Dialogue Dynamics and Levels of Interaction

David Traum    Peter Bohlin    Johan Bos    Stina Ericsson
Staffan Larsson    Ian Lewin    Colin Matheson
David Milward

Distribution: PUBLIC

TRINDI

Task Oriented Instructional Dialogue
LE4-8314
Deliverable D3.1
February 2000
Task Oriented Instructional Dialogue

Gothenburg University
Department of Linguistics

University of Edinburgh
Centre for Cognitive Science and Language Technology Group, Human Communication Research Centre

Universität des Saarlandes
Department of Computational Linguistics

SRI Cambridge

Xerox Research Centre Europe

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

The TRINDI Project Administrator
Department of Linguistics
Göteborg University
Box 200
S-405 30 Gothenburg, Sweden
trindi@ling.gu.se

Copies of reports and other material can also be accessed from the project’s homepage, http://www.ling.gu.se/research/projects/trindi.
©2000, The Individual Authors

No part of this document may be reproduced or transmitted in any form, or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission from the copyright owner.
Responsibility for authorship is divided as follows. David Traum was the overall editor. The interaction grids in Chapter 2 were developed by the TRINDI consortium participants. Chapter 3 was written by Johan Bös. Chapter 4 was written by Ian Lewin and David Milward. Chapter 5 was written by Staffan Larsson, Peter Bohlin and Stina Ericsson. Chapter 6 was written by Colin Matheson.
Contents

1 Introduction ......................................................... 11

2 Interaction Grids ............................................... 15
   2.1 Input modalities ........................................... 15
      2.1.1 Input Interface ...................................... 15
      2.1.2 input content ....................................... 16
      2.1.3 interpretation ....................................... 16
      2.1.4 input dialogue moves ............................... 17
      2.1.5 Input Grid points .................................... 17
   2.2 Output Modalities .......................................... 18
      2.2.1 output dialogue moves ............................. 18
      2.2.2 Generation .......................................... 19
      2.2.3 Output content ..................................... 19
      2.2.4 Output Interface .................................... 20
      2.2.5 Output Grid points ................................. 20

3 Midas ............................................................. 23

7
3.1 Introduction ......................................................... 23
3.2 Input Modalities ..................................................... 24
   3.2.1 Input Interface ............................................... 24
   3.2.2 Input Content ............................................... 25
   3.2.3 Interpretation ............................................... 25
   3.2.4 Input Dialogue Moves ...................................... 27
3.3 Output Modalities .................................................. 28
   3.3.1 Output Dialogue Moves .................................... 28
   3.3.2 Generation .................................................. 28
   3.3.3 Output Content ............................................. 29
   3.3.4 Output Interface ............................................ 29
3.4 Grid Points .......................................................... 29
   3.4.1 Input Grid Points .......................................... 29
   3.4.2 Output Grid Points ........................................ 32

4 SRI Autoroute Dialogue Demonstrator .................................. 33
   4.1 Overview ......................................................... 33
   4.2 Input and Output ............................................... 33
      4.2.1 IO for a Dialogue Systems based on Conversational Games .... 34
      4.2.2 IO for an Educational System for Conversational Games .......... 37
      4.2.3 IO for a Robust Interpreter for Spoken Dialogues ............... 39
   4.3 Interpreter ...................................................... 40
4.3.1 A Simple Phrase Spotter ........................................ 40
4.3.2 A Robust Interpreter ........................................... 41
4.4 Generator .......................................................... 43
4.5 Summary: Position in the Grid of Interaction ................. 43
  4.5.1 Input Modalities ............................................... 43
  4.5.2 Output Modalities ............................................ 45

5 Godis ................................................................. 47
  5.1 Introduction ..................................................... 47
  5.2 GoDiS Input Modalities ......................................... 47
    5.2.1 Interface and content ...................................... 47
    5.2.2 Dialogue moves and interpretation ....................... 50
  5.3 GoDiS Output Modalities ....................................... 51
    5.3.1 Dialogue moves ............................................. 51
    5.3.2 Generation, content and interface ....................... 52
  5.4 Grid Points ..................................................... 53
    5.4.1 Input modalities ........................................... 53
    5.4.2 Output modalities ......................................... 53

6 EDIS ................................................................. 55
  6.1 Dialogue Acts .................................................. 56
  6.2 Input .......................................................... 56
  6.3 Interpretation .................................................. 57
6.4 Generation ......................................................... 58
6.5 Output .............................................................. 59
6.6 Thistle Interface ................................................... 60
6.7 Interaction Grids .................................................. 60
Chapter 1

Introduction

A large part of the TRINDI project has involved specifying computational theories of information state in dialogue, and particularly on the update of this information state as a dialogue proceeds. As outlined in, e.g., [Cooper et al., 1999, Traum et al., 1999], participation in a dialogue can be viewed as making dialogue moves, and a main focus has been on which updates to information state should be made when particular dialogue moves have been observed, and which dialogue moves should be performed, given particular information states. Several theories of information state corresponding to this general view have been developed and refined, e.g., as described in [Traum et al., 1999]. A dialogue move engine has also been developed, in which such theories can be implemented [Larsson et al., 1999] as part of a dialogue system.

Updates to information state on the basis of dialogue moves is only part of a complete dialogue system, however. Dialogue moves are generally abstractions over actual performances by participating agents. Ideally, a system could communicate with a user, using a full range of natural language, as well as multi-media paralinguistic and gestural communications. The difficulty arises, however, in that the richer the communication language, the more difficult it may be to associate the actual communications with dialogue moves that are understood by the dialogue move engine and rules for updating information state. In [Cooper et al., 1999], we examined coding of natural language dialogues (transcribed speech) with dialogue moves. In a system, either such coding must be automated, or a more restricted input mechanism must be present.

In this document, we will consider the other aspects of dialogue communication, beyond the information state updates, proper. In general, this will involve aspects of both input to the system (for perceiving communications by a user) and output of the system (for presenting communications to the user). Each of these can be divided into four interrelated subcomponents, for eight aspects all together:
input interface - concerning the way the user selects the input to the system, e.g., speech, typed text, menu selection, physical manipulation, web form, etc.

input content - concerning what kind of symbolic or analogue information is used as the input, regardless of how this might have been retrieved by the system

interpretation - how the input is converted to dialogue moves, or other form that can be used to directly update the information state

input dialogue moves - which dialogue moves are used as the results of interpretation, could be either a complete set corresponding to a theory of dialogue, or some more limited set of moves, restricting the types of dialogue contributions that a user can make. The set may also be more general and applicable to any sort of input or interface, or include some acts that are particular to certain modalities (such as gesture and prosody).

output dialogue moves - which dialogue moves are produced by the system. Can also be the full range of acts in a taxonomy, or some restricted subset.

generation - how selected dialogue moves are realized as output content

output content the symbolic or analogue information to be presented to the user.

output interface - the way the output information is presented to the user (audio signal, text, web page, graphics, gestures, etc).

