15

Dialogue

15.1 Introduction

While discourse materialized in texts delivers static information, dialogue is dynamic and consists of two interacting discourses. Once written, a discourse content is unalterable and will remain as it is for its future readers. On the contrary, a dialogue enables exchange information flows, to complement and to merge them in a composition, which is not known in advance. Both dialoguing parties provide feedback, influence, or modify the final content along with the course of the conversation.

In this chapter, we will envision dialogue within the framework of an interface between a system and a user. Parties have information to transmit or to request using natural language. The dialogue purpose will be to make sure that the request is complete or the information has been well captured or delivered. Naturally, as for other discourse applications, a dialogue module is only a part of the whole system using language processing techniques we have described before.

15.2 Why a Dialogue?

The first role of a dialogue module as an interface is to manage the communication and to coordinate the turn-taking between the user and the system. It is also a kind of integration shell that calls other language processing components to analyze user utterances or to generate system answers. In addition, interaction and dialogue techniques can help linguistic analysis be more flexible and recover from failures.

We saw that coreferences are sometimes difficult to resolve. They provide an example of interaction usefulness. Instead of having an interactive system conjecture about an ambiguous pronoun, it is often safer for it to ask the user himself/herself to resolve the ambiguity. Two strategies are then possible: the system can infer a missing reference and ask the user for a confirmation. Or, in case of a more difficult ambiguity, it can ask the user to reformulate completely his/her sentence.

To summarize, dialogue systems can help manage a user's discourse to:

- complement information when pieces are missing to understand a sentence or to carry out a command
- clarify some ambiguous words or constructions
- confirm information or intention to manage errors when a certain failure rate is unavoidable, e.g., with speech recognition operating on naturally flowing speech

15.3 Simple Dialogue Systems

Many simple dialogue systems in commercial operation aim to provide information through telephones. Such **speech servers** receive calls from users requesting information in a specific domain and answer interactively to questions. Although many servers still interact with a user using touch-tone telephones, more and more they feature speech recognition and speech synthesis modules.

Speech recognition conditions are difficult with telephone servers since they have usually to handle a poor acoustic environment. It naturally comes at a price: recognition is prone to errors and the number of active words – words that the system can recognize at a given time – cannot be much more than a hundred on many systems. Speech recognition is then a bottleneck that limits the whole system performance.

For reasons of robustness and cost, operating components of real-world dialogue applications rarely correspond to the integration of classical linguistic layers: morphology, syntax, and semantics. The recognition itself does not attempt to produce the full stream of words but generally uses word spotting techniques. Word spotting enables a word to be recognized within a short fragment of surrounding speech or noise. So a word will be correctly identified, even if you say *hmm* before or after it.

Because of word spotting, spoken systems do not stack a complete parsing after speech recognition. They focus on interesting words, meaningful according to context, and link them into information frames. These frames miss some words, but the important issue here is not to miss the overall meaning and to keep dialoguing with the user in real time and at a reasonable computational cost. Typical examples of such dialogue systems, elementary from a linguistic viewpoint, are based on automata.

15.3.1 Dialogue Systems Based on Automata

Dialogue systems based on finite-state automata have transitions triggered by a limited number of isolated words (Fig. 15.1). At each state, the automaton synthesizes a closed question: it proposes a finite choice of options. If the word recognition device has not understood a word, it loops onto the same state, asking for the user to repeat his/her command. If it corresponds to a legal transition, the automaton moves the user to another state.

As we said, speech recognition is not foolproof. The system avoids possible errors through a confirmation message while proceeding to a next state. On the leftmost edge of Fig. 15.1

We are happy to give you information on loans?

