Linköping Electronic Articles in
Computer and Information Science
Vol. 3(2000): nr 7

Information states and dialogue move engines

Staffan Larsson
Robin Cooper
Elisabet Engdahl
Peter Ljunglöf
Department of linguistics
Göteborg University
Göteborg, Sweden

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis/2000/007/
Revised version

Revised version, published on February nn, by
Linköping University Electronic Press
581 83 Linköping, Sweden
Original version was published on (date), 2000

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
<Author>. <Title>. Linköping Electronic Articles in
Computer and Information Science, Vol. 3(2000): nr 7.
http://www.ep.liu.se/ea/cis/2000/007/. *(date), 2000.*

The URL will also contain links to both the original version and
the present revised version, as well as to the author's home page.

Abstract

We explore the notion of *information state* in relation to dialogue systems, and in particular to the part of a dialogue system we call the *dialogue move engine.* We use a framework for experimenting with information states and dialogue move engines, which is being implemented in the form of TRINDIKIT [14], a toolkit for building dialogue move engines and dialogue systems. We also show how an experimental dialogue system (GoDiS) currently being developed in Göteborg within the framework can be provided with rules to handle accommodation of questions and plans in dialogue.

1 Introduction

As we see it, there are currently two dominant approaches in dialogue modelling and dialogue systems design[1]. General planning and inference systems represent one end of the complexity spectrum; at the other end we have simple slot-filling or finite-state systems used in most practical dialogue system applications today. One of the ideas with the TRINDI architecture is to make it possible to explore the rest of the spectrum. The idea is to avoid the complexity problems that come with general reasoning and planning, but still be able to display complex and natural behaviour.

We use the term *information state* to mean, roughly, the information stored internally by an agent, in this case a dialogue system. A *dialogue move engine* updates the information state on the basis of observed dialogue moves and selects appropriate moves to be performed. In this paper we use a formal representation of dialogue information states that has been developed in the TRINDI[2], SDS[3] and INDI[4] projects to explore the kind of information updates that can be associated with dialogue moves and in the implementation of dialogue systems. The information state approach to dialogue management has been implemented in TrindiKit[5], a toolkit for building and experimenting with information states, dialogue move engines, and dialogue systems. A specific type of information state, based on Ginzburg's notion of Questions Under Discussion (QUD) [6, 7, 8], has been implemented in GoDiS, an experimental dialogue system for information-seeking dialogue.

The structure of this paper is as follows: First, we give a brief description of the TrindiKit architecture. We discuss how rules formulated in terms of conditions and operations on information states can be used to (1) update information states based on observed dialogue moves and (2) select dialogue moves based on the current information state. We then present a particular notion of information state that we have been experimenting with, and give an overview of GoDiS. We look at the role of accommodation in information state transitions and point to examples of two kinds of accommodation: accommodation of questions under discussion and of dialogue plan. We also show how the implementation of these rules yields improved behaviour in the experimental dialogue system. Finally, we outline our view on the relation of our work to previous approaches to dialogue management.

2 The TrindiKit architecture

The aim of TrindiKit is to provide a framework for experimenting with implementations of different theories of information state, information state update and dialogue control. Key to the information state approach is identifying the relevant aspects of information in dialogue, how they are updated, and how updating processes are controlled. This simple view can

[1] This paper reports the state of our research in 1999-2000. Since then, much has happened and the research reported here has been superseded e.g. by [16]. We would like to thank Johan Boye, Joris Hulstijn and Ingrid Zukerman for helpful comments and interesting discussions about earlier versions of this paper.

[2] TRINDI (Task Oriented Instructional Dialogue), EC Project LE4-8314, `www.ling.gu.se/research/projects/trindi/`

[3] SDS (Swedish Dialogue Systems), NUTEK/HSFR Language Technology Project F1472/1997, `http://www.ida.liu.se/~nlplab/sds/`

[4] INDI (Information Exchange in Dialogue), Riksbankens Jubileumsfond 1997-0134.

[5] `http://www.ling.gu.se/research/projects/trindi/trindikit.html`

be used to compare a range of approaches and specific theories of dialogue management within the same framework.

The general architecture we are assuming is shown in Figure 1.

