Dialogue

Dimitra Tsovaltzi
Stephan Walter
Aljoscha Burchardt

MiLCA, Saarbrücken

Abstract

Dialogue is at the same time the most fundamental and broadly used form of language, as well as the most complex one. And most of all, dialogue is the most natural medium of communication for human beings. It is these aspects of dialogue that make its modelling a research area of its own, and one of greatest interest at that. In many cases of interaction between humans and machines, using dialogue may enables better results with regard to the target each time.

In these series of lectures we are going to address the issue of dialogue and dialogue management.

In the first lecture we are going to look at some existing dialogue systems and evaluate their performance with regard to the general characteristics that a dialogue system should exhibit. We will consider different approaches to dialogue management and assess their methods of dealing with different dialogue characteristics.

In the second lecture we will have a closer look at systems based on finite state automata (FSA). We will consider their possibilities and limitations by exploring the dialogue specifications necessary and their manipulation in FSA. In the case where the system does not currently cater for all desirable dialogue properties, we will weigh the possibility of extending it.

In this lecture, we will come to more complex dialogue modeling and will discuss a more powerful systems We will talk about speech and dialogue acts, their relation and the way they are used to facilitate dialogue modeling. We will also address the issue of general system architectures to the extent that they partake of or influence dialogue management.

MiLCA, Computerlinguistik, Universität des Saarlandes, Saarbrücken, Germany

Dezember 2003

Contents

1

Examples of Dialogue Systems

1.1 Dialogue Processing

1.1.1 Warming Up

Before we look at state-of-the-art systems, have some fun and try out a classic system.

ELIZA

Eliza (to be found here[1]) is one of the first dialogue systems. You can regard it as your friend. Tell it your problems and it will try to help you. It cannot really understand what you tell it. It picks on words that it then maps onto specific output. Play with it for fun. Do try to work out how it operates!

1.1.2 General Dialogue characteristics

We'll now give some *general dialogue characteristics* that should be handled by a dialogue manager in some way or another. Besides giving insight into how dialogues work and how they're structured, these characteristics serve as beackround for the evaluation and classification of dialogue systems. We will use them when we discuss the dialogue system of the Saarbrücken speaking elevator later in this chapter, and you will also have the opportunity to apply them to further systems in your exercises.

So what are the general characteristics of dialogues that a dialogue manager should be able to handle?

1. *Turn-taking* : When, how, and for how long should each dialogue participant talk? (Example (Section 1.2.2), Elevator characteristic (Section 2.4.1)).
2. *Adjacency pairs and insertions* : What is the appropriate way of responding to a given input? How can one correct what one has said, ask for additional information in order to and before one can provide the appropriate response, etc.?(Example (Section 1.2.3), Elevator characteristic (Section 2.4.2)).
3. *Grounding* : How does a dialogue participant make sure that their contribution is rightly understood, or that they themselves have understood correctly a previous contribution? How can misinterpretations be corrected? (Examples (Section 1.2.4), Elevator characteristic (Section 2.4.3).)

[1] `http://www-ai.ijs.si/eliza/eliza.html`

4. *Dialogue context* : How do dialogue participants use the contributions and the conclusions previously drawn to interpret the current contribution and decide on their next contribution? (Example (Section 1.2.5), Elevator characteristic (Section 2.4.4).)

5. *Ellipsis* : How do dialogue participants make sense of fragmentary responses that are often provided in dialogue situations?(Example (Section 1.2.6), Elevator characteristic (Section 2.4.5).)

6. *Reference resolution* : How can participants disambiguate what referring expressions refer to? (Example (Section 1.2.7), Elevator characteristic (Section 2.4.6).)

7. *Mixed initiative* : What is the amount, quality and level of contribution of every participant? (Example (Section 1.4.7), Elevator characteristic (Section 2.4.7).)

1.2 Speaking Elevator: Examples of Required Behaviour

1.2.1 Introduction of the Speaking Elevator Sytem

The Saarbrücken computational linguistics department's[2] speaking elevator is the standard elevator in building 17.2. Should you ever come to Saarbrücken, it is a must to have a ride with it. Try to imagine what kind of thing you would expect it to do and see if you can get it to do it. For example, change your mind about the floor you have asked it to take you to and request another floor at different points in the dialogue.

Since we don't have the elevator here, here's an example of the kind of dialogue it handles (of course the original Saarbrücken elevator speaks German):

User:	Elevator.
System:	Which floor do you want?
User:	Professor Bill Barry.
System:	I'm taking you to the Fifth floor.
User:	Thank you.
	(Mechanical command execution follows).

The following examples depict dialogue behaviour of the kind that we would like the elevator to simulate. Below each example you find a link to a paragraph that discusses whether the speaking elevator at present displays the described behaviour, and if not how complicated it would be to extend its dialogue module (which we will elaborate in Section 2.3.1) accordingly.

1.2.2 Turn-taking

System:	Which floor do you want?
User:	(no input)
System:	I'm sorry I didn't understand you. Please specify a floor number, a person or a department you

[2] http://www.coli.uni-saarland.de

The elevator prompts in a more elaborate way when there is no response by the user after a pre-defined amount of time. It gives information on what kind of answer is expected.

Current state and possibility of extension: See Section 2.4.2. See also description of elevator, Section 2.3.

1.2.3 Adjacency pairs and insertions

System:	Which floor do you want?
User:	Which floor is Malte Gabsdil on?
System:	Malte Gabsdil is on the first floor.
User:	Then, first floor, please.

The user requires more information in order to decide which floor they want to be taken to. The elevator provides it. When the user makes their mind up, based on the additional information, the elevator is able to understand that the last utterance is a response to the first question.

Current state and possibility of extension: See Section 2.4.2. See also description of elevator, Section 2.3.

1.2.4 Grounding

System:	Which floor do you want?
User:	Third floor.
System:	I'm taking you to the fourth floor.
User:	No, third floor.
System:	I'm taking you to the third floor.

The elevator misrecognises the requested floor. The user corrects it and the elevator takes the correction correctly into account.

Current state and possibility of extension: See Section 2.4.3. See also description of elevator, Section 2.3.

1.2.5 Dialogue context

	(user gets in on third floor)
System:	Which floor do you want?
User:	Ground floor.
System:	I'm taking you to the ground floor.
	(trip starts)
User:	I left my bag in the office. Can we go back?
System:	Yes. I'm taking you to the third floor.

The user specifies a target floor. After the trip has started he says that he wants to go 'back'. The system interprets 'back' in context and understands that the user wants to go to the third floor, where the trip started from.

Current state and possibility of extension: See Section 2.4.4. See also description of elevator, Section 2.3.

1.2.6 Ellipsis

System:	Which floor do you want?
User:	Fifth.
System:	I'm taking you to the fifth floor.

The user gives a fragmentary response. Instead of saying 'I want to go to the fifth floor.', he says 'fifth'. The elvator still understands that the fifth floor is where the user wants to go.

Current state and possibility of extension: See Section 2.4.5. See also description of elevator, Section 2.3.

1.2.7 Reference resolution

System:	Which floor do you want?
User:	The top floor.
System:	I'm taking you to the 5th floor.

The elevator does not give a floor number but a referring expression for it. The elevator is able to resolve the referring expression and realise that 'The top floor' is the fifth floor.

System:	Which floor do you want?
User:	Phonetics
System:	Do you want the 4th or the 5th floor?
User:	The fifth floor, please.

The same here. The system resolves 'Phonetics' to be referring to either the fourth or the fifth floor. Being aware of the ambiguity, it requests a clarification.

Current state and possibility of extension: See Section 2.4.6. See also description of elevator, Section 2.3.

1.3.2 The domain

The basic dialogue scenario that *TRAINS* deals with is route-planning in a transportation domain. The user sees a display of a map with cities and their train connections. He is asked to solve logistics problems, taking into account environmental factors as well as the standard connections. Both spoken and typed communication is possible.

1.3.3 A TRAINS example dialogue

User[1]:	I need to get to Detroit, Syracuse and Montreal.
Syst[2]:	OK. [highlights the three cities]
User[3]:	Let's send the trains from Chicago to Detroit.
Syst[4]:	Ok. [draws route from Chicago to Detroit via Toledo] But the terminal at Toledo is delaying
User[5]:	That's ok. Now let's go from Atlanta up to Syracuse.
Syst[6]:	Please tell me a route to use to get from Atlanta to Syracuse [highlights engine at Atlanta and
User[7]:	Go via Charleston and Pittsburgh.
Syst[8]:	Ok. [draws route via Cincinnati] But traffic through Cincinnati is being delayed due to constr
User[9]:	Ok, let's go via Charleston and Buffalo instead.
Syst[10]:	No problem. [draws route]
User[11]:	That's great.
Syst[12]:	I hope it was helpful.
User[13]:	Finally, I want to go from Montreal to Philadelphia.
Syst[14]:	Ok. [draws route]
User[15]:	That's great. I'm done.

1.3.4 The mode of interaction

The dialogue starts by either the system presenting the user with a problem, or the user presenting the system with their goal. The system was built with the objective of allowing the user and the system to collaborate towards solving the problem in a way that humans collaborate with each other. The participants have to define tasks in order to solve the original problem and talk about them. Interactions are interpreted with reference to all previous interactions.