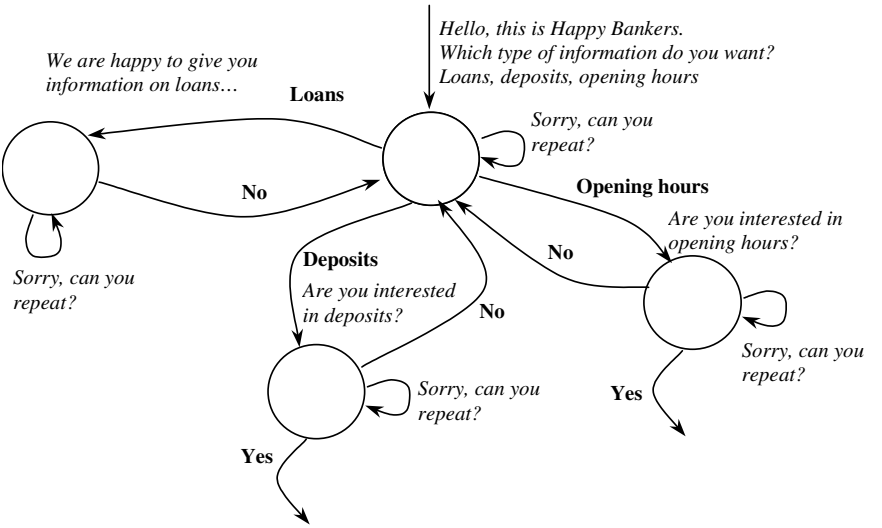

Fig. 15.1. An automaton for word-triggered dialogues.

the system uses an implicit confirmation. The user can accept the ongoing transition either explicitly by saying *yes*, or implicitly by saying nothing. It can also contradict – only if necessary – and reject the transition by saying, for instance, *no* or

No, I wanted to know about my account balance.

In this case, the user will regress to the previous state. Other edges as in the middle and rightmost transitions:

Are you interested in deposits?

correspond to explicit confirmations. They require a mandatory answer – *yes* or *no* – from the user.

As a general design rule, confirmations should not be too numerous to be acceptable by users because they tend to be tedious if overused. The first strategy is certainly preferable.

15.3.2 Dialogue Modeling

The basic structure of dialogue automata is far from providing a natural interaction. However, it implements some fundamental characteristics shared by all dialogue systems. Interactions between the user and the system correspond to pairs: the system's turns and the user's turns. These pairs, which may be nested, have been extensively studied. Levinson (1983) proposed a classification of them according to the nature of their first member. Table 15.1 shows an excerpt of Levinson's classification.

Levinson's model, however, is not sufficient to take possible errors and confirmation into account. Moeschler and others at the University of Geneva (Moeschler

Table 15.1. Classification of dialogue pairs.

First member	Preferred second member	Dispreferred second member
Offer, Invitation	Acceptance	Refusal
Request	Compliance	Refusal
Assessment	Agreement	Disagreement
Question	Expected answer	Unexpected answer, no answer
Blame	Denial	Admission

1989; Moeschler and Reboul 1994) proposed a more elaborate model, which corrects some flaws. The model divides a dialogue as a sequence of exchanges, but it complements pairs with a final assessment. An exchange is a sequence of three different interventions:

- *initiative interventions*, which open an exchange (I)
- *reaction interventions*, which are answers to initiatives (R)
- *evaluation interventions*, which assess exchanges and possibly close them (E)

In addition, exchanges may be nested and hence have a recursive structure.

Table 15.2 shows a first exchange pictured by the leftmost edge of the automaton in Fig. 15.1. It is annotated with intervention tags: I, R, and E. The two first turns are an initiative and a reaction. The last turn is an implicit acknowledgment showing that the system has understood the user command.

Table 15.2. An exchange.

Utt. no.	Turns	Utterances	Tags
1	S:	*Which type of information do you want: loans, deposits, opening hours?*	I_1
2	U:	*Loans*	R_1
3	S:	*We are happy to give you information on loans*	E_1

Along with the deposit question (middle edge), Fig. 15.1 shows a nested interaction in the evaluation. There are first an initiative and a reaction. The third turn is an evaluation, which is a recursive exchange, consisting of an initiative and a reaction (Table 15.3).

15.4 Speech Acts: A Theory of Language Interaction

Turns triggered by isolated words and phrases are a rudimentary dialogue model. Ideally, systems should handle more complicated phrases or sentences. In addition, they should take a more elaborate interaction structure into account.

Table 15.3. An exchange with a nested evaluation exchange.

