Proceedings Twentieth Workshop on the Formal Semantics and Pragmatics of Dialogues*, Enschede. 207–222.
- Poesio, Massimo and Traum, David R. 1998b. Towards an axiomatization of dialogue acts. In *Proceedings of Twendial’98, 13th Twente Workshop on Language Technology: Formal Semantics and Pragmatics of Dialogue*. 207–222.

Poesio, Massimo and Traum, David, editors 2000. *Proceedings of GötaLog 2000*, number 00-5 in GPCL (Gothenburg Papers Computational Linguistics).

Polanyi, L. and Scha, R. 1983. On the recursive structure of discourse. In Ehlich, K. and Riemsdijk, H.van, editors 1983, *Connectedness in Sentence, Discourse and Text*. Tilburg University. 141–178.

Qu, Yan and Beale, Steve 1999. A constraint-based model for cooperative response generation in information dialogues. In *Proceedings of AAAI-99*, Orlando, FL.

Reed, Chris A. and Long, Derek P. 1997. Collaboration, cooperation and dialogue classification. In Jokinen, K., editor 1997, *Working Notes of the IJCAI97 Workshop on Collaboration, Cooperation and Conflict in Dialogue Systems, IJCAI 97*, Nagoya, Japan. 73–78.

Reichman-Adar, R. 1984. Extended man-machine interface. *Artificial Intelligence* 22(2):157–218.

Rich, Charles and Sidner, Candace L. 1998. Collagen: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction* 8(3/4):315–350.

Rich, Charles; Sidner, Candace L.; and Lesh, Neal 2000. COLLAGEN: Applying collaborative discourse theory to human-computer interaction. Technical Report TR-2000-38, Mitsubishi Electric Research Laboratories.

Rose, Carolyn Perstein; Moore, Johanna D.; VanLehn, Kurt; and Allbritton, David 2001. A comparative evaluation of socratic versus didactic tutoring. In *Proceedings 23rd Annual Conference of the Cognitive Science Society*, Edinburgh, Scotland, UK.

San-Segundo, Ruben; Montero, Juan M.; Guitierrez, Juana M.; Gallardo, Ascension; Romeral, Jose D.; and Pardo, Jose M. 2001. A telephone-based railway information system for spanish: Development of a methodology for spoken dialogue design. In *Proceedings of the 2nd SIGdial Workshop on Discourse and Dialogue*. 140–148.

Schiffrin, D. 1987. *Discourse Markers*. Cambridge University Press, Cambridge.

Severinson Eklundh, Kerstin 1983. The notion of language game – a natural unit of dialogue and discourse. Technical Report SIC 5, University of Linköping, Studies in Communication.

Sidner, Candace L. 1994a. An artificial discourse language for collaborative negotiation. In *Proceedings of the fourteenth National Conference of the American Association for Artificial Intelligence (AAAI-94)*. 814–819.

Sidner, Candace. L. 1994b. Negotiation in collaborative activity: A discourse analysis. *Knowledge-Based Systems* 7(4):265–267.

SIRIDUS, 2002. Implemented siridus system architecture (enhanced). Project deliverable 6.4, SIRIDUS.

- Steedman, Mark 2000. *The Syntactic Process*. MIT, Cambridge, Massachusetts.
- Stenström, Anna-Brita 1984. *Questions and Responses*. Lund Studies in English: Number 68. Lund : CWK Gleerup.
- The DISC consortium, 1999. Disc dialogue engineering model. Technical report, DISC, <http://www.disc2.dk/slds/>.
- Torre, Doroteo and others, 2001. User requirements on a natural command language dialogue system. Deliverable 3.1, Siridus Project.
- Traum, D. R. and Allen, J. F. 1994a. Discourse obligations in dialogue processing. In *Proceedings 32nd Annual meeting of the Association for Computational Linguistics (ICSLP92)*. 1–8.
- Traum, D. R. and Allen, J. F. 1994b. Discourse obligations in dialogue processing. In *Proc. of the 32nd Annual Meeting of the Association for Computational Linguistics*, New Mexico. 1–8.
- Traum, David; Bos, Johan; Cooper, Robin; Larsson, Staffan; Lewin, Ian; Matheson, Colin; and Poesio, Massimo 1999. A model of dialogue moves and information state revision. Technical Report Deliverable D2.1, Trindi.
- Traum, D. R. 1994. *A Computational Theory of Grounding in Natural Language Conversation*. Ph.D. Dissertation, University of Rochester, Department of Computer Science, Rochester, NY.
- Tsovaltzi, Dimitra and Matheson, Colin 2002. Formalising hinting in tutorial dialogues. In *EDILOG: 6th workshop on the semantics and pragmatics of dialogue*, Edinburgh, Scotland.
- Van Der Sandt, R. A. 1992. Presupposition projection as anaphora resolution. *Journal of Semantics* 9(4):333–377.
- Van Kuppevelt, Jan; Van Leusen, Noor; Van Rooy, Robert; and Zeevat, Henk, editors 1999. *Proceedings of Amstelogue'99 Workshop on the Semantics and Pragmatics of Dialogue*.
- Verbmobil-Corpus, 1995. Data collection for a speech to speech translation system - scheduling domain. Institut für Phonetik, München.
- Walton, Douglas N. and Krabbe, Erik C. W. 1995. *Commitment in Dialogue*. State University of New York, New York.
- Wittgenstein, Ludwig 1953. *Philosophical Investigations*. Basil Blackwell Ltd.