1.3.5 Planning

Each participant is responsible for the part of the task that they can perform better. That means that the user is responsible for the top level goals of how to attack a problem. The system constantly tries to infer from the user's input what the user's goals are, with respect to the task. Based on that, it draws the user's attention to possible problems and makes relevant suggestions. Due to the aimed behaviour for the system, a lot of planning and plan recognition becomes necessary. The TRAINS system uses domain-specific reasoning techniques to make plan recognition less computationally expensive. As far as planning is concerned, the system is deliberately weak, so that more collaboration with the human user is provoked.

1.3.6 How the user input is interpreted

The input by the user is interpreted and the objectives (or speech acts (page 25)) that are attempted in every turn are assigned to them. The assignment is done by means of a hierarchy of speech acts. The system also makes use of the dialogue context, including references to objects or previous states, and domain reasoning to disambiguate user input and infer their objectives. Clarification dialogues can also be initiated by the system in case the input cannot be disambiguated or if additional information is judged necessary before the system itself can do any reasoning.

1.3.7 Dealing with the input

The system confirms understanding. The appropriate parts in the map displayed are highlighted each time the user defines a goal. The system tries to give useful feedback in the context of what the user has suggested as a solution.

1.4 TRAINS General Dialogue Characteristics

1.4.1 Turn-taking

Because of the nature of the domain, the mode of interaction seems to be based on the user suggesting something and the system responding to that. The content of each turn is not restricted to some expected input. The system receives the input, whatever that is, and subsequently tries to interpret it and reason about it. However, it is not clear in the documentation how turn-taking is defined.

See TRAINS dialogue example (Section 1.3.3); The user makes suggestions and the system, invariably responds to them, albeit, in a sophisticated way.

1.4.2 Adjacency pairs and insertions

The system has an elaborate way of dealing with adjacency pairs by use of the theory of discourse obligations, whose role is to define expectations as to the intentions that a participant should form and aim at realising. Sub-dialogues can be handled as well. The hierarchical representation of goals allows for such behaviour. When the user initiates a sub-dialogue, the goal behind it will be inferred by the system and treated accordingly. That is, the sub-dialogue will be represented as a sub-goal.

See TRAINS dialogue example (Section 1.3.3); In [3] the user makes a suggestion, in [4] the system acknowledges the suggestion first (‘OK’) and then points out the problematic issue, which is a sub-dialogue.

1.4.3 Grounding

There are two modules in TRAINS, the post-parser and the chart parser, that are involved in making sure that the input is correctly understood and interpreted. The system offers understanding confirmation. It displays the relevant domain objects whose employment the user has asked for, as well as the ones that the system judges need to be involved in order to achieve the user’s goal. That gives the opportunity to the user to intervene in case of wrongly interpreted input. Since the system is highly involved into solving the problem and tries to make relevant suggestions, any wrong interpretation would soon become apparent to the user, anyway.

See TRAINS dialogue example (Section 1.3.3); In [7] the user makes a suggestion, in [8] the system shows that it understands the suggestion (‘OK’). Then it criticises the suggestion based on what it thinks the goal is, i.e., that the transport should rather be faster than slower. In [9] the user changes the original suggestion due to the criticism.

1.4.4 Dialogue context

TRAINS takes dialogue context into account in different ways. A module, the verbal reasoner, interprets speech acts in context. In order to do that it communicates with other resources in an attempt to have a clear picture of the state of the discourse. The interpretation is based on the representation of beliefs, goals and intentions that the two participants have. Domain reasoning is also involved in the contextual interpretation. The problem solver in collaboration with the domain reasoner try to figure out how the user's suggestions to tackle the task problem fit into the more general goals already recognised.

See TRAINS dialogue example (Section 1.3.3); As we have already seen, in [8] the system criticises the suggestion based on what it thinks the goal is, i.e., that the transport should rather be faster than slower. That goal has been inferred by reasoning about the domain. In [9] the user takes the advice and offers an alternative solution. Given the dialogue context thus far, the system realises in [10] that Buffalo is requested instead of Cincinnati. It correctly deletes the Cincinnati route and draws the new one.

1.4.5 Ellipsis

The parser is equipped with a set of *monitors* whose job is to recover from eliptical utterances a partial representation of what the intended objective of the fragmentary utterance might have been. This has the additional advantage of compensating for misrecognitions or failures from the speech recogniser.

1.4.6 Reference resolution

There is a whole module which is responsible for resolving references. It either maps definite references to objects in the domain, or it gives out a request for a reference clarification at a later point. See TRAINS dialogue example (Section 1.3.3); In [3] the user refers to 'the trains from Chicago'. The system in [4] maps that onto some object in the domain that matches the description and represents it in the map display.

1.4.7 Mixed initiative

The initiative is shared between the dialogue participants. It cannot be said, however, to be freely mixed, as there are constraints as to the responsibilities each agent is better capable to meet. Therefore, the human user keeps the initiative as to the higher goals involved in the task problem. The system takes care of more bottom level details involved in realising those goals. Although it cannot make original suggestions on the higher level, it can point to the need of a goal shift, if it judges that there might be a problem based on environmental constraints. See TRAINS dialogue example (Section 1.3.3); In [5] the user defines an underspecified goal. In [6] the system identifies the goal partly, recognises the underspecification, and asks for further specification before it can deal with the details.}

1.5 TRAINS References

- [3]. Download here.[3]

[3] `allen-et-al-acl96.pdf.gz`

- [8]. Download here.[4]

[4] ferguson-allen-miller-aips96.ps.gz

2

Finite States Techniques for Dialogue Processing

2.1 FSA-based Dialogue Systems

2.1.1 Processing dialogue with finite state systems

When modeling a dialogue with an *automaton* , the key idea is to think of the states of that automaton as standing for different states of the dialogue, and of its edges as corresponding to things that happen in the dialogue.

So *states* are always defined with certain expectations as to what the system can have had as input, and what else can have happened at certain stages of the dialogue. For instance, the initial state of a dialogue automaton is naturally the beginning of the dialogue, and final states will normally correspond to possible end-points of a dialogue.

Edges (also known as *transitions*) of a dialogue automaton may for instance be defined to take the user input into account in order to decide what the new system state is, and to activate certain procedures, for instance output or movements. That way whenever a particular input is recognised by the system, a predefined behaviour and the way to realise it can be produced next.

So one adaption that has to be made to use *FSA* for dialogue processing is to allow various kinds of actions to be connected to edges, other than just consuming given input. Although this step is formally quite far-reaching, it is conceptually simple. For the practical purposes we're going to look at, the striking simplicity of FSAs remains untouched.

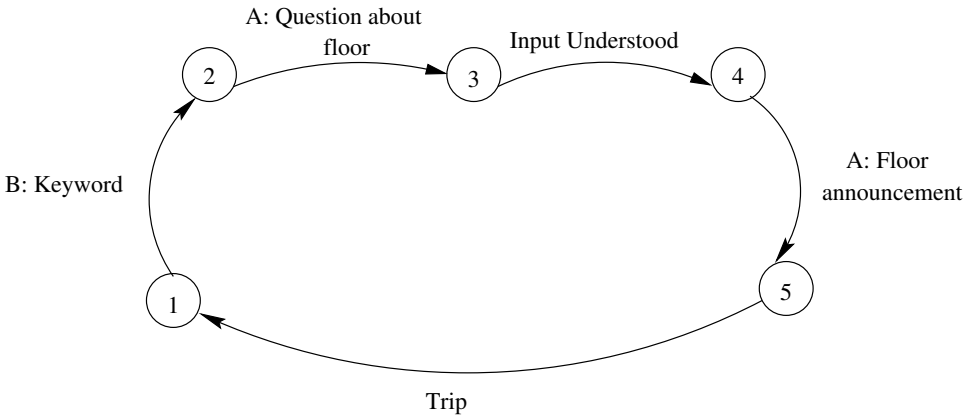
2.1.2 A simple FSA example

The dialogue module of the speaking elevator is modeled by a finite state automaton. It handles input and output. For this purpose it communicates with a speech recogniser and a text to speech synthesiser. Specific kinds of spoken input are expected according to the current state. Besides that, it also communicates with the hardware and controls mechanical actions.

As a first example let us look at a FSA dialogue manager that could be used to produce very simple elevator dialogues. As we will soon see, the real speaking elevator's FSA is somewhat more complex. Still it is based on the same structure. The dialogue module

of the speaking elevator handles input and output and it controls mechanical actions. It communicates with a speech recogniser and a text to speech synthesiser. Specific kinds of spoken input are expected according to the current state. It also communicates with the hardware. So a very simple FSA for elevator dialogues consists of five states and five arcs and looks as follows:

A simple finite state automaton



State 1 is the initial state. The system (A) expects a particular keyword from the user (B). When the keyword is recognised, the system is set to State 2. A question about which floor the user wants to go to is produced, which sets the system to state 3. The user's input is recognised, which sets the system to state 4. The system informs the user of the floor he is being taken to and the state after that is State 5. After State 5, the trip begins and the system is reset.