Utt. no.	Turns	Utterances	Tags
1	S:	*Which type of information do you want: loans, deposits, opening hours?*	I_1
2	U:	*Deposits*	R_1
3	S:	*Are you interested in deposits?*	$\left.\begin{matrix} I_1^2 \\ R_1^2 \end{matrix}\right] E_1$
4	U:	*Yes*	

Language is basically a means for people to act one upon the other. However, linguistic theories as we have considered them until now do not cover this interaction aspect. Some authors have tried to remedy it and to investigate other approaches. In contrast to formal or lexical semantics, which are based on logic, they have attempted to give language a more "performative" foundation. Such a framework has interested modern linguists such as Bühler (1934, 1982) and later language philosophers such as Austin (1962) and Searle (1969).

Bühler postulated that language had three semantic functions according to his organon model:

- a representation (*Darstellung*) of objects and the state of affairs that is being described
- an expression (*Ausdruck*) materializing the psychological state of mind of the speaker – the sender of the message
- an appeal (*Appell*) corresponding to an effect on the hearer – the receiver of the message

Although Bühler admitted the dominance of the representation function of language acknowledged before him, he stressed the psychological aspects of spoken communication describing participants as "psychophysical" systems. He was the first modern linguist to introduce that speech involved a sequence of acts that he named *Sprechakt* enabling the hearer to recognize the speaker's state of mind or internal planning.

Austin came to a similar conclusion and considered also speech as a sequence of acts. For each of these acts, he distinguished what pertained to the classical side of linguistics and resorted to morphology, syntax, and semantics from pragmatics and the theory of action. He referred to the former as locutions and the latter as illocutions. From these considerations on, Austin modeled the act of saying something, with three components representing three different aspects of communication:

- *locutionary* – i.e., an act of saying something, corresponding to a phonetic utterance, a syntactic structure, and a formal semantics content
- *illocutionary* – i.e., a conversational act, which can be, for instance, to inform, to suggest, to answer, to ask
- *perlocutionary* – i.e., the effects of these acts, which can be to frighten, to worry, to convince, to persuade

Classical grammar recognizes certain links between locutionary and illocutionary content. Some types of syntactical forms are frequently associated with speech acts (Table 15.4).

Table 15.4. Syntactical forms and speech acts.

Classical speech acts	Syntactic forms
Assertions, statements	Affirmatives or declaratives
Orders, commands	Imperatives
Questions	Interrogatives

However, the association is not systematic. Speech acts are not always related to a logical – or propositional – content that could have been derived from the formal structure of sentences. Rhetorical questions such as *Can you open the door?* are in fact orders, and imperatives such as *Have a good day!* are greetings or wishes. In addition, a syntactical classification is too coarse to reflect the many needs of interaction analysis.

To cope with different aspects of communication, many authors have proposed a classification of illocutionary acts. We retain Searle's initial classes, which are best known because they probably capture essential interaction paradigms:

- **assertives**, such as stating, asserting, denying, informing.
- **directives**, such as requesting, asking, urging, commanding, ordering.
- **commissives**, such as promising, committing, threatening, consenting, refusing, offering.
- **declaratives**, such as declaring war, resigning, appointing, confirming, excommunicating. Declarative speech acts change states of affairs.
- **expressives**, which are related to emotions or feelings such as apologizing, thanking, protesting, boasting, complimenting.

Searle (1969) refines the speech act model by proposing conditions to complete an act successfully. Conditions are a set of conversational postulates that should be shared by speakers and hearers. These conditions are divided into a propositional content, a preparatory condition, a sincerity, and an essential condition. Tables 15.5 and 15.6 reproduce two success conditions to speech acts (Searle 1969, pp. 66–67).

The work of Austin and Searle has been very popular in the computational linguistics community and, far beyond it, in language in certain fields of philosophy and psychology. Although some people consider them as inventors, their findings are not completely new. Gorgias, a Greek rhetorician who lived 2500 years before them, wrote:

> The effect of speech upon the condition of the soul is comparable to the power of drugs over the nature of bodies. For just as different drugs dispel different secretions from the body, and some bring an end to disease and others to life, so also in the case of speeches, some distress, others delight,

Table 15.5. Conditions to *request*, *order*, *command*. After Searle (1969).