Given a fixed set of dialogue moves and information state theory, one could still have a large variety of different dialogue systems, depending on how the components above are put together. For instance, one might conceive of input content as natural language strings. However, these strings might either be typed from the keyboard, or selected from a menu list. Likewise, the interpretation procedure might be full natural language processing or some simpler matching procedure used to find a closest match to one of a fixed set of messages. Limitations on any of the dimensions will tend to limit the overall system breadth of coverage. These limitations would hopefully be to a set adequate for performance of the expected dialogue task.

These different dimensions can be encapsulated as components within a dialogue system built using TRINDIKIT [Larsson et al., 1999]. The architecture of TRINDIKIT is shown in figure 1.1. The information state is shown below, while processing modules are shown above. On the left side is the input interface, by which the user interacts with the system. This interaction is monitored by the input module, and results are placed in the input interface variable, for the interpretation module (and perhaps others) to work with. The results of the interpretation modules’ processing is a set of dialogue moves, placed in the latest_moves interface variable. This is the proper input for the dialogue move engine to
use to update the information state. Moves selected to be realized as output are placed by the dialogue move engine in the next_moves interface variable, serving as input for the processing of the generation module. The output of the generation module is the output content, placed in the output interface variable. This is used by the output module to present the content to the user, via the user interface.

![Diagram of the TRINDIKIT architecture](image)

Figure 1.1: The TRINDIKIT architecture

In the next chapter, we present a “grid” of different possibilities for each of these dimensions in terms of choices that could be adopted for inclusion in a dialogue system. Some of these have been adopted in actual dialogue systems, using TRINDIKIT. These choices are briefly described in the following chapters. More information on the whole systems can be found in [Bos et al., 1999].
Chapter 2

Interaction Grids

In this chapter we lay out various possibilities for each of the eight I/O components mentioned in the previous chapter. Combining aspects for each of the components yields a multi-dimensional grid of possible I/O facilities. This list is meant to be suggestive and cumulative, rather than fully exhaustive. In the following chapters, we present the types of I/O used by several simple dialogue systems, using different concepts of information state, which have been built with the TRINDIKIT architecture.

2.1 Input modalities

2.1.1 Input Interface

There are a number of different types of input interfaces that can be used, including:

1. simple menu selection or buttons (choosing one item)
2. complex menu selection (multiple choices for various components)
3. gesture (with mouse, or other pointing device)
4. text typing
5. handwriting or gesture with pen device
6. physical gesture captured on camera (including both large-scale body gestures, as with hands and arms, as well as more subtle movements, e.g., facial expressions or gaze).

7. Speaking

8. combinations of the above

### 2.1.2 Input Content

There are also a number of types of content for the input. There is a natural connection between input and content type, but this connection is not necessarily present in a one to one fashion. For instance, one might have a menu device which produces text strings (rather than a value from a fixed range), or allow text typing that corresponds to one of a fixed set of allowable message values. Some of the types of input content include:

1. value in a fixed set of allowable messages
2. string in a well defined formal grammar
3. natural language text message
4. speech signal
5. video signal
6. other analog signals, such as mouse or pen trajectories
7. combinations of the above

### 2.1.3 Interpretation

Interpretation is a function from the input content to the dialogue moves that are used by the dialogue move engine. There are a number of different ways of interpreting the input, depending of course on the type of input, but also on the desired output, in terms of dialogue moves. There are again, some “natural” connections between particular input forms and interpretation mechanisms, but there are many possibilities. For simple input types, such as a value in a range, interpretation is usually just the null function, or a straightforward mapping from the value to the desired dialogue move. For strings in a grammar, a parser is the natural interpretation method. However, one could also use different methods for same input type, e.g., pattern matching, statistical fitting, decision

16
trees, or other methods. Also, in many systems, interpretation is actually a complex, multi-stage process. E.g., for interpreting speech, one method would be to start with a speech recognizer using statistical methods (e.g., HMMs or neural networks), then pass the result to a natural language syntactic parser, then a semantic interpreter, and finally other context-sensitive discourse processesing modules (e.g., for anaphora and ellipsis resolution), and general logical inference, before finally ending up with the resulting dialogue moves.

Some

1. functional translation (i.e., lookup)
2. pattern matching from set patterns
3. parsing according to a fixed grammar
4. statistical techniques
5. deductive or abductive inference
6. combinations of the above

2.1.4 input dialogue moves

Assuming an abstract set of dialogue moves, there are several limitations on which moves might be allowed, for restricted systems, including:

1. answers to questions
2. restricted instructions to system
3. questions to system
4. repairs and meta-dialogue to system
5. combinations of the above

2.1.5 Input Grid points

Some of the combinations of these modalities include the following:
U1: Physical menu selection The user may select (e.g. by mouse or TV hand set) from a number of options presented visually by the machine. This will only work in a system that is conceived in an essentially multi-modal way. That is selected areas of the screen have meaning in terms of the information store which the system possesses and the system can use its interpretation of the user’s actions in the generation of its own natural language contributions to the dialogue.

U1s: Speech enhanced menu selection The user is able to perform menu selection or give commands to icons using the modality of speech. In one variant it is possible for the user to assign arbitrary spoken names to menu items or icons.

U2: Natural language answers to questions posed by the system The system asks very precise questions of the user which ”naturally” control” the user’s responses. e.g. What is the name of the street which you want to go to? rather than Where do you want to go?.

U2s: Spoken language answers to questions posed by the system

U3: Restricted instructions to the system The user is allowed to issue instructions in a restricted language (e.g. a restricted language allowed by the Core Language Engine). Of course, there are issues as to how easy it is for a user to learn such a language or for the interaction to be designed in such a way that user responses are naturally constrained to fall within the restricted language.

U3s: Restricted spoken instructions to the system

U4: Questions to the system The user is allowed to question the system freely and the system responds (robustly) as best it can.