From this picture we can see another difference between finite state systems used in dialogue processing and FSAs in the strict sense as we have seen them in previous lectures: Dialogue automata don't always have final states. This is of course because dialogues systems - like for instance in an elevator - often need to be ready for new input immediately when one dialogue is over, and shouldn't be 'switched off' after one interactive sequence. This can be symbolised by simply looping back to the intial state. In the above picture, we could just as well regard state 5 as final state and view the trip-and-reset as external to the automaton.

Example dialogue

Our simple automaton is able to generate only dialogues of the following kind.

User:	Elevator.
System:	Which floor do you want?
User:	Professor Bill Barry.
System:	I'm taking you to the Fifth floor.
	(Mechanical command execution follows)

One thing that this dialogue does not allow the user is to confirm or correct the system in case of a misrecognition. We will now look at various ways of mending this shortcoming.

2.2 Extending FSA

2.2.1 Why Extending FSA?

The kind of dialogue that the automaton we've just seen allows is only an abstraction of the expected dialogues. It covers so few of the characteristics that make dialogue an efficient way of communicating that it almost defeats the purpose of using dialogue at all. A drop-down menu would probably be equally efficient.

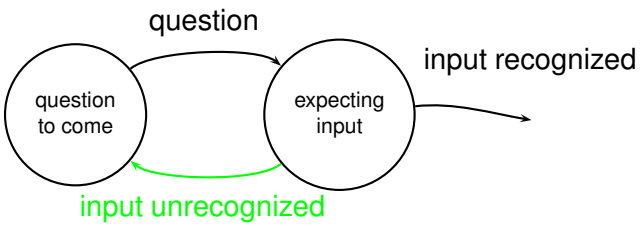
Let us see, then, what a programmer has to do in order to extend a FSA dialogue manager in order to encapsulate the dialogue characteristics considered in Section 1.1.2. We will do that by looking at an example, namely, handling grounding.

So let's look at the possibilities of extending the simple automaton in Section 2.1.2 to cover different forms of grounding. Due to the low performance of the state of the art speech recognisers, it is absolutely necessary for a system to model this feature in order to prevent irrecoverable errors.

2.2.2 Grounding Extension 1

An obvious solution for modeling grounding would be to add a backward-transition from every state where input is expected, pointing to the state before it, where the corresponding question is to be produced. This new edge should then be taken when the input given by the user is not understood. Here's the general picture:

Adding a transition



The resulting dialogue would be as follows:

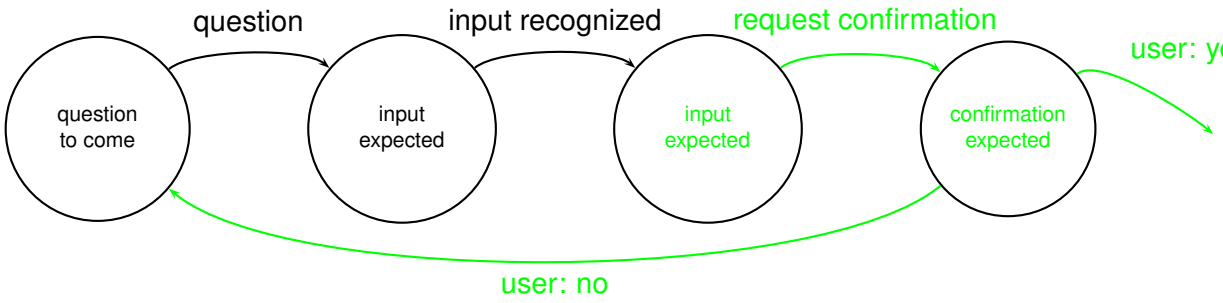
[User:	Elevator.]
System:	Which floor do you want?
User:	Professor Bill Barry.
(input unrecognised)	
System:	Which floor do you want?
User:	Professor Bill Barry.
(input unrecognised)	
System:	Which floor do you want?
...	...

This is a likely result, as the user cannot know why the failure to recognise the input occurred in the first place.

2.2.3 Grounding Extension 2

The grounding method we've just seen covers (in a very trivial way) the case that the system doesn't recognize the user's input at all. Yet even if the system does recognize the user's input, grounding is important to make sure that the input has been understood correctly. For this, one solution is to add extra states to every state that expects input. Those states are responsible for asking for explicit or implicit confirmation based on what the proposed recognition is.

Adding states



The resulting dialogues would be as follows:

[User:	Elevator.]
System:	Which floor do you want?
User:	Professor Bill Barry.
System:	Did you say professor Barry?
User:	Yes
System:	[Trip...]

Or:

[User:	Elevator.]
System:	Which floor do you want?
User:	Professor Bill Barry.
System:	Did you say professor Barry?
User:	No
System:	Which floor do you want?

It is of course quite aggravating for the user to have to confirm what he has said every time he says something. That would amount to almost double the number of turns that humans need to hold a dialogue with the same outcome. If the number of the original questions that the system needs to ask is more than a couple, which is true for most domains, this becomes a significant problem.

2.2.4 Grounding Extension 3

Delayed Confirmation

A third option for systems that request a lot of information from the user is to have delayed implicit requests for confirmation. That means adding for some states other states that asks the user to confirm all the information that the system has elicited in the previous states. This method avoids prolonging dialogues too much.

A dialogue with such a system might look as follows:

User:	Elevator.
System:	Please specify the city for which you want a weather forecast.
User:	Saarbrücken.
System:	OK, For when?
User:	Tomorrow afternoon.
System:	So, you want a weather forecast for Saarbrücken for tomorrow afternoon.
User:	Okay.
System:	[...]

Or:

User:	Elevator.
System:	Please specify the city for which you want a weather forecast.
User:	Saarbrücken.
System:	OK, For when?
User:	Tomorrow afternoon.
System:	So, you want a weather forecast for Stuttgart for tomorrow afternoon.
User:	No, that's wrong.
System:	Please specify the city for which you want a weather forecast.

The system first collects some user input and then requires confirmation for all the collected information at once. This method is already quite advanced, since it requires some kind of way for storing all of the previous answers recognised before the confirmation is done.

2.2.5 And what about Repair?

The last two examples involve defining a lot of states and transitions. Yet note again that they do not take into account the case where the input is not understood at all. In order to do that, we could first implement the first method we saw in Section 2.2.2. That is, adding a transition to return to the state before the corresponding question if the input is not understood.

Even this combination of methods would not allow the user to correct the input directly after the request for confirmation. If the information gathered by the system is inadequate, the user has to go through answering all questions again. Instead it would be nice to allow for more targeted corrections by the user, for example corrections like:

	...
System:	So, you want a weather forecast for Stuttgart for tomorrow afternoon.
User:	No, I said Saarbrücken.
System:	So, you want a weather forecast for Saarbrücken for tomorrow afternoon.

Or:

	...
System:	So, you want a weather forecast for Saarbrücken for tomorrow morning.
User:	Wrong time.
System:	So, you want a weather forecast for Saarbrücken for tomorrow afternoon.
System:	OK, For when?
	...

To allow for this kind of dialogues, we have to add still more states and transitions, to distinguish between different kinds of user-reactions to the system's confirmation request and repeat different questions based on what recognition errors the user points out.

?- Question!