Conditions	Values
Propositional content	Future act A of Hearer
Preparatory	1. Hearer is able to do A. Speaker believes Hearer is able to do A 2. It is not obvious to both Speaker and Hearer that Hearer will do A in the normal course of events of his own accord 3. (For *order* and *command*) Speaker must be in a position of authority over Hearer
Sincerity	Speaker wants Hearer to do A
Essential	Counts as an attempt to get Hearer to do A

Table 15.6. Conditions to *greeting*. After Searle (1969).

Conditions	Values
Propositional content	None
Preparatory	Speaker has just encountered (or has been introduced to, etc.) Hearer
Sincerity	None
Essential	Counts as courteous recognition of Hearer by Speaker

some cause fear, others make the hearers bold, and some drug and bewitch the soul with a kind of evil persuasion.

Encomium of Helen (Trans. RK Sprague)

15.5 Speech Acts and Human–Machine Dialogue

15.5.1 Speech Acts as a Tagging Model

Many language processing applications use the speech act theory as a kind of syntax to parse a discourse or a dialogue. Constituents are the discourse segments, and categories are illocution classes, termed broadly as speech acts or dialogue acts. As a result, a discourse is a sequence of segments annotated with conversation acts.

Authors may not follow the Searle's classification. Gazdar and Mellish (1989) provide a small set of "illocutionary acts," among which they quote: request, statement, suggestion, question. Using these acts, Gazdar and Mellish (1989) can label the dialogue in Table 15.7.

Acts such as challenge or concession may be more suited to analyzing a human conversation rather than a spoken human–machine interaction. In addition, applications may need different sorts of acts. Therefore, most sets of speech acts are designed for a specific dialogue system and are closely tied to it. Acts then serve as tags

Table 15.7. Illocutionary acts in a dialogue. After Gazdar and Mellish (1989, p. 385).

Turns	Utterances	Illocutionary acts
A	I really think the automobile needs servicing	Statement
B	But we had done it recently	Challenge
A	No, not for two years…	Challenge Interruption
A	Incidentally did you hear that gas prices are about to double?	Concession

to annotate discourse segments. Although disputable from a theoretical viewpoint, this interpretation of speech acts as tags is used as a model for scores of human–machine dialogue systems. We examine one of them in the next section.

15.5.2 Speech Acts Tags Used in the SUNDIAL Project

Bilange (1992) and Cozannet (1992) list a collection of speech acts that they used in the SUNDIAL project (Table 15.8). The acts are divided into initiatives, reactions, and evaluations following Moeschler's (1989) dialogue modeling. They are intended to enable a user to make a train ticket reservation by telephone.

Table 15.8. Speech acts used in SUNDIAL (slightly modified).

Acts	System/ User act	Descriptions
Initiatives		
`request(P)`	`S`	Open question or request for the value of P
`yn_question(P, Val)`	`S`	Is value of `P` `Val`? Answer should be *yes* or *no*
`altern_question(P)`	`S`	Alternative question: *Vanilla or strawberry*?
`repeat(P)`	`S/U`	Repetition request
`inform(P)`	`S/U`	Inform of `P`
`recap(P)`	`S`	Recapitulation of solved problems
Reactions		
`answer(P, Val)`	`U`	Gives a value `Val` on the request of `P`
`select(P, Val)`	`U`	Gives a value `Val` on an alternative question on `P`
`accept(P, Val)`	`U`	Accept or confirm the value `Val` of `P`
`reject(P, Val)`	`U`	Reject the value `Val` of `P`
Evaluations		
`impl_valid(P, Val)`	`S`	Implicit validation of confirmation of the value `Val` of `P`
`correct(P, Val)`	`U`	Gives a new value `Val` to `P`

Other projects such as VERBMOBIL use speech acts that are even more tied to the application (Jekat et al. 1995). VERBMOBIL provides a language support to an appointment system and its acts include `INTRODUCE_NAME`, `ACCEPT_DATE`, `REJECT_DATE`, `SUGGEST_SUPPORT_DATE`.