U4s: Spoken questions to the system Given current technology we cannot rely on the system having free natural language generation capabilities either. We will therefore consider different levels of system natural language capability

2.2 Output Modalities

2.2.1 output dialogue moves

There are similar restrictions possible on the types of dialogue moves a system might make, including:

1. questions
2. answers to questions
3. directed actions
4. repairs and meta-dialogue
5. combinations of the above

2.2.2 Generation

There are different ways that the dialogue moves can be converted to output content, depending of course on what that output content is. Some possibilities include:

1. look-up of pre-stored text
2. template-filling
3. grammar-based lexical selection and realization
4. dialogue-appropriate realization (including appropriate use of inter-sentential anaphora, ellipsis, short answers, etc.)
5. combinations of the above

2.2.3 Output content

Regardless of the generation method, there are also multiple types of output content, including at least the following:

1. range values
2. text messages
3. annotated text (e.g., with prosodic labels or formatting instructions).
4. speech output
5. graphical layout
6. video animation
7. actions instructions (e.g., for controlling a robot or device to produce gestures)
8. combinations of the above
2.2.4 Output Interface

There are of course various possibilities for presenting the output content to the user, including:

1. text presented to the screen or other peripheral device
2. audio signals
3. graphical presentation
4. video presentation
5. physical gestures
6. combinations of the above

2.2.5 Output Grid points

Some of the combinations of these modalities include the following:

S1 : Pre-stored text The system responds to the user with a document from a collection of stored documents.

S1s: Pre-stored speech The system responds to the user with a "prepared speech" that is stored in the system.

S2: Generated language The system generates text based on its knowledge base and its profile of the user.

S2s: Generated speech The system generates speech based on its knowledge base and its profile of the user.

S3: Generated written dialogue contributions The system generates language appropriate to the current state of the dialogue.

S3s: Generated spoken dialogue contributions The system generates spoken language appropriate to the current state of the dialogue.

Standard hypertext technology is an example of U1/S1 level interaction where a mouse-click is interpreted as an instruction to respond with a unique piece of pre-stored text. The
highest levels of interaction U4s/S3s are currently not feasible for practical systems. There is a need to develop technology that will enable the enhancement of current systems so that they can score higher in this classification.
Chapter 3

Midas

3.1 Introduction

MIDAS (Multiple Inference-based Dialogue Analysis System) is a prototype of a dialogue system exploring semantic representation and first-order reasoning to model and plan the ongoing dialogue. The domain in which MIDAS is implemented covers the “Autoroute” scenario. The system has initiative throughout the whole dialogue.

The type of information state used in the MIDAS dialogue-system is centered around Discourse Representation Theory. This makes it relatively easy to integrate it with sophisticated natural language processing modules such as parsers, semantic-construction tools, and inference engines, as they are already existing and can be taken ‘off-the-shelf’. Speech-input is not realized in the MIDAS-system, and will evoke considerable work on the interface between the speech-lattice and the parse, as robustness will play an important role here.

MIDAS is developed (and actually still under development) by Johan Bos at the Department of Computational Linguistics, University of the Saarland. A web-interface is available at http://www.coli.uni-sb.de/~bos/trindi/midas.html.
3.2 Input Modalities

3.2.1 Input Interface

The input interface of MIDAS is a combination of complex menu selection and text typing. At the start of a session, users can select buttons to monitor different components of MIDAS during the dialogue (see Fig. 3.1). These selections can be revised during the course of a dialogue. The menu items are listed below:

![Screen-shot MIDAS (starting)](image)
• **Information State**
  Displays the Information State of the dialogue after each update;

• **Readings**
  Displays the readings (the semantic analyses) of the user's utterance analysis;

• **Update Rules**
  Displays the update rules applied to the information state;

• **Axioms**
  Displays the axioms used for reasoning tasks;

• **Strategy**
  Selects a *grounding* strategy.

Besides the above features, there are some buttons for selecting different formats for outputting data structures. Actual utterances are given to the system by text typing or menu selection, as shown in Fig. 3.2.

The interface is presented without explicit user instructions. The user has access to the vocabulary of Midas.

### 3.2.2 Input Content

The nature of the input to the system is on the basis of typed strings or menu-selected strings. The language used is English—it is planned to integrate German as well. Speech input is mimicked by adapting a fuzzy match on the words in the input string with the words in the lexicon.

### 3.2.3 Interpretation

The lexicon contains about 2700 inflected forms, based on the Autoroute corpus. A simple straightforward phrase structure rule grammar is used. Utterances can be (multiple) sentences, noun phrases, prepositional phrases, questions (yes-no and simple wh-questions), or cue-phrases (such as greetings, “yes”, “no”). On the sentence level conditionals and disjunctions are covered. There is a basic noun phrase coverage, including modification of nouns by adjectives, prepositional phrases, or relative clauses. Further, noun phrases cover proper names, mass nouns, pronouns, and possessives. Verb phrases include auxiliaries (to
Figure 3.2: Screen-shot MIDAS (input)
be and to have), modals, and to-complement verbs. Coordination is covered for basic categories such as nouns, adjectives, and verbs. Comparative constructions are not covered. Ellipsis are only possible at the utterance level (noun phrases or prepositional phrases).

For parsing a left-corner parser has been adopted, implemented in Prolog. The syntactic structure produced is a tree which guides building the semantic representations. Building semantic representations is done via proper lambda-abstraction and \( \beta \)-conversion [Bos et al., 1994, Muskens, 1996]. The semantics fragment is not completely isomorphic with the syntactic fragment. Phenomena like modal verbs are left-out in the semantics.

The semantic representations constructed are underspecified discourse representation structures [Blackburn and Bos, 1998]. These are resolved (pronouns, presuppositions, quantifier and operator scope) producing Discourse Representation Structures (DRSs) of Kamp’s Discourse Representation Theory [Kamp and Reyle, 1993]. The DRSs representing the dialogue’s information state are translated into first-order predicate logic, along the lines of [Blackburn et al., 1999]. Reasoning on these first-order representation is done using the MATHWEB distributed inference engines services [Franke and Kohlhase, 1999], including the theorem provers BLKSEM, FDPLL, and SPASS. Reasoning tasks cover consistency and entailment checks used for ambiguity resolution [Blackburn and Bos, 1998], selection of output dialogue moves, and determining proper answers for posed questions.

The framework of Discourse Representation Theory (DRT) offers direct means to model discourse and context, and MIDAS exploits this feature by coding the information state of the dialogue as a DRS. On top of this, Van der Sandt’s algorithm for anaphora and presupposition resolution is used to resolve pronouns, proper names, definite descriptions, and other presupposition triggers [Van der Sandt, 1992]. The DRS contains also dialogue-act information as proposed by Poesio & Traum [Poesio and Traum, 1998].