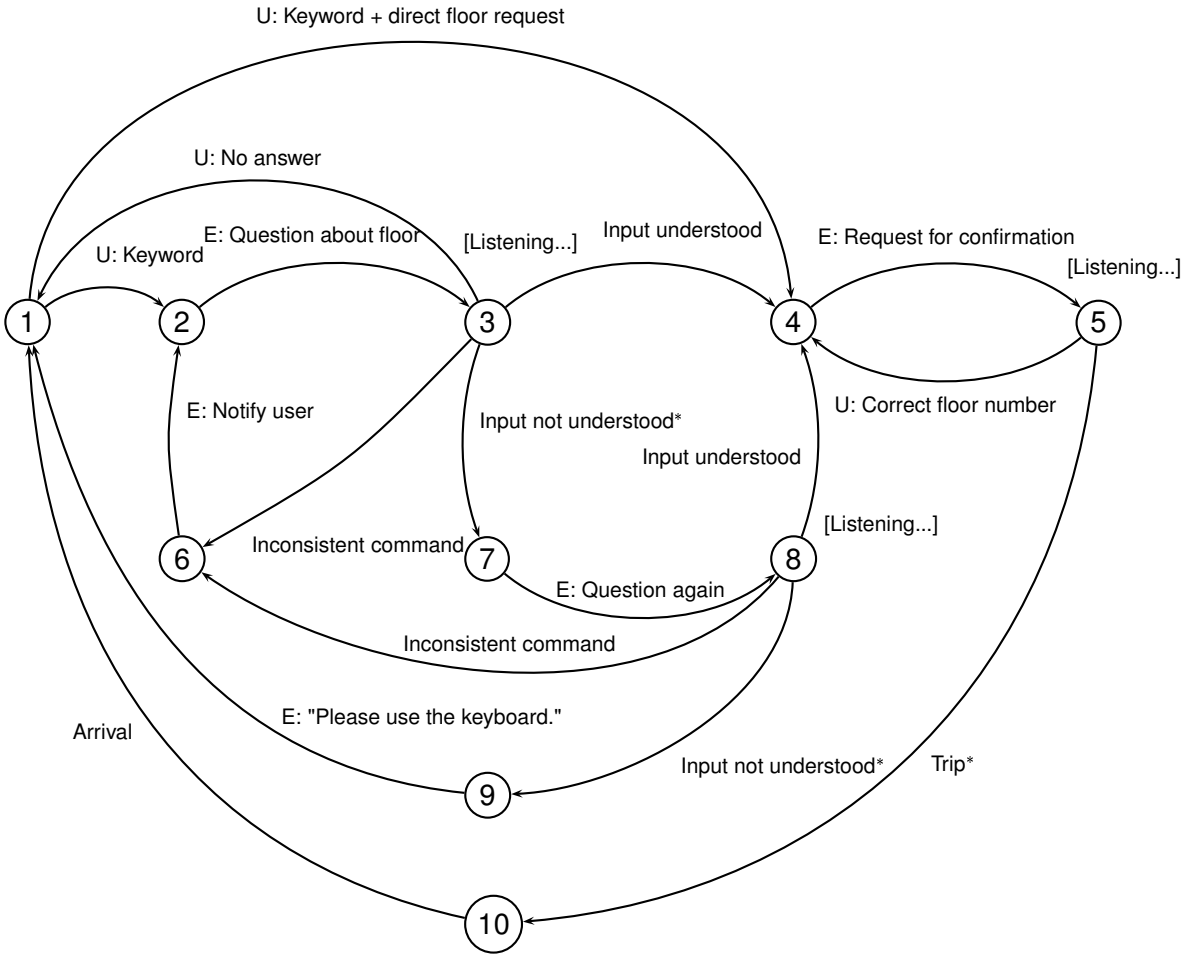
What about including insertions, for example clarification sub-dialogues, and still allowing for grounding within them in the manner described above?

2.3 An In-depth Look at the Speaking Elevator

2.3.1 The Saarbrücken CL department's Speaking Elevator

Now let's have a closer look at the dialogue engine of the Saarbrücken CL department's speaking elevator. In the following we will assume that the automaton at the heart of the speaking elevator looks like this:

Elevator dialogue finite state automaton



?- Question!

Compare this automaton to the simple one in Section 2.1.2. Do you recognize the additions that correspond to the grounding techniques we've just discussed?

See movie in HTML version.

Try different trips interactively!

2.3.2 How does the dialogue progress?

The initiation of the dialogue occurs via a keyword, namely, either 'Fahrstuhl' or 'Aufzug' (which of course means 'elevator', or 'lift'), that the user has to be aware of and utter. There is also the possibility to skip the second and third state and go directly to state 4, if the user uses the keyword and makes a travel request at the same time. This possibility is defined separately from the single keywords in the recognition grammar. It is only an added feature for quicker interaction for the people familiar with the system.

When one of the keywords is recognised, the elevator asks in German 'Which floor do you want to go to?', which corresponds to transition 2–>3 in the model (page 17). By that, the system is set to the state of expecting an answer, which corresponds to state 3. The answers are all in the recognition grammar of the speech recognizer and they include numbers of floors as well as descriptions that can be mapped onto floor numbers, e.g. names of professors that sit on that floor. Turns are strictly defined: The elevator prompts for information, the user answers, the elevator prompts again, and so on.

2.3.3 An additional memory

There is a memory that is used for storing global variable values, which are accessible from all states. One thing such variables do is providing a mechanism to have *parametrized* actions connected to edges instead of fully instantiated ones. This extends the possibility of things that can happen over one and the same transition from one state to the next. So that way, the memory allows defining less states explicitly, as the actual arguments are defined via global variables instead of states. For instance we will see that there is no need to add explicit states for every different target floor in order to produce correct confirmation sentences. The floor requested is looked up in the memory and is added to the standard confirmation utterance.

The memory is also used for various other purposes that we will soon come across. It is important for the time management. The request of the selection function to the speech recogniser must be assigned a point in time when a relevant utterance from the user is to be expected. Relevant here means, an utterance that matches one of the arguments that the selection function is expecting. And the memory also adds dialogue context by keeping track of already recognised targets, which floors the elevator has been on, and the current floor. The global memory is reset when the automaton reaches the starting state again at the end of a trip or when it has given up understanding the user.

2.3.4 How long does the system wait for a response?

The system waits for responses for a pre-defined specific amount of time (this happens at states 3, 5, and 8. We have indicated such 'time windows' in the diagram (page 17) by writing [Listening...]). If there is no answer within the time limit at a state with a 'time window', the system carries out a specified default action. We indicate default actions by marking the corresponding edge label with a little asterisk *.

So for instance at state 3 the system waits for user input for some time. If, however, there is no input within that time, the system goes to state 7 because that is the default action. From there it moves to state 8 and asks for a floor specification again. If there

is no response in due time after that second prompt in a row, the elevator asks the user to use the keyboard and it is reset to the starting point.

2.3.5 How does the elevator deal with unrecognised utterances or inconsistent input?

If the user utterance is not understood, a transition is chosen which corresponds to the elevator stating so, 'I have not understood.'. If the second try by the user is still not understood, there is a request that the user uses the buttons instead, and the elevator is reset. (Since the *Input not understood*-edges are marked as default, this is of course exactly the same course of things that we just discussed for time-outs at the 'time windows' at states 3 and 8).

Due to its global memory, the elevator also has a modest possibilty to check the (understood) user input for consistency. It compares the floor number that it understood from the request to that of the floor it's on, which it finds in its memory. If these are the same, the automaton (page 17) moves to state 6, then to back to state 2 and informs the user: 'We are already on floor *x*'. Being in state 2, the elevator asks for instructions again.

2.3.6 And what happens if the elevator does understand the input?

If however the user's utterance is understood at state 3 (that is, if a floor number is understood that makes sense as destination floor), the elevator (page 17) moves to state 4. From there a transition follows for the confirmation of the floor understood. After this, the elevator is in state 5. If then the user corrects the elevator, we are back at state 4 (confirmation to come). This can end up in an infinite loop.

?- Question!

Can you trace the loop in the dialogue model (page 17)?

2.3.7 How is the user input confirmed?

User input is confirmed indirectly. The elevator informs the user of the floor it is about to start traveling to. If there is no correction or if the input is 'Thank you', the elevator starts the trip.

2.3.8 During the trip...

The dialogue cannot continue during or after the trip to the requested floor. It has to start from state 1 again.

When the trip starts, the automaton (page 17) moves to state 10 and waits until the mechanical command has been executed. The user is then informed that they have arrived at the particular floor (transition 10–>1). This transition resets the automaton to the starting state.

2.3.9 What happens when there is ambiguous input?

The Saarbrücken Phonetics department is in the same building as the CL department. They use the same elevator, and because they are on floors 4 and 5, they probably use it quite often. Now someone might ask the elevator: 'Take me to the phonetics department!' If there is such an ambiguous request, the elevator should ask a clarification question and be reset to state 3. For example, the elevator should ask 'Do you want to the 4th or the 5th floor?'.

?- Question!

How would our automaton have to be extended to deal with this situation?

2.3.10 Speaking elevator reference

- Download here.[1]

2.4 Dialogue Characteristic and Extension of the Elevator

2.4.1 Turn-taking

Turn-taking is strict. It has to be explicitly defined in states. The system asks the user something and waits for the user's contribution for a defined amount of time. An empty turn, where the user does not say anything, or not within the time limit, is also defined as a turn.

2.4.2 Adjacency pairs and insertions

Insertion dialogues and adjacency pairs have to be defined explicitly as possible transitions from every state. This would lead to computational explosion! Try to work out what the dialogue model (page 17) would look like if you wanted to add some clarification dialogues.

2.4.3 Grounding

Grounding in the speaking elevator (page 17) is *implicit*. Traversing the edge 4–>5, the floor requested is looked up in the system's global memory and is then mentioned in a standard confirmation utterance: 'I'm taking you to the x-th floor.', which the user can then object to.

As we've discussed above, by using an extra global memory to store the floor requested we can keep the number of states and edges needed for grounding quite small: There is no need to add explicit states for every different target floor in order to produce the correct confirmation sentence. A nice extension would be to remember the floor that was misunderstood, so that it does not recognise it wrongly twice, that is, if the user has already stated that the original recognition was wrong.

[1] `speaking-elevator.ps.gz`

Repair is also possible, although in a very limited fashion in the depicted dialogue model. It is defined as a possible transition from the state after the confirmation utterance has been produced (5–>4). The elevator (page 17) can only recognize correct floor requests over this transition. Everything else activates the *Trip* edge. That is, you can only correct it by giving alternative (and consistent) floor requests, not by saying for instance 'No!' or 'Stop!'.

?- Question!

How would the speaking elevator-automaton have to be changed to allow for more natural corrections by the user?

2.4.4 Dialogue context

Almost all dialogue context is hard-coded in the states and transitions. It is not explicitly represented, with the exception of storing the floor requested and the floor currently on. That makes it difficult to change general aspects of the dialogue. All states have to be re-defined.