15.5.3 Dialogue Parsing

Dialogue applications, for example, speech servers, are aimed at answering relatively simple inquiries such as providing information on train timetables, airfares, or credit card authorizations. Their possibilities are generally well understood by users who call them and who do not expect to have a philosophical conversation with the system.

For this reason, in many applications it is possible to restrict a human–machine transaction to a dialogue opening, a negotiation where a user formulates a problem and solves it with the system, and a closing. Using Moeschler's model, we can describe each of these parts as a sequence of exchanges where utterances are divided into initiatives (I_i), reactions (I_r), and evaluations (I_e).

Table 15.9 shows a dialogue example from Andry (1992), and Table 15.10 shows the derived structure of the negotiation part. Utterances come either from the user (u) or the system (s) and consist of one or more speech acts. Utterance S2

London Paris which date?

is split into two acts. The first one (S1a)

London Paris

corresponds to an implicit confirmation that the system has understood the departure and arrival cities `Ie(s, [impl_valid])`. The second one (S2b)

which date

is an explicit question to the user `Ii(s, [request])`.

We can parse the exchange in Table 15.10 and get its structure using DCG rules. We first write a grammar to model the nonrecursive exchanges. We use variables to unify the speaker – user or system – and the type of act.

```
exchange(ex(i(X, SA1), r(Y, SA2), e(E))) -->
  initiative([X, SA1]),
  reaction([Y, SA2]),
  evaluation(E),
  {X \= Y}.
exchange(ex(i(X, SA1), r(Y, SA2))) -->
  initiative([X, SA1]),
  reaction([Y, SA2]),
  {X \= Y}.
exchange(ex(e(X, SA1), r(Y, SA2))) -->
  evaluation([X, SA1]),
  reaction([Y, SA2]),
  {X \= Y}.
```

We model initiatives, reactions, and evaluations as a sequence of speech acts:

Table 15.9. An example of dialogue with SUNDIAL.

Turns	Utterances
S1	*Bonjour puis-je vous aider?* 'Hello, can I help you?'
U1	*Je veux une réservation de Londres à Paris* 'I want a reservation from London to Paris'
S2	*Londres Paris à quelle date voulez-vous voyager?* 'London Paris which date?'
U2	*Je veux voyager le 20 juin* 'I want to travel June 20'
S3	*Le 20 juin à quelle heure?* 'June 20 which time?'
U3	*À 17 heures 15* '5.15 pm'
S4	*17 heures 15?* '5.15 pm?'
U4	*Oui* 'Yes'
S5	*Le vol BA-123 part à 17 heures 15 et arrive à Charles-de-Gaulle à 18 heures 39, est-ce que cela vous convient?* 'Flight BA-123 departure 5.15 pm and arrival 6.39 pm at Charles-de-Gaulle airport. Is that OK for you?'
U5	*Oui* 'Yes'
S6	*Quel est votre nom?* 'What is your name?'
U6	*Luc Martin*
S7	*Luc Martin une réservation pour Luc Martin sur le vol BA-123 arrivant à Charles-de-Gaulle à 18 heures 39* 'Luc Martin a reservation for Luc Martin on flight BA-123 arriving at Charles-de-Gaulle at 6.39 pm'
U7	*Oui* 'Yes'
S8	*Votre réservation a été confirmée au revoir* 'You reservation has been confirmed goodbye'
U8	*Au revoir* 'Goodbye'

```
initiative([Speaker, SpeechActs]) -->
  acts([Speaker, SpeechActs]).

reaction([Speaker, SpeechActs]) -->
  acts([Speaker, SpeechActs]).

evaluation([Speaker, SpeechActs]) -->
  acts([Speaker, SpeechActs]).
```

Table 15.10. Intervention structure.