### 3.2.4 Input Dialogue Moves

There are two main dialogue moves: assert and query. Cue-phrases contain some specific dialogue-acts in their lexical entries, including greeting and thanks.
3.3 Output Modalities

3.3.1 Output Dialogue Moves

The main output dialogue moves are ASSERT, QUERY, REPEAT-QUERY (repeating a previously posed question, on which no proper answer is given), and CHECK (posing a check-question, as a result of the grounding process).

Further output dialogue moves include GREETING, PARDON (a move to indicate syntactic non-understanding), INCONSISTENT (a move to indicate semantic non-understanding), and CLOSING (finishing the conversation).

3.3.2 Generation

MIDAS converts dialogue moves to output content via grammar-based lexical selection and realization. The generation module in MIDAS uses the same lexicon as the interpretation module. Besides, there are some fixed expressions that are hardwired with specific dialogue moves. Most grammar rules are duplicates of the analysis grammar, but with direct encoding of the λ-DRT-semantics, based on unification of its λ-arguments (the analysis grammar assumes underspecified representations, and uses proper β-conversion rather than unification).

The generator is implemented in Prolog. It has three input parameters: a dialogue move, an utterance DRS, and a context DRS. The output parameter is a list of words or a string. On the basis of the dialogue move the generator decides the basic utterance (e.g., whether it will be a check-question, or an ordinary question), and incrementally traverses trough the DRS, and deletes information from it while applying the grammar and lexical rules. Generation of a list of words succeeds when after applying the grammar rules the utterance DRS arrives at an empty set of discourse referents and conditions.

The generator uses the context of the dialogue. Given the DRS of the total information state of a point in the dialogue, the utterance DRS is the current discourse unit under discussion, and the context DRS is the remaining DRS. The context DRS serves to generate context-sensitive expressions such as proper names or definite descriptions, which appear as free variables in the utterance DRSs, but are bound by discourse referents in the context DRS.
3.3.3 Output Content

The actual system’s output utterance is a string of words (in English). Currently, effort is made to include a speech synthesizer. Most speech synthesizers allow the string to be annotated with prosodic information, in order to improve the synthesis. The MIDAS system supports SABLE 1.0, the standard synthesis mark-up language (for documentation please consult www.bell-labs.com/project/tts/sable.html). The output utterance text can also be shown with the prosodic annotation according to the SABLE scheme. An example output-string, a check-question, illustrates these tags:

```
<SABLE><LANGUAGE ID="en">
you travel from saarbruecken to edinburgh
</LANGUAGE></SABLE>
```

Using SABLE tags is the missing link for investigating the way in which information-state revision can lead to generation of more cohesive and natural texts than is available with current technology, as discussed in TRINDI deliverable D4.1 [Engdahl et al., 1999].

3.3.4 Output Interface

The output interface of MIDAS is a combination of text messages and graphical presentation (see Fig. 3.3). Depending on the input-selections of the user, the output shown is the current content of the information state (part of it being a DRS of the dialogue), the update rules that were applied while updating the information state, the axioms used for inference tasks, and the results of the semantic analysis (see Fig. 3.4).

3.4 Grid Points

3.4.1 Input Grid Points

The input grid points of the system MIDAS cover U1 (menu selection, to select pre-stored utterances or system parameters), U2 (natural language answers to questions posed by MIDAS), and partly U4 (questions posed to MIDAS).
Figure 3.3: Screen-shot MIDAS (output)
Figure 3.4: Screen-shot MIDAS (output)
3.4.2 Output Grid Points

The output modalities of MIDAS fall within S3 (generated language appropriate to the current state of the dialogue).
Chapter 4

SRI Autoroute Dialogue Demonstrator

4.1 Overview

The SRI Autoroute Dialogue Demonstrator actually consists of three different variants. All three are directed at the same task of obtaining a route for a user, using natural language input and a freely available route-planning service on the web. Two of the variants are based on the notion of Conversational Game (for the theory of this see [Traum et al., 1999]). The third variant is directed towards another goal of the Trindi project, that of robust spoken dialogue interpretation (see [Milward, 2000]).

The sections below discuss how the variants differ in their input and output, interpretation, generation, and according to the grid of interaction.

4.2 Input and Output

The example route-planning application functions as a “natural-language in” and “web page out” interface. The following screen display shows the basis of the interface used by all three variants. The application is loaded by calling up an appropriate web-address. The server then delivers a page containing the initial system utterance and a text-box seeking user input as shown in Figure 4.1.

Input can be typed, or via any speech recogniser which allows input into an HTML text
box. The current recogniser is a generic dictation system. This needs training to particular
speakers, and the word error rate is higher than you would expect for a recogniser trained
to the domain. However, when combined with the robust interpreter described below it
performs surprisingly well, especially for longer utterances. The reason is that, although
it makes errors, they rarely affect the interpretation process. An incorrect hypothesis for a
word can be any word from the 200,000 word vocabulary, and this is less likely to confuse
the interpreter than the errors you get from a domain specific recogniser where the choice
of candidate words is smaller. For example, in a recognisers trained for the similar ATIS
(Air Traffic Information Service) domain you commonly get errors such as ‘to’ being chosen
rather than ‘from’.

After speaking or typing into the text box, the user signals the end of his turn by clicking
on “submit” or by hitting the return key. The system processes the contents of the text
box and delivers a further web page containing another system utterance and another text
box for user input. There is also a graphical display which is used to show the current
state of play i.e. where the user wants to leave from, where the user wants to go, and the
arrival time or departure time.

4.2.1 IO for a Dialogue Systems based on Conversational Games

An example next page for the Conversational Games system is shown in Figure 4.2. In
this system the graphical display is not intended to influence the progress of the dialogue:
you could just as easily imagine the dialogues occurring over a phone. In this example
the dialogue strategy is conservative, with confirmations required for each new piece of
information.
Until a Conversational Game has finished, no new propositions have been agreed and so the contents of the graphical display remain unaltered. A sample display following the closing of the game is shown in Figure 4.3.

Finally, at the close of the dialogue, the system displays a route as returned to it by public route planning service (Figure 4.4). No text-box is provided for the user to provide further input.
Dialogue history

1. Where do you want to go to?
2. from london to cambridge
3. You are starting from london. Is that correct?
4. yes
5. You are going to cambridge. Is that correct?
6. yes
7. Okay.
8. When do you want to travel?
9. seven
10. You will start your journey at 7:0. Is that correct?
11. yes
12. Okay.
13. Do you want the quickest journey or the shortest journey?
14. quickest
15. You want the quickest journey. Is that correct?
16. the quickest
17. Okay.