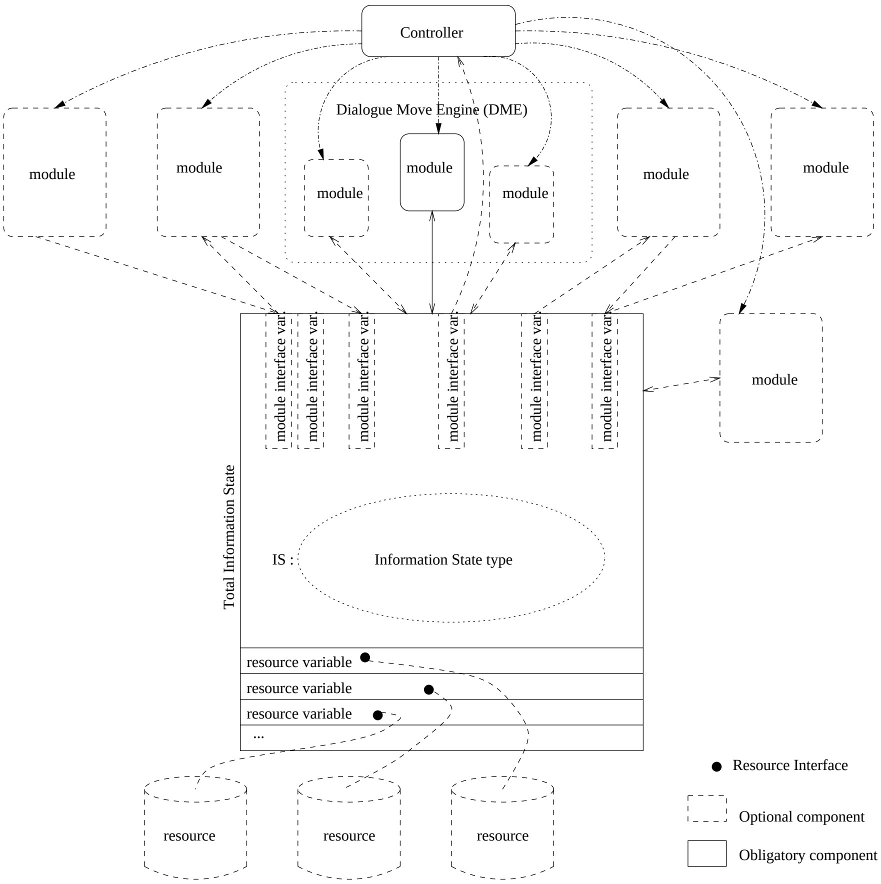

Figure 1: TrindiKit architecture

The components of the architecture are the

- the Total Information State (TIS), consisting of
  - the Information State proper (IS)
  - module interface variables
  - resource interfaces
- the Dialogue Move Engine, consisting of one or more DME modules
- additional (non-DME) modules, e.g. for getting input from the user, interpreting this input, generating system utterances, and providing output for the user.
- a control module, wiring together the other modules, either in sequence or through some asynchronous mechanism.

Any useful system is also likely to need

- Interface variables for modules, which are designated parts of the TIS where the modules are allowed to read and write according to their associated TIS access restrictions.
- Resources such as databases, plan libraries etc. The resources are accessible from the modules through the resource interfaces, which define applicable conditions and (optionally) operations on the resource.

Apart from the general architecture shown in (2), the framework also specifies formats for defining update rules, selection rules and dialogue moves (see section 2.2), and provides a set of tools for experimenting with different information states, rules, and algorithms. Simple interpreters and generators are also provided.

2.1 Building a system using the TrindiKit

To build a dialogue system using the TrindiKit toolkit, one needs to provide definitions of rules, moves and update algorithms, as well as the structure of the information state. Of course, to do this one needs some theory of dialogue. One aim in the design of the TrindiKit is that the formalisation of such a theory should be as close as possible to the actual system as implemented in the TrindiKit, i.e. to isolate low-level implementation issues in the TrindiKit implementation and allow a more high-level implementation of specific dialogue theories.

Although this is not dictated by the TrindiKit architecture, it is always a good idea to keep domain-independent and domain-specific components separate. One way of achieving this is to build a generic[6] system, and then specify a number of domain-dependent resources to make particular specific instances of dialogue systems (see figure 2). For example, the implementer may use TrindiKit to specify information state type, update rules, selection rules and other modules external to the dialogue move engine (control module, interpreter, generator, input and output modules). The collection of these modules that we have specified for GoDiS form such a generic dialogue system. In the case of GoDiS, in order to make a fully instantiated system one in addition needs a lexicon, a database and domain knowledge. The idea is that one should be able to substitute different instances of these modules with the same update rules etc. and thus obtain different instances of generic GoDiS.

2.2 Moves and rules

Traditionally, dialogue moves (or speech acts) are defined using preconditions, effects, and a decomposition [1]. From the perspective of implementing a dialogue move engine, we think it may be useful to think about what a dialogue system (or any dialogue participant) actually needs to do (not necessarily in a sequential order):

- interpret utterance from the user
- update the information state according to the move(s) (supposedly) performed by the user
- select a move/moves to be performed by the system

[6]That is, generic at least given a certain kind of dialogue, e.g. information-seeking; of course one cannot require every system to handle any kind of dialogue.

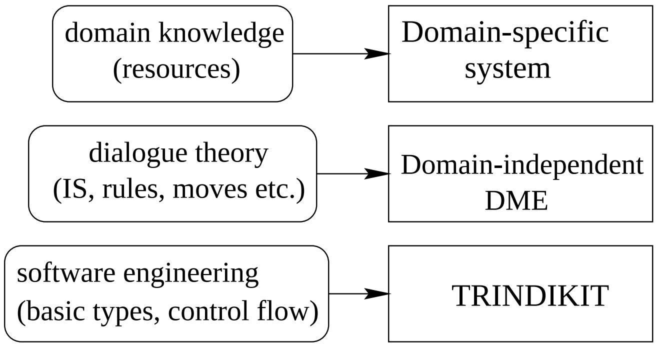

Figure 2: Building a system

- generate appropriate utterance to perform move(s)
- update the information state according to the move(s) performed by the system

Instead of defining the dialogue moves themselves in terms of preconditions and effects, we define *update rules (u-rules)* and *selection rules (s-rules)* for updating the TIS based on the recognised move(s) and selecting the next move(s), respectively.