Exch.	Interventions	Recursive interventions	Turns
	`Ii(s, [request])`		S1
E1	`Ir(u, [answer])`		U1
	`Ie(S, [impl_valid])`		S2a
	`Ii(s, [request])`		S2b
E2	`Ir(u, [answer])`		U2
	`Ie(s, [impl_valid])`		S3a
	`Ii(s, [request])`		S3b
E3	`Ir(u, [answer])`		U3
E3e		`Ie(s, [impl_valid])`	S4
		`Ir(u, [accept])`	U4
E4	`Ii(s, [recap, yn_question])`		S5a S5b
	`Ir(u, [accept])`		U5
	`Ii(s, [request])`		S6
E5	`Ir(u, [answer])`		U6
	`Ie(s, [impl_valid])`		S7a
	`Ii(s, [recap])`		S7b
E6	`Ir(u, [accept])`		U7
	`Ie(s, [impl_valid])`		S8

To take the recursive exchange into account, we have to add:

```
evaluation(S) --> exchange(S).
```

Finally, we define the dialogue as a sequence of exchanges:

```
dialogue([SE | RS]) -->
  exchange(SE), dialogue(RS).
dialogue([]) --> [].
```

Although these rules do not completely implement Moeschler's model, they give an insight to it.

15.5.4 Interpreting Speech Acts

To complete our dialogue survey, we outline ways to map utterances to speech acts, that is, in our example above, to annotate *What is you name?* as an open question. Some words, phrases, or syntactic features have a correspondence in terms of speech acts, as shown in Table 15.11. A first method is then to spot these specific patterns or cues. Cues enable us to delimit segments, to generate candidate speech acts, and to annotate the corresponding segment content. Once segments are identified, we can proceed to parse them and obtain their logical form.

However, identification is not straightforward because of ambiguity. Some words or syntactic features have more than one speech act candidate. The interrogative mode usually corresponds to questions, but not always as in *Can you do that for*

Table 15.11. Syntactic forms or templates linking utterances to speech acts.

Syntactic features	Candidate speech acts
Interrogative sentence	`yn_question, altern_question, request`
yes, right, all right, OK	`accept, impl_valid`
no, not at all	`reject`
Declarative sentence	`inform, impl_valid`
sorry, pardon, can you repeat	`repeat`
not X but Y, that's not X it's Y in fact.	`correct`

me?, which is likely to be a polite order in a human conversation. The system then produces several possible acts for each utterance or sequence of utterances: *Yes* in the dialogue in Table 15.9 is either an acceptation (U5) or an implicit validation (S4).

The identification of speech acts for unrestricted dialogues has not received a definitive solution yet. However, it has attracted much attention and reasonably good solutions for applications like speech servers. In a spoken dialogue, what matters are the user's acts that an automatic system identifies using a tagging procedure. Tagsets can be relatively generic, as in the SUNDIAL project, or application-oriented, as in VERBMOBIL. DAMSL (Allen and Core 1997) is another oft-cited tagset.

Speech act tagging uses statistical approaches or reasoning rules, or possibly a combination of both. While many systems used by speech servers are based on rules, Alexandersson (1996) describes a statistical technique similar to those used in part-of-speech tagging. He uses a dialogue corpus where the turns are annotated with illocutionary acts instead of parts of speech. The tagger is a hidden Markov model, and the training procedure derives dialogue act n-grams. As for part-of-speech tagging, the stochastic dialogue act tagging consists in finding the most likely sequence of tags given a sequence of words and features.

As general principles, the features that rules or statistical modeling take into account are:

- Cue words or phrases, which may be linked to specific speech acts.
- The syntactic and semantic forms of the utterance.
- Expectations to apply constraints on possible speech acts. These are based on transitions from a previous state to the current state of the dialogue: when the system asks a question, it expects an answer, a rejection or a failure, and it can discard other acts.
- Task modeling and goal satisfaction. This point extends the previous one. It restrains possible user acts and parameter values according to the progress point where the user is in the dialogue.

15.5.5 EVAR: A Dialogue Application Using Speech Acts

EVAR – *Erkennen, Verstehen, Antworten, Rückfragen* – is a dialogue system intended to provide information on train schedules in Germany (Mast 1993, Mast et al.

1994). It gives an example of a thorough task modeling that enables the system to restrain the number of possible dialogue acts at a given point of the inquiry process. It avoids random initiatives from the user thanks to a constant guidance process. EVAR structures the dialogue process in a sequence of phases:

- a greeting by the system
- a request from the user
- a clarification consisting of a confirmation of data requested by the user and possible requests for details and specifications
- an answer from the system
- a closing

EVAR enables transactions such as those in Table 15.12.