2.4.5 Ellipsis

Ellipsis can be dealt with only in the very particular context of the current state. Recognition is based on keyword spotting. So, as long as elliptical phrases are present in the *recognition grammar* , they will be recognised. The recognition grammar is the same for every state. Therefore, the elevator cannot be extended by allowing a different grammar to be called for every state. There are other similar systems that do make use of that possibility.

2.4.6 Reference resolution

There is no possibility of extending the elevator to handle reference resolution apart from explicitly including the referring expressions in the recognition grammar. That can only be done for a tiny amount of referring expressions out of the, in principle, unlimited possible ones.

2.4.7 Mixed initiative

Only the system has initiative, apart from the initialisation of the interaction. Still that can only be done by one of the pre-specified keywords. If something goes wrong, the system is reset and the dialogue has to start anew, with loss of all information. The system prompts the user for a specific kind of contribution and waits for a relevant utterance in the very limited context of the current state. The user cannot provide more information that could be helpful, or underspecify a parameter required by the system, as only the utterances defined in the recognition grammar can be recognised. Users cannot even choose to answer questions still in the pre-defined way, but in an order that better fits their purpose. In addition, the system cannot reason about anything that is not hard-coded. It cannot make suggestions based on context, for example.

There is no possibility of extending the elevator for mixed initiative other than defining states that allow the system to take further initiative itself. An extension to accept

user initiative can also partially be handled in the same way. It is, however, more problematic as there is no upper limit as to what the user can say, or request. This is a general problem with allowing a high degree of user initiative - even sophisticated plan recognition techniques are faced with it.

A fairly easy extension would be for the system to inform the user of the options they have at each point, so that the dialogue does not come to a break. In a way, this is modeled now by the system telling the user to use the keyboard after two misrecognitions in a row.

2.5 Summary and Outlook

2.5.1 Advantages and disadvantages

Dialogue structure

Dialogue management with FSA is simplified. That means that they are easy and quick to develop, as long as the domain is well-structured. The developer can define the states and transitions necessary based on the structure already present in the domain. This also presupposes that only the system has the initiative, because as soon as the user is allowed to take the initiative the input and structure cannot be controlled. Moreover, when there are more than just the basic dialogue requirements for a domain, FSA become an effort-some and time-consuming dialogue modeling method that is moreover not very robust. The developer has to hard-code every single behaviour while at the same time he always runs the risk of leaving something out, which can eventually cause the system to break down. In other words, there is no clear way of capturing a general conceptualization that makes a system less bug-prompt.

User input

Another advantage exemplified by FSA is that speech recognition and interpretation are simplified, because of the predefined user input. As far as the latter is concerned, keyword spotting is enough. That means that only the important semantic entities in the user's answer need to be recognised and interpreted. That, however, amounts to the restriction of the input the user is supposed to give in a very unnatural mode and any information based on structure of the input is lost. Moreover, the user cannot give more information than what has been asked for explicitly each time, since there is no handling of over-answering.

Conclusion

The above drawbacks make dialogue modeling with FSA appropriate only for very simple domains with flat structures. Any domain that involves possible sub-tasks, and especially in an unpredictable order, would require a lot of expensive backtracking in order to recover the state before the sub-task was initiated.

Summing up

In summary, the *advantages* of FSA are:

- Quick and easy to develop
- Controlled user input
- Simple speech recognition
- Simple interpretation
- Appropriate for well-structured domains

The *disadvantages* of FSA are:

- Need to hard code dialogue specifications
- Need to predict dialogue possibilities
- Inflexible structure
- Prompt to break-downs
- Inappropriate for more complicated domains

2.5.2 The CSLU tool

One rather practical advantage of finite state techniques for dialogue processing is that there are various tools for designing dialogue automata that are readily available. If you are keen on finding out more about FSA designing tools, here[2] is the link to the CSLU tool webpage. That includes information on the toolkit and tutorials on use. The tool runs only on Windows.

2.5.3 Beyond Finite State Techniques

As we've just seen, using finite state techniques in dialogue processing may lead to good results with relatively little effort, but only under certain conditions. If for instance great flexibility in a complicated domains is what is needed, finite state techniques often don't lead to satisfying solutions.

In such cases, one uses methods that are more sophisticated linguistically as well as technically. Such methods often involve a fair amount of *reasoning*, for instance about plans and communicative intentions. A key idea is that of representing the user's and the system's knowledge state explicitely and employ reasoning techniques on these state representations. Fully-fledged reasoning also allows for the consideration of domain knowldege to the extent needed for solving even quite complex tasks. Another concept of great importance in designing advanced dialogue systems is that of speech acts (page 25) (or dialogue acts), defining what role a each utterance plays within a dialogue.

Two examples of quite ambitious dialogue systems are the TRAINS system (we have seen in Section 1.3) and its successor, the TRIPS system which we will present in Section 3.4.

[2] `http://cslu.cse.ogi.edu/toolkit/docs/index.html`

3

Speech Acts and Dialogue Management

3.1 Introduction to Speech Acts

3.1.1 The background

The idea of *speech acts* has its roots in the Philosophy of Language. J. A. Austin ([5]) was the first one who wanted to capture the fact that there is more in the function of language than semantics. Traditionally, mapping of entities of a proposition onto referents and defining the truth value of a proposition was the major area of interest in language semantics. With Austin, and his follower J. R. Searle, there is a shift towards the events or acts that occur via language, hence the name 'speech acts'. These acts effect changes both in the observable world, as well as in the mental states of dialogue participants. Austin's approach introduces pragmatics in studying and modeling language. Consequently, the focus is now on utterances and not propositions.

Note: How can a truth value be assigned to the utterance 'Submit the answers to the exercises to Alexander Koller'? Is the utterance true or false?

3.1.2 What are speech acts and what do they do?

According to Austin, there are three types of acts that can be performed by every utterance, given the right circumstances:

Locutionary is the act of actually uttering.

Illocutionary is the act performed in saying something. The illocutionary act is not in one-to-one correspondence with the locution from which it is derived. There are different locutions that express the same illocution and vice-versa. For example, there are indirect speech acts, that is acts with a different force than the obviously deducible one. A typical example is the locution of the utterance 'Could you pass the salt?' uttered at a dinner table. For a speaker of English in the particular situation this means 'Pass the salt, please' and no one would assume that the speaker is indeed interested in whether the addressee would be able to pass the salt.

Perlocutionary is the act performed by saying something in a particular context. It represents the change achieved each time, in a particular context. Depending on the kind of perlocution, different conditions have to hold in order for it to be achieved. For example, the addressee in the salt example has to realise that the speaker's intention is to ultimately get hold of the salt.

Verbs that name the speech act that they intend to effect are called *Performatives* . A performative uttered by the right person under the right circumstances has as a result a change in the world. For example, 'I pronounce you husband and wife' uttered by a priest, in the church with all the legal and traditional aspects being settled, will have the actual effect of the couple referred to being husband and wife after the performative has taken place.

3.1.3 Searle's Classification of Speech Acts

[11] suggests the following classification of speech acts:

Assertives **:** They commit the speaker to something being the case. The different kinds are: suggesting, putting forward, swearing, boasting, concluding. Example: 'No one makes a better cake than me'.

Directives **:** They try to make the addressee perform an action. The different kinds are: asking, ordering, requesting, inviting, advising, begging. Example: 'Could you close the window?'.

Commisives **:** They commit the speaker to doing something in the future. The different kinds are: promising, planning, vowing, betting, opposing. Example: 'I'm going to Paris tomorrow'.

Expressives **:** They express how the speaker feels about the situation. The different kinds are: thanking, apologising, welcoming, deploring. Example: 'I am sorry that I lied to you'.

Declarations **:** They change the state of the world in an immediate way. Examples: 'You are fired, I swear, I beg you'.

3.2 Speech Acts in Dialogue Management

3.2.1 Important Aspects

In general there are certain aspects that need to be defined in order for speech acts to be used in dialogue management.

- Precise definitions of every speech act are needed. That involves conditions specifying what counts as a speech act of a particular kind. It also involves enumerating effects that a speech act has for the dialogue state, the mental state of the participants, the task, where applicable, or whatever area in a domain speech acts can bring a change about.

- Recognition criteria are also necessary. Speech act definitions utilise, to a large extent, the mental state of the participants, which is not directly observable. This might render the conditions specified insufficient for the recognition of speech acts. In that case, different recognition criteria need to be defined or the definitions need to be enriched and augmented to enable robust recognition.

- *Planning schemes* must be defined as well. These should encompass the top level of how far into the dialogue the system should plan. That means, deciding if planning the speech acts of the next utterance is enough, or if planning further ahead is necessary for better communication. The other level to be considered is that which connects whatever it is to be communicated with the realisation that achieves that aim.

- Last but not least, the role of speech acts itself in dialogue management must be clear. If speech acts are only the means of communication and not the object of communication, then an object, as well as its relation to speech acts must be defined. So far, intentions have been commonly assumed to be the object to be communicated (See Section 3.4.9).