Figure 4.4: Conversational Games Demonstrator (3)
4.2.2 IO for an Educational System for Conversational Games

The Conversational Games system described so far functions as an example of a natural language based route-planning application as it might appear to a user, regardless of any theory or implementational details underlying it. A variant of this system functions as an educational or tutorial aid for the notion and implementation of Conversational Games themselves. This allows users to provide input into how the Conversational Games dialogue manager decides on interpretations to make and actions to undertake.

The additional interactions are designed to enable a user to understand something of how the system arrives at the moves it makes. For example, the very first interaction shown in Figure 4.5 consists not in the system issuing an utterance and offering a text box for a reply, but in the system offering a choice of what move to make first.

![Diagram](image)

Figure 4.5: Conversational Games Tutorial (1)

In this example, the system is considering four possible moves made available by the current game grammars. The moves are ordered according to a preference value decided by the system itself. The user however can select a dispreferred move by clicking on it. We suppose the user selects the most preferred option, resulting in the move which is shown in Figure 4.6.

Whenever any move is made, the Conversational Games dialogue parser analyses it. This includes analysing the system’s own utterances. Consequently in Figure 4.6, even though the user has just instructed the system to make a qw move, the system still analyses it. This is because the system cannot be sure that an intended move will be correctly interpreted. Consequently, a set of possible analyses is generated. The set includes an analysis in which the system’s own utterance is deemed unrecognizable and one in which it is deemed
unimportant (presaging an interruption). Again, the set is ordered by preference values but the user can select any of the possibilities by clicking with the mouse.

If the user instructs the system to accept its most preferred analysis (that it did indeed issue a qw move), then the user will be prompted with a text box to make a move himself, just as in the 'basic' demonstrator.

Figure 4.6: Conversational Games Tutorial (2)

Figure 4.7: Conversational Games Tutorial (3)
There is one further ‘additional interaction’ with the user in the Tutorial demonstrator. The user is offered a view of all the system’s analyses of the user’s last utterance and the chance to choose a different analysis than that preferred by the system. This choice is illustrated in Figure 4.7.

4.2.3 IO for a Robust Interpreter for Spoken Dialogues

In the current implementation of the robust system, user utterances are accepted without confirmation. This means that the graphical display has a more crucial role, serving as the primary means of feedback to the user. The user is expected to look at the graphical display and make any necessary corrections e.g. “not Leicester, Chester”.

Figure 4.8: Screen Shot for Robust Demonstrator

An example dialogue interaction is given in Figure 4.8. The interaction can be explained as follows:
1. The system asks an initial question
2. The user replies, satisfying the question but also supplying the origin for the journey
3. The system reevaluates which question to ask. Since the origin has already been established, it now asks for the arrival time.
4. The user realises from the graphical display that they have specified the wrong origin, so corrects it to Chester.
5. The system corrects the origin, and asks for the arrival time again
6. The user specifies what departure time is required
7. The system considers the information it has, and decides it does not need the arrival time since it has the departure time. It now asks whether a quick or short route is preferred
8. The user specifies a quick route, and the route is calculated

4.3 Interpreter

The interpreter used in the Dialogue Games systems is a relatively simple phrase-spotter, and provides a useful baseline with which to compare the robust interpretation system. The two interpreters are not quite interchangeable since the Games approach requires each utterance to provide a single move, whereas the output of the robust interpreter can be an arbitrary set of moves. Note that the Games approach does allow a move to comprise more than one simple move e.g. we can define a complex move equivalent to two assertions. However such complex moves need to be defined as part of a Game, they cannot be created on the fly.

4.3.1 A Simple Phrase Spotter

This interpreter implements a function from strings to propositional contents. It is a simple elaboration of a phrase-spotter. It employs a top-down left-to-right algorithm searching for a string of substrings covering the total input. Each substring is either a token which is ignored semantically or is a substring which contributes one or more propositions to the meaning of the input string. The string *I want the quickest route to get me to cambridge by four p m* might be divided up thus
(I) (want) (the) (quickest) (route) (to) (get) (me) (to cambridge by four p m).

Of these substrings, *quickest* generates the proposition *tripmode*(*quick*), and *to cambridge by four p m* generates *endtime*(16:0) and *to*(*cambridge*). All the other substrings – *I, want* etc. – are ignored.

The input is scanned left-to-right for an initial substring which can contribute a proposition. If one is found, we recurse on the remainder of the input string. Otherwise, the first word is discarded and then we recurse. When looking for an initial substring, we look for more informative substrings first. For instance, one category of substring is simply *clock-time* (e.g. *four p m*). Another category is *place* (e.g. *cambridge*). Each substring can contribute propositional information. In the absence of other information, places are assumed to be destinations and times are assumed to be departure times. However, the covering substring *to cambridge by four p m* removes the need for such default inference (the presence of *to* indicates that cambridge is the *destination*).

The algorithm is also dialogue-sensitive in that it knows the current propositions under discussion when it parses the next user input. This enables it to interpret fragmentary or elliptical input in the current dialogue context, i.e. when a default inference is required.

### 4.3.2 A Robust Interpreter

The robust interpreter is designed to incorporate benefits from both keyword spotting and full semantics approaches. A detailed description of the interpretation process can be found in [Milward, 2000].

Given an input utterance, the system builds a semantic chart using a fully left-to-right parser. The parser is fully incremental in the sense that it connects up pieces of syntactic structure as soon as possible, for example, subjects of verbs are connected to the verb before an object is encountered. There is no requirement that a single edge must be found which spans the whole utterance, and there is no attempt to ‘glue’ fragments together to form a sentence.

The semantic representation provided by the chart can be regarded as a particular kind of ‘flat structure’. A standard recursive representation of the semantics of *John believes Jack runs* would be $\text{believe}(\text{john}, \text{run}(\text{jack}))$. We can create a flat representation by first indexing each level of the structure i.e. $\text{believe}_{12}(\text{john}_{i3}, \text{run}_{i5}(\text{jack}_{i6}))_{i1}$, and then splitting up the structure into a set of constraint i.e.

\[
i1:i2(i3,i4), i2:believe, i3:john, i4:i5(i6), i5:run, i6:jack
\]
The representation provides the morphemes, believe, john, run and jack together with a number of constraints specifying their relationship, but all in a ‘flat’ list. We have only changed how we represent the semantics: there is a one-to-one mapping between the set of constraints and the original recursive representation, and the basic flat structure can be regarded as an alternative notation, or as a description of the original semantic representation. The interest in flat structures here is motivated by the distributed nature of the representation. There is a better chance of achieving robustness when dealing with fragmentary input, or in making rules (e.g. mapping to database slot-values) sensitive to other parts of a representation. For example, suppose an analysis produces three separate fragments P, a, and Q(b, c). Suppose this information is represented by

\[ i2:P, i3:a, i4:i5(i6,i7), i5:Q, i6:b, i7:c \]

This set is a subset of the flat structured representations for \( P(a, Q(b,c)) \),

\[ i1:i2(i3,i4). i2:P, i3:a, i4:i5(i6,i7), i5:Q, i6:b, i7:c \]

with only one constraint, \( i1:i2(i3,i4) \) missing. The change of representation has made the semantics of fragments and full constituents more similar and this enables a uniform use of rules which can apply to fragments and full constituents. It becomes easier to understand how to relax a ‘full constituent rule’ in order to generate a rule appropriate for fragmentary interpretation.