The update rules are rules that update the information state, e.g. when the user has input something to the system. The selection rules are rules that both update the information state and selects a dialogue move to be executed by the system. Both rule types have preconditions and effects. The preconditions are a list of conditions that must be true of the information state. The effects are a list of operations to be executed if the preconditions are true. The preconditions must guarantee that the effects can be executed.

Dialogue move definitions consist of a name, a type (optional) and a list of number and types of arguments (e.g., speaker, content, etc). Dialogue moves are the output of analysis and input to generation. Also, they are the objects selected by s-rules. U-rules may refer to them, and they may be part of the information state.

We also use the term *tacit move* to refer to the act of applying an update rule, i.e. the act of updating the TIS.

3 Question-based Information State

The question about what should be included in the information state is central to any theory of dialogue management. The notion of information state we are putting forward here is basically a version of the dialogue game board which has been proposed by Ginzburg. We want to stress that the choice of this type of information state is specific to GoDiS; the TrindiKititself does not specify a particular type of informations state, rather it provides methods of specifying different types of information states.

Our general strategy has been to use as simple datastructures as possible and make them successively more complicated as the need arises. This is something that is possible given the kind of notational power and modularity provided by TrindiKit. A change of data structures in successive versions of the system does not involve a wholesale reimplementation of the system. Also, the choice of datatypes should itself be seen as a research issue where the appropriateness of different data-structures for modelling attitudes and discourse units is investigated and explored. This means that the current choice of data-structures for GoDiS may be altered in future versions of the system if there are good reason for it.

We represent information states of dialogue participants as records of the type in Figure 3.

$$\left[\begin{array}{lll} \textsc{private} & : & \left[\begin{array}{lll} \textsc{plan} & : & \textsc{List}(\textsc{Action}) \\ \textsc{agenda} & : & \textsc{Stack}(\textsc{Action}) \\ \textsc{tmp} & : & \left[\begin{array}{lll} \textsc{bel} & : & \textsc{Set}(\textsc{Prop}) \\ \textsc{qud} & : & \textsc{Stack}(\textsc{Question}) \\ \textsc{lm} & : & \textsc{Move} \end{array}\right] \end{array}\right] \\ \textsc{shared} & : & \left[\begin{array}{lll} \textsc{bel} & : & \textsc{Set}(\textsc{Prop}) \\ \textsc{qud} & : & \textsc{Stack}(\textsc{Question}) \\ \textsc{lm} & : & \textsc{Move} \end{array}\right] \end{array}\right]$$

Figure 3: GoDiS information state type

As any abstract datatype, this type of information state is associated with various conditions and operations which can be used to check and update the information state. For example, fst(SHARED.QUD,Q) succeeds if Q is unifiable with the topmost element on the shared QUD stack, and pop(SHARED.QUD) will pop the topmost element off the stack.

The main division in the information state is between information which is private to the agent and that which is shared between the dialogue participants. What we mean by shared information here is that which has been established (i.e. grounded) during the conversation, akin to what Lewis in [18] called the "conversational scoreboard".

The PLAN field contains a dialogue plan, i.e. is a list of dialogue actions that the agent wishes to carry out. The plan can be changed during the course of the conversation. For example, if a travel agent discovers that his customer wishes to get information about a flight he will adopt a plan to ask her where she wants to go, when she wants to go, what price class she wants and so on. The plan must be ordered, since some actions need to be performed before others. However, a stack is not sufficient since it does not allow checking membership of non-topmost elements. (As is explained in Section 5, answers which do not match a question on QUD are matched against questions in the plan, which requires the membership check not available for stacks.)

The AGENDA field contains the short term goals or obligations that the agent has, i.e. what the agent is going to do next. For example, if the other dialogue participant raises a question, then the agent will normally put an action on the agenda to respond to the question. This action may or may not be in the agent's plan. For the agenda, order matters since recent issues are assumed to be more salient and important than older issues. For example, if one participant has an action on the agenda to ask a question but the other participant asks another question, the action to answer the latter question will be pushed on the agenda. Thus, the latter question will be asked first.

We have included a field TMP that mirrors the shared fields. This field keeps track of shared information that has not yet been grounded, i.e. confirmed as having been understood by the other dialogue participant[7]. In

[7] In discussing grounding we will assume that there is just one other dialogue

this way it is easy to delete information which the agent has "optimistically" assumed to have become shared if it should turn out that the other dialogue participant does not understand or accept it. If the agent pursues a cautious rather than an optimistic strategy then information will at first only be placed on TMP until it has been acknowledged by the other dialogue participant whereupon it can be moved from TMP to the appropriate shared field.

The SHARED field is divided into three subfields. One subfield is a set of propositions which the agent assumes for the sake of the conversation. Sets were chosen since no order is imposed on the beliefs, but membership checking is needed.