3.2.2 Context and Dialogue Management

The locutionary and perlocutionary force of utterances are always interpreted and achieved in a particular context. As far as dialogue management is concerned, that results in a need to define speech acts for different genres and take into account the characteristics of a dialogue and the specific dialogue context in which they appear. For example, different speech acts are used in a human-human than a human-computer interaction; For one, there is an increase in the need for grounding (see Section 1.1.2) in the latter. The same is true for a general conversation versus task-oriented interactions; There will be more speech acts used in the latter in order to co-ordinate the performance of the task.

3.2.3 More Context

Searle's great contribution to speech acts was his attempt to define formally the conditions under which different kinds of illocutionary acts are performed. In other words he tried to define the context that makes speech acts succeed. The conditions he specified involve knowing the conventions of a language, paying attention, understanding the locutionary act etc. In Section 3.4 we will look into the TRIPS system as an example of a dialogue management architecture that uses speech acts. The qualitative difference between what Searle and other people after him have done and the TRIPS approach, is the juxtaposition of linguistic clues, intentions and planning in interpreting speech acts. Planning provides the necessary context for disambiguating speech acts. The thing that brings the intentions of the speaker and planning together is speech acts (See also Section 3.4.9).

3.3 Example Dialogue Annotation Schemes

3.3.1 Introduction of Dialogue Acts

In order to model dialogue by use of speech acts there is a need to enrich the original notion and do away with the simplifying assumptions that it presupposes. In other words, the multiple function of a single utterance at the speech act level has to be accounted for. Moreover, communication cannot be assumed, understanding has to be established. Finally, speech acts cannot be interpreted outside the context in which they appear as they are commonly only one part of a higher, more complicated goal. This shift from original speech acts is captured in naming acts used in dialogue modeling *dialogue acts* .

An empirical approach to dialogue act modeling, that is one that is grounded in actual data, presupposes that the data are annotated for the dialogue acts each theory/genre/domain makes use of. That is not a trivial task. For one it is very time consuming. The practical exercise of this lecture will give you the opportunity to discover for yourself what annotating data involves, as well as an understanding of the connection between annotation and theory.

3.3.2 HCRC Map task

The *HCRC map task* (see HCRC page[1]) annotated corpus consists of transcripts of spontaneous task-oriented dialogues between two human agents. One agent holds a map with landmarks and a route that he has to communicate to the other. The second participant holds a slightly different map without the route. The task is to reconstruct the route on the second map. The derivation of dialogue acts was done independently of the particular task and the aim was to produce a blueprint for dialogue coding, rather than an instantiation of that blueprint for the particular dialogues in the data.

3.3.3 HCRC Map task: Levels of annotation

1. *Conversational moves:* They represent the utterance function and are categorised according to their function. They consist of different kinds of initiations and responses. An example is the *instruct* move, which commands the partner to carry out an action.

   Route-giver: Go right round, ehm, until you get to just above them.

2. *Conversational games:* They represent sets of moves. They are defined in terms of initiations which give rise to different discourse expectations. These expectations must be fulfilled by the responses. Each game starts with an initiation and ends when the goal of the game has been fulfilled or abandoned. Conversational games are equivalent to dialogue games, interactions or exchanges, often found in the bibliography. Games can nest. Adjacency pairs and insertions would appear in this level.

3. *Transactions:* They consist of conversational games and represent sub-dialogues towards achieving a major goal in the task. In the map task, all the interactions necessary for reconstructing a segment of the route make up a transaction.

[1] `http://www.hcrc.ed.ac.uk/maptask/`

3.3.4 Verbmobil 2

The *Verbmobil* (see Verbmobil page[2]) corpus comprises transcripts of human-to-human, spontaneous, task-oriented dialogues. The task is to negotiate on appointment scheduling between the participants. One participant is also able to give travel information on demand by the other. The interactions can be in German, English and Japanese.

3.3.5 Verbmobil 2: Levels of annotation

1. Dialogue acts: They are structured in a hierarchical order and there are decision trees for choosing the right act. As an example, the branching that leads to the act REQUEST-COMMENT, starting from the top, is: DIALOG-ACT, PROMOTE-TASK, REQUEST, REQUEST-COMMENT. REQUEST-COMMENT takes place when a dialogue participant requests the other to explicitly comment on a proposal the first has made. The example that follows is from the English corpus:

   A: Thursday, evening; would that be fine?

2. *Dialogue phases* : They represent stages in the dialogue. Dialogue acts are marked for the phases in which they can legally appear and dialogue phases consist of dialogue acts. Protocols of behaviour can be generated based on the current dialogue phase. There are five phases:

   (a) *Hello:* greeting and introduction on both sides.
   (b) *Opening:* introduction of the topic for negotiation.
   (c) *Negotiation:* everything that the negotiation involves.
   (d) *Closing:* recapitulation of the already agreed upon topic.
   (e) *Goodbye:* both participants say goodbye.

3.3.6 TRAINS

In Section 1.3 we have introduced the TRAINS system. Its corpus consists of transcripts of simulated human-computer interactions. The user/manager gives a transportation task to the computer, that is, a task of finding the best way to realise the transportation desired on a shared map. The system has more information on the practicalities of the task. The user/manager is responsible for setting the goals.

3.3.7 TRAINS: The approach

The annotation makes use of *Conversation Acts* . The emphasis of the annotation is on the way dialogue participants establish mutual understanding of what is being discussed. A *Discourse Unit* represents the aggregate of acts necessary for establishing a piece of information. It is an approach rooted in the cooperative nature of dialogue. There are four basic kinds of conversation acts:

1. *Core speech acts* : These are the traditional speech acts like Inform, Request, Promise, wh-questions etc.

[2] `http://verbmobil.dfki.de/`

2. *Argumentation acts* : They represent higher goals and are made up of Core speech acts. For example, an Inform core speech act may serve the argumentation act of summarising what was previously said.

3. *Grounding acts* : Are used for establishing information between the participants. They include categories like initiate, continue, acknowledge, repair etc.

4. *Turn taking acts* : They are keep-turn, release-turn and its sub-variant assign-turn, and take-turn. They define turn taking.

3.3.8 DAMSL

DAMSL (Dialogue Act Markup in Several Layers[3]) is different from the previous annotation schemes, in the sense that it was not created to fit the needs of any particular domain or corpus. It was proposed as the standard annotation scheme by the DRI (Discourse Resource Initiative) for dialogue tagging. Therefore, it is domain and task independent.

3.3.9 DAMSL: The scheme

The main aim is to capture the multiple function utterances can have, as well as the interrelation of different speech acts. There are three layers of Communicative Acts described:

1. *Forward Communicative Functions* : They correspond, by and large, to the traditional speech acts.

2. *Backward Communicative Functions* : They account for the relation of the current utterance to the dialogue up to that point.

3. *Utterance Features* : They capture information both about the content and the form of utterances. For example, they give information on the relation of the utterance to the communication and task management. An utterance is labeled for its communication management function only when there is no task management function present.

A dialogue should be annotated on all three levels. The different categories within Forward and Backward Communicative Functions are independent, so an utterance can be annotated for more than one dialogue act belonging to the same level. Within each level there is a further hierarchical structure. For example, the Backward Communicative Function *Agreement* is sub-divided into *Accept, Accept-Part, Maybe, Reject-Part, Reject* and *Hold*.

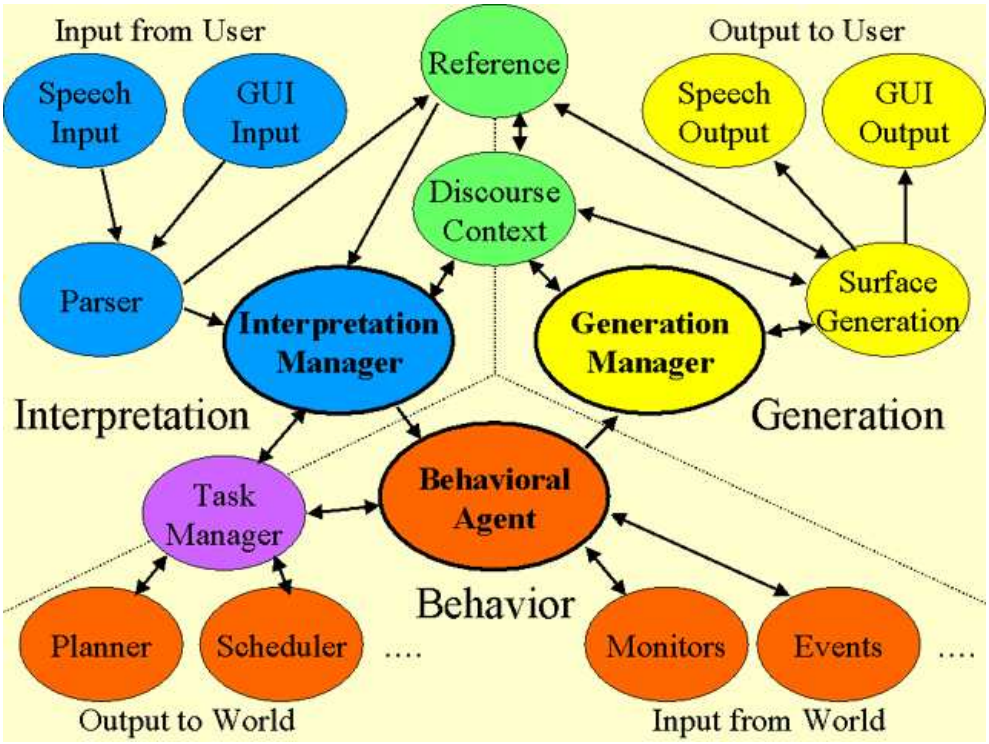
[3] `http://www.cs.rochester.edu/research/cisd/resources/damsl/RevisedManual/RevisedManu`

3.3.10 Automated tagging of dialogue acts

3.4 TRIPS

3.4.1 System Architecture

The *TRIPS* (see TRIPS page[4]) system was built with an emphasis on modularity. It, therefore, consists of a number of modules that are each responsible for different tasks and inter-communicate to share the necessary information for the overall effect. The communication is again managed by a general purpose manager. This architecture allows the extension of the system by plugging in different autonomous off-the-self resources. There are three main components that are responsible for the communication and management of the rest: The Interpretation Manager, the Behavioural Agent and the Generation Manager. They operate asynchronously.