Once a chart has been constructed, a set of translation rules are applied to it in order to generate the meaning of the input utterances. The rules function as constraint-based pattern-matchers over edges in the chart, where the patterns include constraints over the semantic annotations on the chart edges (both edges whose content is “used up” by the translation process – ‘translated terms’, and edges which function purely as context – ‘rest of utterance’), over knowledge carried forward from the prior utterance (e.g. the last question asked) and over knowledge carried forward from prior dialogue (e.g. that we believe we already know what the departure time is). The result of applying a rule is a new set of constraints.

Below we illustrate a (simplified) rule designed for picking out the departure time where the question asked what the departure time was, and the answer contains a time.

Constraints for translated terms: \{L:T, time(T)\}
Constraints for rest of utterance: {}  
Constraints from prior utterance: \{M:query(departure\_time)\}
Contextual slot values: _
New constraints: \{Q:departure\_time=T\}

The present algorithm applies each translation rule to the chart, and picks up a set of potential slot-value pairs (for the slots, destination, origin, mode, and arrival/departure time). Each slot-value pair is weighted according to the specificity of the rule used. The weighting is currently pretty crude, and achieved by adding the number of items mentioned in a rule, with higher weightings for closer items, and higher weighting for items found in the rest of the utterance than in the context.

After creating the set of potential slot-values, the algorithm then filters the set to obtain a consistent set of slot-value pairs. The first stage is to filter out any cases where the translated material overlaps. In these cases the more specific translation is retained, the others are dropped. Secondly there is a check to ensure that no slot is given the same value twice. If there is a conflict the higher weighted value is adopted.

The effect of the weighting and filtering is to prefer more specific rules which use more linguistic or contextual information. Information within the utterance is weighted more than contextual information, since we want to be guided more by what the user actually said than by what we expected her to say. For example, if the user replies to the query *where do you want to go with from London*, the slot value pair origin=London is preferred over destination=London.

4.4 Generator

The generation components of the system comprise a simple template filling approach.

4.5 Summary: Position in the Grid of Interaction

4.5.1 Input Modalities

The input modalities of the three variants are given in Figures 4.9, 4.10 and 4.11.
<table>
<thead>
<tr>
<th>Input Interface</th>
<th>Text Typing / Speaking also, Button Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Content</td>
<td>Natural Language Text Message / Speech Signal also, Values from fixed set</td>
</tr>
<tr>
<td>Interpretation</td>
<td>Parsing according to a fixed grammar</td>
</tr>
<tr>
<td>Input Dialogue Moves</td>
<td>Answers to questions Repairs</td>
</tr>
</tbody>
</table>

Figure 4.9: Input Modalities for Dialogue Games Demonstrator

<table>
<thead>
<tr>
<th>Input Interface</th>
<th>Text Typing / Speaking also, Button Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Content</td>
<td>Natural Language Text Message / Speech Signal also, Values from fixed set</td>
</tr>
<tr>
<td>Interpretation</td>
<td>Parsing according to a fixed grammar</td>
</tr>
<tr>
<td>Input Dialogue Moves</td>
<td>Answers to questions Repairs</td>
</tr>
</tbody>
</table>

Figure 4.10: Input Modalities for Tutorial Demonstrator

<table>
<thead>
<tr>
<th>Input Interface</th>
<th>Text Typing / Speaking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Content</td>
<td>Natural Language Text Message / Speech Signal</td>
</tr>
<tr>
<td>Interpretation</td>
<td>Parsing according to a fixed grammar Pattern matching over constraints</td>
</tr>
<tr>
<td>Input Dialogue Moves</td>
<td>Answers to questions Repairs</td>
</tr>
</tbody>
</table>

Figure 4.11: Input Modalities for Robust Demonstrator
4.5.2 Output Modalities

The three systems share the same output modalities (Figure 4.12)

<table>
<thead>
<tr>
<th>Output Interface</th>
<th>Text Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Content</td>
<td>Graphical Layout</td>
</tr>
<tr>
<td>Generation</td>
<td>Text Messages</td>
</tr>
<tr>
<td>Output Dialogue Moves</td>
<td>Graphical Layout</td>
</tr>
<tr>
<td></td>
<td>Template Filling</td>
</tr>
<tr>
<td></td>
<td>Questions</td>
</tr>
<tr>
<td></td>
<td>Repairs</td>
</tr>
</tbody>
</table>

Figure 4.12: Output Modalities for Autoroute Demonstrator
Chapter 5

Godis

5.1 Introduction

This chapter presents the input and output components of GoDiS, including interpretation, generation, and relevant dialogue moves.

5.2 GoDiS Input Modalities

5.2.1 Interface and content

The user communicates with GoDiS through written natural language sentences that are typed using a keyboard. This can be done either in the form of Prolog commands, as shown in Figure 5.2.1, or using a www interface, as shown in Figure 5.2.1.

The GoDiS input module simply reads a string (until new-line) from the keyboard, preceded by the printing of an input prompt. The system variable “input” is then set to be the value read.
Figure 5.1: GoDiS Prolog Interface
Figure 5.2: GoDiS Web Interface
Currently the possibility of augmenting GoDiS with speech is being looked at. The easiest way of adding speech as an input modality is simply to connect GoDiS to a separate, already developed, general-purpose piece of software that transcribes continuous natural speech. This gives reasonably good results, in particular if the transcription system allows domain-specific recognition contexts.

### 5.2.2 Dialogue moves and interpretation

Interpretation in GoDiS involves the transformation of user utterances to dialogue moves that will later be used by the dialogue move engine. Five prominent input dialogue moves include, first of all, **GREET** and **QUIT**. Secondly, the user can **ANSWER** or **ASK** a question. Finally, there is the possibility that the user requests the system to repeat a question or statement, **REQREP**.