The second subfield is for a stack of questions under discussion (QUD). These are questions that have been raised and are currently under discussion in the dialogue. The stack structure is meant to reflect the fact that dialogues can be nested; a question q_1 can be met by a counter-question q_2, and only (the story goes) when q_2 has been answered can q_1 be answered.[8]

Actually, there are two rather different notions of QUD at issue here[9]: as a structure to be used in dialogue management on the one hand, and as a semantic structure (used e.g. for resolving ellipsis) on the other. We are taking a simplified view of Ginzburg's idea so that it can be applied to simple implemented systems, and in this sense we are interested in defining QUD as a datastructure which will be used in dialogue management. However, we think it would be a mistake to separate semantics on the one hand from dialogue management on the other. A lot of what Ginzburg is talking about in his semantic approach is a theoretical approach to aspects of dialogue management and that is a large part of the interest in it. What we see as the main distinction between the QUD that we have used and Ginzburg's original notion is that ours is a local QUD. It represents the precise questions that are currently under discussion, i.e. that are so to speak up front and have been explicitly introduced into the dialogue. Ginzburg's original notion, as you say, was much more of a "global" QUD representing questions that arise from what has been said. We suspect that both notions of QUD will ultimately be necessary[10], though perhaps our plan does some of the work of a global QUD. For further discussion of QUD in relation to local and global discourse structure, see [5] and [16].

The third field contains information about the latest move (speaker, move type and content).

4 GoDiS

In Göteborg, an experimental dialogue system called GoDiS (Gothenburg Dialogue System) is being developed based on the framework described above and using the type of information state described in Section 3.

It should be emphasised that this type of information state is specific for GoDiS, and is not part of the TRINDIKIT architecture. The toolkit TRINDI-

participant.

[8] Actually, this is a simplification of Ginzburg's theory where the QUD is a partially ordered set. In the current implementation we do not rely on the fact that the QUD is a stack rather than, say, a set, since there is never more than one question on the QUD.

[9] Thanks to Joris Hulstijn for pointing this out.

[10] In fact, the system reported in [16] uses two separate structures for global and local QUD.

KIT specifies a general architecture and a format for update rules, selection rules, information states and modules that can be interfaced with the dialogue move engine (control, interpreter, generator, input and output modules). This allows the implementer to specify a number of modules to make a generic dialogue system that can be combined with different resources to make particular specific instances of dialogue systems. For example, the implementer may use TRINDIKIT to specify information state type, update rules, selection rules and processing modules external to the dialogue move engine (control module, interpreter, generator, input and output modules). The collection of modules that we have specified for GoDiS form a generic dialogue system for information-seeking dialogue. In order to make a fully instantiated system one in addition needs a lexicon, a database and domain knowledge. The idea is that one should be able to substitute different instances of these modules with the same update rules etc and thus obtain different instances of generic GoDiS. So far, GoDiS has been experimentally adapted for the travel agency and autoroute domains, for handling menu navigation in the Nokia 3210 mobile phone (both in Spanish and Swedish), and as an interface to a handheld computer [10].

4.1 GoDiS architecture

The GoDiS architecture, seen in 4 is an instantiation of the general TRINDIKIT architecture. In addition to the **control** module, there are six modules in GoDiS: **input**, **interpret**, **generate**, **output**, **update** and **select**. The last two are DME modules, which means that they together make up the DME in GoDiS. There are six module interface variables, three resources and a record structure for the information state.

4.2 Interpretation, Generation, Semantics and database

In the current implementation, interpretation and generation are canned, which means that the range of input and output strings is very restricted. However, it is also possible to communicate using moves directly, e.g. by typing `ask(P^(price=P))` instead of `'What is the price?'`.

The semantics (if it deserves the name) represents propositions as pairs of features and values, e.g. (month=april), and questions are λ-abstracts over propositions, e.g. $\lambda x(month = x)$. A set of propositions and a query together constitute a database query which is sent to the database once the system has received sufficient information to be able to answer the question. A question and an answer can be reduced to a proposition using β-reduction. For example, the question λx(month=x) and the answer *april* yield the proposition $[\lambda x(month = x)](april)$, i.e. $(month = april)$.

4.3 Rules, moves and algorithms

In this section we describe some of the rules and algorithm definitions we use. The current algorithms are very simple and the behaviour of the system is therefore mainly dependent on the definitions of the update and selection rules.

Update algorithm:

1. Are there any update rules whose preconditions are fulfilled in the current IS? If so, take the first one and execute the updates specified in the effects of the rule. If not, stop.
2. Repeat.