3.4.2 Human-computer communication

There are a number of modes for the human and the computer to interact. Spoken, typed, moused and menu selection input is possible. Output can be spoken or displayed. They all communicate with the chart parser and the generator.

3.4.3 The Interpretation Manager

It controls all the interpretation components. It takes as input speech acts recognised by the Parser (Section 3.4.5). It interprets the intentions of the user from those speech acts. It also coordinates reference resolution, works out discourse obligations (Section 3.4.9)

[4] http://www.cs.rochester.edu/research/cisd/projects/trips/

and creates domain specific knowledge representation. All that is done by context information, which it gets from the Discourse Context Component (Section 3.4.8), and with domain information, which it gets from the Task Manager (Section 3.4.6).

3.4.4 The Speech recogniser

The *speech recogniser* takes speech as input and it outputs word sequences and messages to draw attention on interpretations that might need revising.

3.4.5 The Chart parser

The *chart parser* takes input from all input sources. It's output is a set of speech acts, that is, an interpretation of what the linguistic form of the structure is trying to accomplish.

3.4.6 Task Manager

It turns general planning strategies to specific plans and comes up with a course of action, taking into account the user's input. It can make use of different resources for the planning and plan recognition depending on the needs of the domain, for example, databases, planners, schedulers.

3.4.7 The Reference and Discourse Context Components

The Reference and Discourse Context Components are used both by the Interpretation and the Generation Manager.

The Reference Component It resolves any references that occur in the user utterance. It maps references to objects defined in the domain or, for indefinite references (a vehicle), it maintains a query to be resolved later.

The Discourse Context Component It is responsible for the manipulation of discourse obligations and grounding, in collaboration with the the Interpretation Manager. Information about reference resolution, elipsis and turn taking also comes from the Discourse Context Component.

3.4.8 The Generation Manager

It communicates with the Discourse Context Component and the Behavioural Agent. Based on their input it reasons about the specific content of the next system utterance.

The Speech Generation Component It takes as input the output of the natural language generator. It generates spoken output.

The Surface Generation Component It takes the output content specified by the Generation manager and provides the surface realisation for it.

3.4.9 Speech acts in TRIPS

Speech acts are used in TRIPS by the Interpretation Manager in interpreting the intentions of the user. Intentions are then used, in turn, by the Task Manager in order to evaluate what the user's plan with regard to the task is. If there is more than one possible speech act assigned to an utterance, the right one with the deriving intention that better fits the task will be finally assigned to it. The user's plan of dealing with the task is then evaluated. Based on that plan evaluation the system can then take care of the necessary arrangements for the plans to be realised or suggest alternatives. This is the basis of the mixed initiative modeled by TRIPS.

The Interpretation Manager also uses speech acts for the manipulation of discourse obligations. Obligations represent the standard behaviour imposed by the domain. Based on these obligations, the system can disambiguate any ambiguous input. The Generation Manager, as well, decides on the content of the next system utterance by help of obligations. In general the building block of its reasoning are speech acts.

Adjacency pairs and insertions are also handled by use of speech acts so that there is no need to call the problem solving modules when dealing with these phenomena. For example, the system can automatically issue acknowledgements, that is, declarations of having understood what the user said, following a statement by the user.

On the whole the manipulation of discourse based on speech acts makes the separation of task and discourse planning possible. This separation results in a robust system and allows further improvements in each sector while the other one can be left intact.

Summary

In Summary, speech acts are used for:

1. Interpretation
2. Intention recognition
3. Plan recognition
4. Obligation managing
5. Adjacency pairs and insertions
6. Generation

3.5 References

3.6 Practical exercise.

3.6.1 A small dialogue act taxonomy

Here is a basic dialogue act taxonomy:

Assert The speaker asserts a fact.

Action-directive The speaker asks the addressee to perform a particular action.

Info-request The speaker asks for information.

Offer The speaker offers to do something.

Accept The user accepts a prior offer.

Reject The user rejects a prior offer.

Signal-non-understanding The speaker shows that he has not understood something.

Acknowledge The speaker shows that he has understood something.

Answer The speaker answers a previous question.

Other An act that is not covered by this taxonomy.

The following dialogue is the TRAINS example dialogue which we saw in Lecture 1.

User[1]:	I need to get to Detroit, Syracuse and Montreal.
Syst[2]:	OK. [highlights the three cities]
User[3]:	Let's send the trains from Chicago to Detroit.
Syst[4]:	Ok. [draws route from Chicago to Detroit via Toledo] But the terminal at Toledo is delaying
User[5]:	That's ok. Now let's go from Atlanta up to Syracuse.
Syst[6]:	Please tell me a route to use to get from Atlanta to Syracuse [highlights engine at Atlanta and
User[7]:	Go via Charleston and Pittsburgh.
Syst[8]:	Ok. [draws route via Cincinnati] But traffic through Cincinnati is being delayed due to constr
User[9]:	Ok, let's go via Charleston and Buffalo instead.
Syst[10]:	No problem. [draws route]
User[11]:	That's great.
Syst[12]:	I hope it was helpful.
User[13]:	Finally, I want to go from Montreal to Philadelphia.
Syst[14]:	Ok. [draws route]
User[15]:	That's great. I'm done.

3.6.2 Instructions

1. Each person should annotate the above dialogue for turns and utterances within turns. Number the turns with $T_1, ..., T_n$ and utterances with $U_1, ..., U_n$.

2. Each person should annotate the above dialogue using the small dialogue taxonomy in Section 3.6.1. You are asked to annotate utterances for dialogue acts, that is, say which of the functions described is performed by every utterance. You are also asked to annotate insertions, that is, indicate the initiation and ending when there is a sub-dialogue occurring.

3. In groups of three, compare your annotations: Provide justification for your common choices. Note down your disagreements, if any, and try to explain why the disagreement arose.

4. Are there shortcomings in the taxonomy that you were given? How well could the TRAINS dialogue be predicted based on your annotation? How could you augment/structure the taxonomy to overcome problems?

3.7 Comments on the Exercise

3.7.1 Annotation reliability

If you didn't agree with each other don't worry. Inter-annotators' agreement is a common problem in annotating data. Quite commonly the agreement among different annotators is as small as the agreement that would be expected to occur by chance.

In order to make coding credible enough and be able to test empirical hypothesis by comparing the latter to the data, [6] have argued for the application of a statistical measure known as the *Kappa Coefficient*. The Kappa Coefficient measures pairwise agreement among coders who make category judgments. In terms of dialogue data annotation this can be applied to the decision coders have to make between the different dialogue acts available in the taxonomy used for the annotation. The definition of the Coefficient for dialogue acts annotation is the following:

$$K = (P(A) - P(E))/(1 - P(E))$$

where $P(A)$ is the proportion of times that the annotators agree on the dialogue act assigned to an utterance and $P(E)$ is the proportion of times that they are expected to agree by chance.

A suggestion of how reliability can be achieved was first introduced by [9] for the field of content analysis. He suggested three different ways of tested reliability, which are applicable to annotation reliability:

Stability It measures the agreement between two annotations by the same annotator at two different points in time. There should be no significant difference.

Reproducibility It measures agreement between different coders. There should be no significant difference.

Accuracy It measures the difference between a coder and a standard annotation. The standard is commonly the annotation of an expert, the developer. In that case the results of the test show if the developer's instructions to the coders capture the goal of the developer's scheme.

Two points should be made in relation to the interpretation of the test results. First, that the amount of agreement attributed to chance is dependent on the relative frequency of the dialogue act category. Second, the results of a coding category are dependent on the results of any category that is relying upon. That means, for example, that if annotators do not reach a significant agreement when coding for dialogue participants' turns, the results of coding for utterances, for dialogue acts and insertions will not exhibit significant agreement either.