In more detail, the interpretation module in GoDiS takes a string of text, turns it into a sequence of words (a “sentence”) and produces a set of moves. The “grammar” consists of pairings between lists whose elements are words or semantically constrained variables. Semantic constraints are implemented by a set of semantic categories (**LOCATION**, **MONTH**, **TASK**, **MEANS_OF_TRANSPORT** etc.) and synonymy sets. A synonymy set is a set of words all of which are regarded as having the same meaning.

The simplest kind of lexical entry is one without variables. For example, the word “hello” is assumed to realise a **GREET** move.:

(2) input_form( [hello], greet ).

The following rule says that a phrase consisting of the word “to” followed by a phrase \( S \) constitutes an **ANSWER** move with content \( \text{to} = C \) provided that the lexical semantics of \( S \) is \( C \), and \( C \) is a **LOCATION**. The lexical semantics of a word is implemented by a coupling between a synset and a meaning; the lexical semantics of \( S \) is \( C \), provided that \( S \) is a member of a synonymy set of words with the meaning \( C \).
(3) input_form([to | S], answer(to=C)):‐lexsem(S, C), location(C).

To put it simply, the parser tries to divide the sentence into a sequence of phrases (found in the lexicon), covering as many words as possible.

(4) wordlist2moves([], []).  
  wordlist2moves(Sentence, [Move|Moves] ) :-  
    append( Phrase, Rest, Sentence ),  
    condition( lexicon: input_form(Phrase,Move) ),  
    wordlist2moves( Rest, Moves ).  
  wordlist2moves([Word|Sentence], Moves ) :-  
    \+ condition( lexicon: input_form([Word],_), ),  
    wordlist2moves( Sentence, Moves ).

First, if the sentence is empty, there are no moves. Second, if Sentence begins with a Phrase, for which there is some Move defined in the lexicon, then it also tries to find the Moves for the Rest of the Sentence. Third, if the first Word is not defined as a phrase in the lexicon, it just skips that Word, and continues processing the rest of the sentence.

5.3 GoDiS Output Modalities

5.3.1 Dialogue moves

At the other end of the dialogue system, an output move, or, alternatively, a set of output moves, are given to the generation module – this is after the dialogue move engine has updated the information state in accordance with the input moves, and, obviously, after it has subsequently selected appropriate output moves. Just as for input moves, there are a number of different possible dialogue moves for system output in GoDiS. The system is able to GREET and THANK the user, as well as ANSWER and ASK questions, and make moves to QUIT the dialogue. It can also REPEAT a previous move. In addition, there is a possibility of requesting a repetition from the user; here, the system makes a distinction between REQREP(UNDERSTANDING) and REQREP(RELEVANCE). In the former case the system was not able to comprehend the user utterance, whereas in the latter it could understand (that is, find a match in the grammar or lexicon), but not see the relevance (be unable to accomodate the user utterance).
5.3.2 Generation, content and interface

The GoDiS generation module takes a sequence (list) of moves and outputs a string. The generation lexicon is a list of pairs of move templates and strings, as in:

\[(5) \text{output} \_\text{form}( \text{greet}, \ "Welcome to the travel agency!" ) \].

In some cases, the move template contains some variable which is assumed to be instantiated when the lexicon is consulted. The lexicon will then find a string corresponding to the instantiated variable and insert it into the output string.

\[(6) \text{output} \_\text{form}( \text{answer}(X \text{= price}) \text{, price=Price}), \text{Str} \) :-
 number( Price ), number\_\_chars( Price, PriceStr ),
 append( "It will cost ", PriceStr, Str1 ),
 append( Str1, " crowns", Str ).

To realize a list of moves, the generator looks, for each move, in the lexicon for the corresponding output form (as a string), and then appends all these strings together. The output strings is appended in the same order as the moves.

System output is thus in the form of mainly pre-stored text. These text messages are presented to the user in written form; the output module takes the string in the system variable ”output” and prints it on the computer screen, preceded by the printing of an output prompt. The content of the ”output” variable is then deleted. The module also moves the contents of the system variable ”next\_moves” to the system variable ”latest\_moves”. Finally it sets the system variable ”latest\_speaker” to be the system.

\[(7) \text{output} :-
 \text{check\_condition}( \text{is\_set}( \text{output} ) ),
 \text{check\_condition}( \text{output} \$= \text{Str} ),
 \text{flag}( \text{output\_prompt}, \text{Prompt} ),
 \text{name}( \text{StrN}, \text{Str} ),
 \text{write}( \text{Prompt} ), \text{print}( \text{StrN} ), \text{nl},
 \text{check\_condition}( \text{next\_moves} \$= \text{Moves} ),
 \text{apply\_operation}( \text{unset}( \text{next\_moves} ) ),
 \text{apply\_operation}( \text{set}( \text{latest\_moves}, \text{Moves} ) ),
 \text{apply\_operation}( \text{set}( \text{latest\_speaker}, \text{sys} ) ),
 \text{apply\_operation}( \text{unset}( \text{output} ) ).\]
As was the case for input, the GoDiS output interface is in the form of written text. The next step would be to add text-to-speech synthesis, enabling the system to communicate with the user through speech.

5.4 Grid Points

Here’s an attempt at placing GoDiS in the grid:

5.4.1 Input modalities

- input interface: 4 (text typing)
- input content: 3 (natural language text message)
- interpretation: 2 (pattern matching)
- input dialogue moves: 1 (answers), 3 (questions), 4 (reparis and meta-dialogue)
- input grid points: U4

5.4.2 Output modalities

- output dialogue moves: 1 (questions), 2 (answers), 4 (repair and meta-dialogue)
- generation: 1 (lookup), 2 (template-filling)
- output content: 2 (text messages)
- output interface: 1 (text on screen)
- output grid points: S1 and a very small bit of S2 (the price/route retrieved from the database)
Chapter 6

EDIS

EDIS is the TRINDI implementation of the Poesio-Traum Theory (PTT) of dialogue. The relevant version of the theory is mainly described in [Cooper et al., 1999, Larsson et al., 1999], although the current implementation contains a number of extensions and amendments. The central concerns of PTT in this context are the treatments of grounding and discourse obligations, and EDIS attempts to recreate the theoretical approach in as consistent a manner as possible. Apart from the basic theoretical issues discussed in D1.1, a description of version 1 of EDIS appears in D3.2, and [Matheson et al., 2000] contains a description of the central aspects of version 2.