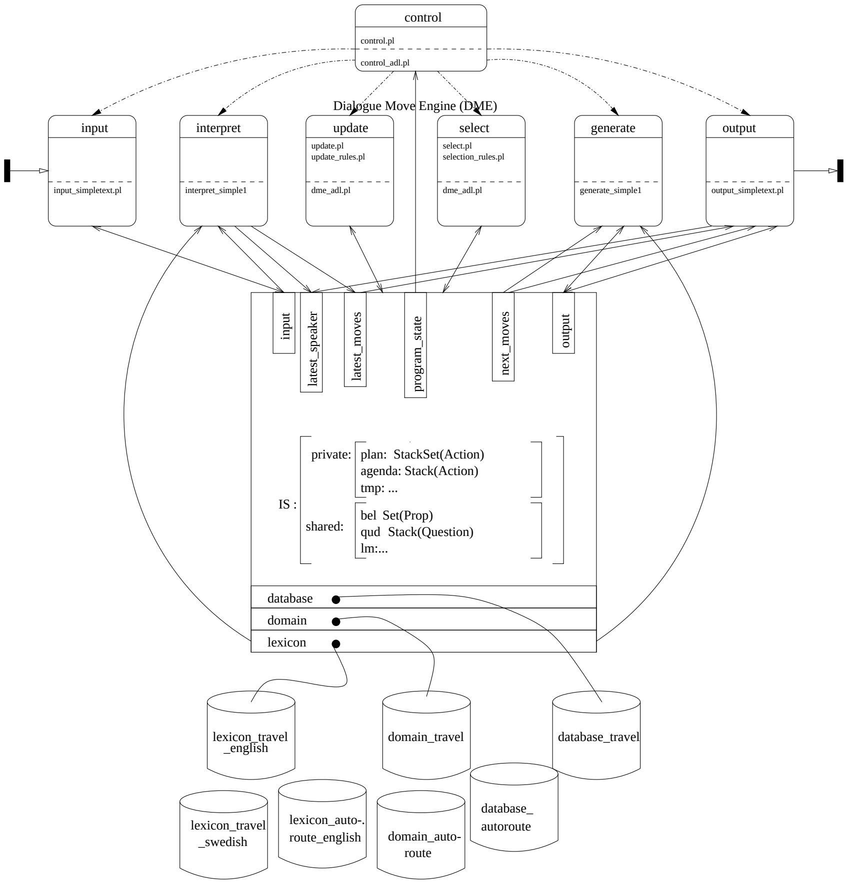

Figure 4: GoDiS architecture

Selection algorithm:

1. Are there any selection rules whose preconditions are fulfilled in the current IS? If so, proceed to step 2. If not, stop.
2. Does the rule specify a dialogue move? If so, stop. If not, execute the updates specified in the effects of the rule.
3. Repeat

Control algorithm:

1. Call the interpreter
2. Call the update module
3. Call the selection module
4. Call the generator
5. Call the update module
6. Repeat

The update rules include rules for question and plan accommodation, as well as rules for handling grounding and rules for integrating the latest move with the DIS. The latter rules look different depending on whether the user or the system itself was the agent of the move. As an illustration, in (1) we see the update rule for integrating an "answer" move when performed by the user, and in (2) the converse rule for the case when the latest move was performed by the system[11].

(1) U-RULE: **integrateLatestMove(answer(usr))**

$$\text{PRE:} \begin{cases} \text{val(SHARED.LM, answer}(usr,A)) \\ \text{fst(SHARED.QUD, } Q \text{),} \\ \text{answer\_to(} Q, A \text{)} \end{cases}$$

$$\text{EFF:} \begin{cases} \text{pop(SHARED.QUD)} \\ \text{reduce}(Q, A, P) \\ \text{add(SHARED.BEL, } P) \end{cases}$$

(2) U-RULE: **integrateLatestMove(answer(sys))**

PRE: val(PRIVATE.TMP.LM, answer(sys, Q, A))

$$\text{EFF:} \begin{cases} \text{set(SHARED.LM, answer}(sys, Q, A)) \\ \text{pop(SHARED.QUD)} \\ \text{reduce}(Q, A, P) \\ \text{add(SHARED.BEL, } P) \end{cases}$$

Here's a paraphrase of rule (1:) "If the latest move was a user answer with content A , and the first question on QUD is Q , and A is an answer to Q, then pop Q off the QUD, perform beta-reduction on Q and A to yield the resulting proposition P, and add P to the shared beliefs. For a more concrete paraphrase, A could be `paris`, Q "`X^(to=X)`" and $P = Q(A) =$ `to(paris)`. A shorter paraphrase is the following: if the user just answered a question on QUD, pop the question off QUD and add the new fact to shared beliefs.

The rule in (2) is the same, except for two things: since the system knows which question it was answering, it is not necessary to check for question-answer relevance; the system only answers questions which are topmost on QUD. A second complicating factor is that the rule in (2) also assumes that the system's move has not yet been grounded - it is stored in PRIVATE.TMP.LM rather than SHARED.LM.

[11]Note that this definition embodies an optimistic approach to grounding by putting answer(sys, Q, A) in SHARED.LM, thereby assuming the systems utterance was understood by the user. Also, the system optimistically assumes that the user accepts the resulting proposition P by adding it to SHARED.BEL.

4.4 Dialogue plans

In our implementation, the domain resource includes, among other things, a set of *dialogue plans* which contain information about what the system should do in order to achieve its goals. Traditionally [2], it has been assumed that general planners and plan recognizers should be used to produce cooperative behaviour from dialogue systems. On this account, the system is assumed to have access to a library of domain plans, and by recognizing the domain plan of the user, the system can produce cooperative behaviour such as supplying information which the user might need to execute her plan. Our approach is to directly represent ready-made plans for engaging in cooperative dialogue and producing cooperative behaviour (such as answering questions) which indirectly reflect domain knowledge, but obviates the need for dynamic plan construction.

Typically, the system has on the agenda an action to respond to a question. However, the move for answering the question cannot be selected since the system does not yet have the necessary information to answer the question. The system then tries to find a plan which will allow it to answer the question, and this plan will typically be a list of actions to raise questions; once these questions have been raised and the user has answered them, the system can provide an answer to the initial question. This behaviour is similar to that of many natural language database interfaces, but the difference is that the architecture of our system allows us to improve the conversational behaviour of the system simply by adding some new rules, such as the accommodation rules described below.