3.7.2 Multi-functional utterances

One utterance can be performing more than one dialogue acts. This means that the dialogue act taxonomy should allow an annotation at different levels. The annotation scheme should not force a choice between these functions. We have seen at least two good examples of how this issue can be handled, namely, DAMSL (See Section 3.3.8 and Section 3.3.9) and Section 3.3.9, Verbmobil (See Section 3.3.5, Dialogue acts). They both allow annotation at different levels or for different functions.

3.7.3 Sub-dialogues

A dialogue act taxonomy needs to account for sub-dialogues/insertions that might occur. The idea of Dialogue Games and Transaction in the HCRC Map Task (See Section 3.3.3) is meant to do exactly that. Two more examples of a similar idea are Dialogue Phases in Verbmobil (See Section 3.3.5) and Argumentation Acts in TRAINS (See Section 3.3.7).

For planning to take place adjacency pairs are commonly used as a means of judging what is the appropriate dialogue act to be produced next. A good example of modeling adjacency pairs are again Dialogue Games (See Section 3.3.3). In Subsection Section 3.2 we briefly addressed the issue of how far a system should plan, which is also a relevant decision to be made.

3.7.4 An example annotation

None yet.

4

Practical Session (MiLCA-Summer-School Tübingen, 2003)

4.1 Exercise: Philips train information system

4.2 The CLT tool

4.2.1 Dialogue Specifications

Dialogue specifications in the CLT tool consist of three components:

1. a graph that represents the dialogue automaton (See Section 4.2.3)
2. a list of devices that are used by the dialogue (See Section 4.2.2)
3. a list of variables that can be used by the dialogue (See Section 4.2.2)

4.2.2 Devices and Variables

Devices

Devices are things like speech recognisers, speech synthesisers, database interfaces, and other external programs that the dialogue manager can talk to. New devices can be defined for the current dialogue specifications from the `Graph - Devices` menu. Every device has a name and a port. Names and port numbers can be edited by double-clicking on the field in the table. The choice of a name is totally up to the developer. The port, however, must match the port used by the client program that the dialogue manager should talk to. For example, our speech synthesis client uses port 1235, so the port should be set to 1235, as well, in the definition of the synthesis device in the CLT tool.

Variables

Variables are just like ordinary Java variables. They have a name, a data-type (string, integer, float, or boolean), and a value. You can define new variables with their types and initial values in the `Graph - Variables` menu.

4.2.3 The dialogue automaton

The main part of the dialogue specifications is a graph that consists of the following types of nodes (we discuss the characteristics of most of them in the next sections):

1. Exactly one start node in which the dialogue starts.
2. End nodes: if the dialogue goes into such a state, it ends.
3. Output nodes, from which an output is sent to a device (See Section 4.2.4).
4. Input nodes, which wait for input from a device and then go to a new state (See Section 4.2.4).
5. Set-variable nodes, which set the value of a variable (See Section 4.2.4).
6. Condition nodes, which branch depending on a condition (See Section 4.2.4).
7. Test-variable nodes, which branch depending on the value of a variable (See Section 4.2.4).
8. Subgraph nodes, which represent a complex subtask that is used only in one place in the dialogue. (See Section 4.2.4).
9. Procedure nodes, which represent a complex subtask that can be called several times, possibly with different parameters. (See Section 4.2.4).

4.2.4 Nodes

Nodes can have outgoing edges that connect them to other nodes. Some node types (e.g. end nodes) may never have outgoing edges. Some (e.g. start nodes) must have precisely one. Some (e.g. conditional nodes) can have an arbitrary number of outgoing edges.

A node is inserted into the graph by right-clicking into the location where the node should be, and then selecting the node type from the context menu. Nodes can always be moved around by dragging them with the mouse.

In general, every node has properties, which can be edited by right-clicking on the node and selecting `Properties` from the context menu, or by double-clicking on the node.

Nodes are little coloured boxes. Attachment points for outgoing edges appear in the form of little triangles at the lower part of the box. An edge is added by dragging a triangle onto another node. The active edge is coloured red, all other nodes are blue. An edge can be activated by clicking on it. An active edge can be deleted by pressing the `Delete` key, or by selecting `Edit - Delete` from the menu.

Output nodes

Output nodes send some output to a device. What output is sent to which device can be defined in the `Output` tab of the properties. The value can be any Java expression that evaluates to a string, and may contain variables. Outputs to the same device are sent in the order in which they appear on the right-hand side. Outputs to different devices are

sent concurrently. Select a device on the left, then add outputs using the `New` button. The most useful output type is `Text`, which sends a string to the device.

A `Reset` signal can be selected using the `Options` tab, if the developer wants to send a `Reset` signal to the client first, which may or may not do something, depending on the implementation of the client. If `Wait until output completed` is selected on this tab, the dialogue manager waits until the clients have completed all output requests. This may take several seconds, for example, for speech synthesis, before going on with the dialogue. Otherwise, it goes on immediately.

Input nodes

`Input` nodes wait for input from a device. The device input should come from can be selected from the `Options` tab. All or just one device can be selected. Then keywords need to be specified in the `Input` tab. Every keyword will correspond to one little triangle for an outgoing edge; left to right is top to bottom.

A timeout can be specified on the `Options` tab. If no input comes from the given devices in so many milliseconds, the dialogue continues over the red default edge. A prompt can also be sent to output devices. It needs to be specified in the `Output` tab, and probably `Wait until output completed` in the `Options` tab needs to be selected.

Variable nodes

In a Set-variable node, the variable and the value one wants to assign to it are specified in the `Node` tab.

In a Test-variable node, values of one variable one wants to test for can be specified on the `Node` tab. That creates one output edge per variable value.

Condition nodes

If a Java expression is included evaluating to a boolean value into the `Condition` field on the `Node` tab, the dialogue manager will evaluate this expression and branch to the green or red out-edge depending on the value.

Subgraph and Procedure nodes

A subgraph node represents a sub-dialogue that can be called only once in the whole dialogue. It basically compresses the subgraph representing the sub-dialogue into one node, so that the complete graph is easier to understand.

Procedure nodes are also sub-dialogues but they can be called many times in the dialogue. Procedures can have parameters, which are like parameters of a procedure in a programming language. Parameters and their types can be specified in the `File - Parameters` menu of the subgraph window. The specified parameters can then be used as variables (in Test-variable nodes, or in expressions) throughout the subgraph. Parameters cannot be part of the main graph, but the main graph can contain call nodes that call the procedure with actual parameters.

By double-clicking on a subgraph or a procedure node, another graph window can be opened in which the subgraph/procedure can be specified. The subgraph/procedure should contain end nodes; the subgraph node in the main graph and every call node for this procedure will both have an outgoing triangle for every end node in the subgraph.

4.3 Running a dialogue

4.4 Exercise: Dialogue design using the CLT tool

4.5 Designing a speaking elevator dialogue

Bibliography

[1] Jan Alexanderson et.al. Dialogue acts in verbmobil-2. Technical report verbmobil-report 204, DFKI GmbH Saarbrücken, Universität Stuttgart, Technische Universität Berlin,Universität des Saarlandes, 1997.

[2] James Allen, George Ferguson, and Amanda Stent. An architecture for more realistic conversational systems. In *Proceedings of Intelligent User Interface, IUI01*, pages 14–17, Santa Fe, New Mexico, USA, 2001.

[3] James Allen et. al. Robust understanding in a dialogue system. In *Procedings of 34th Meeting of the Association Computational Linguistics*, June, 1996.

[4] Eric Auer, Claudia Crispi, Christian Dressler, Cordula Klein, Kerstin Kloeckner, Norbert Pflenger, and Diana Raileanu. Sprachsteuerung im aufzug. Unpublished, 2002.

[5] J. A. Austin. *How to Do Things with Words*. Harvard University Press, 1996.

[6] Jean Carletta, Amy Isard, Stephen Isard, Jacqueline C. Kowtko, Gwyneth Doherty-Sneddon, and Anne H. Anderson. The reliability of a dialogue structure coding scheme. *Computational Linguistics*, 23(1):13–32, 1997.

[7] Mark G. Core and James F. Allen. Coding dialogues with damsl annotation scheme. In *AAAI Fall Symposium on Communicative Action in Humans and Machines*, pages 28–35, Boston, MA, November 1993.

[8] George Ferguson, James Allen, and Brad Miller. Trains-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 70–77, Edinburgh, 1996.

[9] Klaus Kripperndorf. Content analysis: An introduction to its methodology. *Sage Publications*, 20(0), 1980.

[10] Michael McTear. Modelling spoken dialogues with state transition diagrams: Experiences with the cslu toolkit. In *In Proceedings of the 5th International Conference on Spoken Language Processing*, Sydney, Australia, 1998.

[11] J. R. Searle. A taxonomy of illocutionary acts. *Language, Mind and Knowledge, Minnesota Studies in the Philosophy of Science*, pages 344–369, 1975.

[12] Andreas Stolcke et. al. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistic*, 26(3):339–371, 2000.

[13] David R. Traum. Speech acts for dialogue agents. *Foundations of Rational Agency*, pages 169–201, 1999.

[14] David R. Traum and Elizabeth A. Hinkelman. Conversation acts in task-oriented spoken dialogue. *Computational Intelligence Special Issue: Computational Approaches to Non-Literal Language*, 8(3):575–599, 1992.

Index