The current EDIS input and interpretation modules, and on the output side the generation and output stages, are generally very simple. The default TrindiKit code is used more or less unaltered in every case except the interpret module, which has slight differences due to some assumptions about the form of latest_moves. One currently unique aspect of the EDIS output is the use of Thistle [Calder, 1998] to display information states; this is normally used at present as a debugging tool, but we are investigating the possibility that Thistle could supply a graphical interface for interactions with the system, allowing users to select input and view output from within the Thistle window. An example IS is contained in Section 6.6 below. We currently use two ‘macros’ when running EDIS; a ‘user turn’ macro which calls the input, interpret, and update modules and a ‘system turn’ macro for select, generate, output, and update:

```
ut :-
    input,
    interpret,
    update.
```
st :-
  select,
  generate,
  output,
  update.

We shall briefly review the status of each module (apart from update and select) in turn, and then associate the various modalities with the interaction grids described in 2 above. Firstly, however, we list the dialogue acts assumed in EDIS.

### 6.1 Dialogue Acts

EDIS currently uses the following set of dialogue acts, all of which have three attributes representing the predicate (the act name), the dialogue participant, and a list of appropriate arguments:

- `acknowledge(DP,[Act])`
- `accept(DP,[Act])`
- `info_request(DP,[Q])`
- `assert(DP,[P,CL])`
- `check(DP,[P])`
- `answer(DP,[Act1,Act2])`
- `direct(DP,[P])`
- `agree(DP,[Act])`

Here DP is a dialogue participant, Act is a dialogue act identification number, Q a question, CL a confidence level, and P a proposition.

### 6.2 Input

The default input module simply prompts for typed input and sets the system variable `input` to the result. A typical interaction thus looks like this:

```
| ?- input.

$U>
```
Typing a string followed by a linefeed sets input:

| ?- input.  

$U> a\ route\ please

yes
| ?- val(input,L).


### 6.3 Interpretation

The interpretation stage is, as noted, minimally different from the default. The only real difference is that the default explicitly sets up a set of latest moves, while the EDIS version just associates the input string with whatever that string represents in the domain. It is actually possible that EDIS could now use the default as the need for the change is arguably a historical accident; however the current situation is simply that the string is first turned into a list of words as in the default interpret module:

| ?- val(input,L), string2wordlist(L,WordList).

WordList = [a,route,please] ?

This wordlist is then associated with the appropriate moves by the domain specification, which contains the following entry:

```
input_form( hup-english, [a,route,please], stackset([  
  record([pred=ack,  
    dp=c,  
    args=_{}]),  
  record([pred=answer,  
    dp=c,  
    args=_{}]),  
  record([pred=assert,  
    dp=c,  
```
\[ \text{args} = \text{stackset([record([item=routeinfo]), record([item=2])], record([pred=direct, dp=c], args=stackset([record([item=routeinfo])]))].} \]

The interpret module sets \texttt{latest\_moves} to be this stackset, which means that the input \texttt{a route please} is assumed to be an acknowledgement, an answer, and assertion, and a directive. Note that the assert act includes both a string representation of the ‘proposition’ \texttt{helpform(routeinfo)} and a number representing the ‘confidence level’ associated with the assertion.\(^1\) The empty argument slots in the acknowledge and answer acts are placeholders for information which is currently identified by the update process; we intend to develop the interpretation module further in the near future to produce more complete acts. It should also be possible to avoid the partial representation of assertions at this stage in the process by using a more powerful interpretation stage.

### 6.4 Generation

Generation is very much the mirror process of interpretation here; the selection process sets \texttt{next\_moves} to be a stackset of appropriate acts, and the generation stage simply looks up the representation of these moves in the domain module. To continue with our example, after the user’s \texttt{a route please} response, the selection process has set \texttt{next\_moves} to be:

\[ \text{stackset([record([pred=ack, dp=w], args=stackset([record([item='DU3'])]), record([pred=accept, dp=w, args=stackset([record([item='CA6'])]), record([pred=inforeq, dp=w, args=stackset([record([item=start])])])]} \]

The generate module we use is the default (except that we assume that \texttt{next\_moves} is a stackset rather than a set), which constructs the required output form by looking up

\(^1\)The representation of the proposition is debatable; we assume that – in response to the opening \texttt{how can I help?} – the contents are something like ‘the form of help I want is information on a route’. Further, as noted elsewhere, the proposition in the assertion is actually assumed to be partial, being ‘completed’ by the dialogue context.
the string representation of each act in the domain specification and using append. EDIS assumes that only the core act, the information request, has a ‘real’ representation, so we associate the acknowledge and accept acts with empty strings in the domain file:

```
output_form(hup-english,record([pred=ack,
    dp=w,
    args=_]),
  "").
output_form(hup-english,record([pred=accept,
    dp=w,
    args=_]),
  "").
output_form(hup-english,record([pred=inforeq,
    dp=w,
    args=stackset([record([item=start]))]),
  "Where would you like to start?").
```

Thus, running the generate stage results in the output system variable being set to the required string:

```
| ?- generate.

yes
| ?- val(output,L).

L = [87,104,101,114,101,32,119,111,117,108,100,32,121,111,117,32,
```

### 6.5 Output

EDIS uses the default output module, which just prints the output prompt and the string set by generation:

```
| ?- output.

$S> Where would you like to start?
```
6.6 Thistle Interface

As mentioned above, EDIS uses Thistle to display ISs. The following is an example showing the state of the dialogue following the user’s input *a route please*:

6.7 Interaction Grids

The above module descriptions mean that the EDIS interaction grids are as shown below, where the numbers refer to the list points in 2. We begin with the input:

<table>
<thead>
<tr>
<th>Input Interface</th>
<th>(4) Text Typing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Content</td>
<td>(3) Natural Language Text Message</td>
</tr>
<tr>
<td>Interpretation</td>
<td>(2) Pattern Matching from Fixed Set</td>
</tr>
<tr>
<td>Input Dialogue Moves</td>
<td>(2) Restricted Instructions to System</td>
</tr>
</tbody>
</table>
The EDIS output grid is as follows:

<table>
<thead>
<tr>
<th>Output Interface</th>
<th>(1) Text Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Content</td>
<td>(2) Text Messages</td>
</tr>
<tr>
<td>Generation</td>
<td>(3) Graphical Layout</td>
</tr>
<tr>
<td>Output Dialogue Moves</td>
<td>(4) Repairs</td>
</tr>
<tr>
<td></td>
<td>(5) Combinations of the above</td>
</tr>
</tbody>
</table>

| (1) Lookup of Pre-Stored Text |
| (1) Questions               |
| (3) Directed Actions        |
| (4) Repairs                 |
| (5) Combinations of the above |
Bibliography