5 Accommodation

We define dialogue moves as updates to information states directly associated with utterances. If one takes a dialogue or information update perspective on Lewis' notion of accommodation, it corresponds to moves that are tacit (i.e. not associated with an utterance). Tacit moves can be seen as applications of update rules, which specify how the information state should be updated given that certain preconditions hold. Tacit moves could also be called "internal" or "inference" moves. The 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 either be understood as an assertion of that sentence and a presupposition that there is a king of France or as an assertion of the sentence "There is a king of France and he is bald". However, if we assume an additional tacit accommodation move before the integration of the information expressed by the utterance then we can say that the utterance always has the same interpretation.

In a similar way we can simplify our dialogue move analysis by extending the use of tacit moves so that the updates to the information state normally associated with a dialogue move are actually carried out by tacit moves. One argument for doing this is that very few (if any) effects of a move are guaranteed as a consequence of performing the move; rather, the actual resulting updates depend on reasoning by the addressed participant. Thus, we define an update rule **integrateLatestMove** which, given that the latest move was accepted by the system, performs the appropriate update operations. The updates for a move are different depending on whether it was the system or the user who made the move, but the same module is used in both cases.

5.1 Accommodating a question onto QUD

Dialogue participants can address questions that have not been explicitly raised in the dialogue. However, it is important that a question is available to the agent who is to interpret it because the utterance may be elliptical. Here is an example from a recorded dialogue[12]:

(3) `$J: vicken månad ska du åka`
(what month do you want to go)
`$P: 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)

The strategy we adopt for interpreting elliptical utterances is to think of them as short answers (in the sense of Ginzburg [8]) to questions on QUD. A suitable question here is *What kind of price does P want for the ticket?*. This question is not under discussion at the point when P says "as cheap as possible". But it can be figured out since J knows that this is a relevant question. In fact it will be a question which J has as an action in his plan to raise. On our analysis it is this fact which enables A to interpret the ellipsis. He finds the matching question on his plan, accommodates by placing it on QUD and then continues with the integration of the information expressed by *as cheap as possible* as normal. Note that if such a question is not available then the ellipsis cannot be interpreted as in the dialogue in (4).

(4) A. What time are you coming to pick up Maria?
B. Around 6 p.m. As cheap as possible.

This dialogue is incoherent if what is being discussed is when the child Maria is going to be picked up from her friend's house (at least under standard dialogue plans that we might have for such a conversation).

5.2 Accommodating the dialogue plan

After an initial exchange for establishing contact the first thing that P says to the travel agent in our dialogue is:

(5) `$P: flyg ti paris`
< flights to Paris >

This is again an ellipsis which on our analysis has to be interpreted as the answer to a question in order to have content. As no questions have been raised yet in the dialogue (apart from whether the participants have each

---

[12] We will illustrate our discussion from a Swedish human-human dialogue in the travel booking domain that 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. The full transcription is available from `http://www.ling.gu.se/SLSA/dialog.html`.

other's attention) the travel agent cannot find the appropriate question on his plan. Furthermore, as this is the first indication of what the customer wants, the travel agent cannot have a plan with detailed questions. We assume that the travel agent has various plan types in his domain knowledge determining what kind of conversations he is able to have. E.g. he is able to book trips by various modes of travel, he is able to handle complaints, book hotels, rental cars etc. What he needs to do is take the customer's utterance and try to match it against questions in his plan types in his domain knowledge. When he finds a suitable match he will accommodate his plan, thereby providing a plan to ask relevant question for flights, e.g. when to travel?, what date? etc. Once he has accommodated this plan he can proceed as in the previous example. That is, he can accommodate the QUD with the relevant question and proceed with the interpretation of ellipsis in the normal fashion.

This example is interesting for a couple of reasons. It provides us with an example of "recursive" accommodation. The QUD needs to be accommodated, but in order to do this the dialogue plan needs to be accommodated. The other interesting aspect of this is that accommodating the dialogue plan in this way actually serves to drive the dialogue forward. That is, the mechanism by which the agent interprets this ellipsis, gives him a plan for a substantial part of the rest of the dialogue. This is a way of capturing the intuition that saying *flights to Paris* to a travel agent immediately makes a number of questions become relevant.

5.3 Associating accommodation with tacit moves

Update rules can be used for other purposes than integrating the latest move. For example, one can provide update rules which accommodate questions and plans. One possible formalization of the **accommodate_question** move is given in (6). When interpreting the latest utterance by the other participant, the system makes the assumption that it was a **reply** move with content A. This assumption requires accommodating some question Q such that A is a relevant answer to Q. The check operator "answer-to(A, Q)" is true if A is a relevant answer to Q given the current information state, according to some (possibly domain-dependent) definition of question-answer relevance[13].

(6)
$$
\begin{array}{ll}
\text{U-RULE:} & \textbf{accommodateQuestion}(Q, A) \\
\text{PRE:} & \left\{ \begin{array}{l} \text{val}(\text{SHARED.LM}, \text{answer}(\text{usr}, A)), \\ \text{in}(\text{PRIVATE.PLAN}, \text{raise}(Q)) \\ \text{answer-to}(\ A, Q\ ) \end{array} \right. \\
\text{EFF:} & \left\{ \begin{array}{l} \text{del}(\text{PRIVATE.PLAN}, \text{raise}(Q)) \\ \text{push}(\text{SHARED.QUD}, Q) \end{array} \right.
\end{array}
$$

6 Accommodation in a dialogue system

In this section we show an example of how the dialogue system described above can handle accommodation of questions and plans. The example is actual (typed) dialogues with the system, supplemented with information

[13]The definition of relevance implemented in the current GoDiS is very simple; basically, it encodes for each question a set of possible ("relevant") answers. For example, any city name is a relevant answer to the question "Where do you want to go?".

about dialogue moves, tacit moves, and (partial) illustrations of the systems information state at different stages of the dialogue. The dialogue fragment can be seen in Figure 5 and the resulting information state in Figure 6.

After interpreting the users utterance as an **answer** move with the content `[how=plane,to=paris]`, the system starts checking if there are any u-rules which apply. Following the ordering of the rules given in the list of rule definitions, it first checks if it can perform **integrateLatestMove(answer(usr))**. However, this rule requires that the content of the answer must be relevant to the topmost question on QUD. Since the QUD is empty, the rule does not apply. It then tries to apply the **accommodateQuestion** rule, but since the plan is empty this rule does not apply either. However, **accommodatePlan** (7) does apply, since there is (in the domain knowledge resource) a plan such that the latest move matches that plan. More precisely, the latest move provides an answer to a question Q such that raising Q is part of the plan.

(7) U-RULE: **accommodatePlan**

$$\text{PRE: } \left\{ \begin{array}{l} \text{empty(PRIVATE.PLAN)} \\ \text{empty(SHARED.QUD)} \\ \text{empty(PRIVATE.AGENDA)} \\ \text{val(SHARED.LM, } LM \text{)} \\ \text{domain::matches\_plan(} LM, Plan \text{)} \end{array} \right.$$

$$\text{EFF: } \left\{ \text{set(PRIVATE.PLAN, } Plan\text{)} \right.$$

Once this rule has been executed, the update algorithm starts from the beginning of the rule list. This time, it turns out the preconditions of **accommodateQuestion** hold, so the rule is applied. As a consequence of this, the preconditions of **integrateLatestMove(answer(usr))** now hold, so that rule is applied. Actually, it turns out that the latest move is also relevant to a second question (concerning the destination) in the plan, so that question is also accommodated and its answer integrated. Since no additional u-rules apply, the system proceeds to perform the next action on the plan: asking where the user wants to travel from. At the end of the dialogue fragment, the information state after the system has uttered this question is shown.

7 Conclusion

We would argue that general inference and/or planning may not be the ultimate solution for dialogue management because of complexity problems. Simple slot-filling strategies are too simple, and we argue that dialogue plans coupled with mechanisms for question and task accommodation operate on a level of complexity which is just right for many kinds of dialogue. We do not claim that the mechanisms we describe cannot be implemented in any other architecture. Rather, we argue that the TRINDI approach provides natural and intuitive means for implementing dialogue behaviours. As an example, the mechanism of question accommodation in GoDiS only required adding one or two simple, domain-independent update rules. For further reading about TrindiKit and its relation to other approaches to dialogue management, we refer to [17].

```
$S: Welcome to the travel agency

$U: flights to paris.
```

$$\left[\begin{array}{lcl} \mathtt{PRIVATE} & = & \left[\begin{array}{lcl} \mathtt{PLAN} & = & \langle\rangle \\ \mathtt{AGENDA} & = & \langle\rangle \\ \mathtt{TMP} & = & \ldots \end{array}\right] \\ \mathtt{SHARED} & = & \left[\begin{array}{lcl} \mathtt{BEL} & = & \{\} \\ \mathtt{QUD} & = & \langle\rangle \\ \mathtt{LM} & = & \mathtt{answer(usr,[how=plane,to=paris])} \end{array}\right] \end{array}\right]$$

```
# accommodatePlan
     # set(private.plan,[raise(A^(how=A)),
                         raise(B^(to=B)),
                         raise(C^(return=C)),
                         raise(D^(month=D)),
                         raise(E^(priceclass=E)),
                         respond(F^(price=F))])

# accommodateQuestion
     # del(private.plan,raise(A^(how=A)))
     # push(shared.qud,A^(how=A))

# integrateLatestMove(answer(usr))
     # pop(shared.qud)
     # add(shared.bel,how=plane)

# accommodateQuestion
     # del(private.plan,raise(A^(to=A)))
     # push(shared.qud,A^(to=A))

# integrateLatestMove(answer(usr))
     # pop(shared.qud)
     # add(shared.bel,to=paris)

# refillAgenda
     # pop(private.plan)
     # push(private.agenda,raise(A^(return=A)))

private =
    agenda = [raise(A^(return=A))]
    plan = [raise(A^(month=A)),
            raise(B^(priceclass=B)),
            respond(C^(price=C))]
shared =
    bel= [(to = paris)
          (how = plane)]
    lm = answer(usr,[how=plane,to=paris])

$S: From where do you want to go?
```

Figure 5: Sample GoDiS dialogue

$$\begin{bmatrix} \texttt{PRIVATE} & = & \begin{bmatrix} \texttt{PLAN} & = & \left\langle \begin{array}{l} \texttt{raise(R\^{}(return=R)),} \\ \texttt{raise(M\^{}(month=M)),} \\ \texttt{raise(C\^{}(class=C)),} \\ \texttt{respond(P\^{}(price=P))} \end{array} \right\rangle \\ \texttt{AGENDA} & = & \langle\rangle \\ \texttt{TMP} & = & \ldots \end{bmatrix} \\ \texttt{SHARED} & = & \begin{bmatrix} \texttt{BEL} & = & \{\texttt{(to=paris),(how=plane)}\} \\ \texttt{QUD} & = & \langle \texttt{X\^{}(from=X)} \rangle \\ \texttt{LM} & = & \texttt{ask(sys,Y\^{}(from=Y))} \end{bmatrix} \end{bmatrix}$$

Figure 6: Information state resulting from the exchange in Figure 5.

8 Current and future research

Lately, we have been experimenting with translating menu-driven interfaces into dialogue systems by converting menu structures into dialogue plans. This requires a more complex plan representation, with conditionals and embedded subplans. While the dialogue plan presented in this paper is very simple and very much like a slot-and-filler frame, the menu-derived plans go beyond what can be done with frames as standardly conceived. We claim that his supports our view that the dialogue plan approach is more general than the frame-filling approach, while still not being too computationally complex. GoDiS has also been modified to handle instructional dialogue [15], where complex plans are also needed.

We are currently extending the GoDiS DME to handle negotiative dialogue, where e.g. several different solutions to a problem (answers to a question) can be discussed and compared before one is settled on. By contrast, the current GoDiS can only discuss one object (e.g. flight) at a time. This extension will also require abandoning the simple feature-value semantics currently used, and adopting a semantics similar to first order logic.

In future work we hope to investigate in more detail the relation between the QUD-based approach to dialogue management, as implemented in GoDiS, and other approaches, including plan-based approaches such as [9], [12] and [19], strategies based on general reasoning such as [21] and [22], frame-based approaches such as [3], and obligation-based dialogue management ([11]).

References

[1] J. F. Allen. *Natural Language Understanding.* Benjamin Cummings, Menlo Park, CA, 1987.

[2] J. F. Allen and C. Perrault. Analyzing intention in utterances. *AIJ*, 15(3):143–178, 1980.

[3] Jennifer Chu-Carroll. Mimic: An adaptive mixed initiative spoken dialogue system for information queries. In *Proceedings of the 6th Conference on Applied Natural Language Processing*, pages 97–104, 2000.

[4] R. Cooper and S. Larsson. Dialogue moves and information states. In *Proc. of the Third IWCS*, Tilburg, 1999.

[5] Robin Cooper, Elisabet Engdahl, Staffan Larsson, and Stina Ericsson. Accommodating questions and the nature of qud. In Poesio and Traum [20], pages 57–61.

[6] J. Ginzburg. Dynamics and the semantics of dialogue. In Seligman and Westerståhl [23].

[7] J. Ginzburg. Interrogatives: Questions, facts and dialogue. In *The Handbook of Contemporary Semantic Theory* [13].

[8] J. Ginzburg. Clarifying utterances. In J. Hulstijn and A. Niholt, editors, *Proc. of the Twente Workshop on the Formal Semantics and Pragmatics of Dialogues*, pages 11–30, Enschede, 1998. Universiteit Twente, Faculteit Informatica.

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

[10] Rebecca Jonson. Agenda talk - a talking filofax developed with the trindikit toolkit. Master's thesis, Computational Linguistics, Göteborg University, 2000.

[11] Jorn Kreutel and Colin Matheson. Information states, obligations and intentional structure in dialogue modelling. In *Proceedings of the 3rd International Workshop on Human-Computer Conversation*, 2000.

[12] Lynn Lambert and Sandra Carberry. A triparite plan-based model of discourse. In *Proceedings of the 29^{th} Annual Meeting of the Association for Computational Linguistics*, pages 47–544, 1991.

[13] ed. Lappin, Shalom. *The Handbook of Contemporary Semantic Theory.* Blackwell, Oxford, 1996.

[14] S. Larsson, P. Bohlin, J. Bos, and D. Traum. Trindikit 1.0 manual. deliverable D2.2 D2.2 - Manual, TRINDI, 1999.

[15] Staffan Larsson. From manual text to instructional dialogue: an information state approach. In Poesio and Traum [20], pages 203–206.

[16] Staffan Larsson. *Issue-based Dialogue Management.* PhD thesis, Göteborg University, 2002.

[17] Staffan Larsson and David Traum. Information state and dialogue management in the trindi dialogue move engine toolkit. *NLE Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering*, pages 323–340, 2000.

[18] D. K. Lewis. Scorekeeping in a language game. *Journal of Philosophical Logic*, 8:339–359, 1979.

[19] D. J. Litman and J. F. Allen. Discourse processing and commonsense plans. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 366–388. The MIT Press, 1990.

[20] Massimo Poesio and David Traum, editors. *Proceedings of Götalog 2000*, number 00-5 in GPCL (Gothenburg Papers Computational Linguistics), 2000.

[21] B. Raskutti and I Zukerman. Generating queries and replies during information-seeking interactions. *International Journal of Human Computer Studies*, 47(6):689–734, 1997.

[22] D. Sadek, A. Ferrieux, and A. Cozannet. Towards an artificial agent as the kernel of a spoken dialogue system: a progress report. In *Proc. of the AAAI Workshop on Integration of Natural Language and Speech*, 1994.

[23] Jerry Seligman and Dag Westerståhl, editors. *Logic, Language and Computation*, volume 1. CSLI Publications, 1996.