The finite state automaton in Fig. 15.2 models the dialogue progress. The `S_` prefix denotes a system's turn, and `U_` denotes a user's turn.

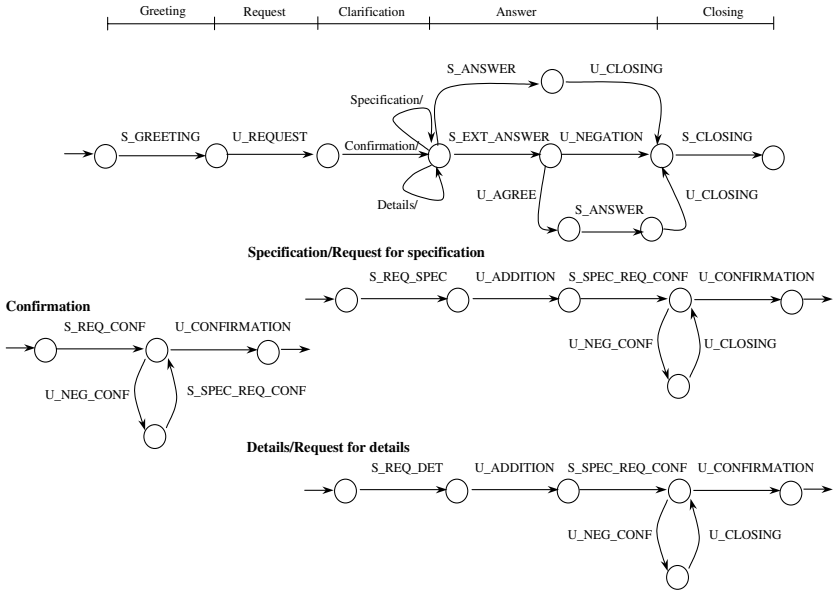

Fig. 15.2. The EVAR dialogue model. After Mast et al. (1994).

15.6 Taking Beliefs and Intentions into Account

The models of dialogue we have examined so far have an external viewpoint in the sense that they are based on observations of the user's behavior by the system. Parallel to them, some authors tried to take into account the user's beliefs, desires, and

Table 15.12. A dialogue in the EVAR prototype. After Mast et al. (1994).

Turns	Dialogue	Phases
S:	*Guten Tag. Hier ist die automatische Intercity-Auskunft. Welche Information benötigen Sie?* 'Hello. This is the Automatic Travel Information System. What information do you need?'	Greeting
U:	*Ich möchte morgen nach Hamburg fahren.* 'Tomorrow I want to go to Hamburg.'	Request
S:	*Sie möchten morgen nach Hamburg fahren?* 'Tomorrow you want to go to Hamburg?'	Confirmation
U:	*Ja morgen.* 'Yes tomorrow.'	
S:	*Von wo möchten Sie abfahren?* 'Where do you want to start?'	Request for details
U:	*München* 'Munich'	
S:	*Morgen von München nach Hamburg* 'Tomorrow from Munich to Hamburg'	
U:	*Ja* 'Yes'	
S:	*Wann möchten Sie fahren?* 'When do you want to leave?'	Request for specification
U:	*Morgen früh* 'Early in the morning'	
S:	*Morgen früh von München nach Hamburg* 'Tomorrow morning from Munich to Hamburg'	
U:	*Ja* 'Yes'	
S:	*Ich habe mehere Verbindungen gefunden. Nach Hamburg HBF, fahren Sie ab München um 7 Uhr. Ankunft in Hamburg HBF um 15 Uhr, . . .* 'I found connections. From Munich departure 7, Arrival Hamburg 15, . . . '	Answer
U:	*Vielen Dank* 'Thank you'	
S:	*Möchten Sie weitere Auskünfte?* 'Do you want further information?'	Closing
U:	*Nein danke* 'No thank you'	
S:	*Auf wiederhören.* 'Have a nice trip.'	

intentions. They hoped to build an internal model of the user and thus to gain a deeper insight into dialogue processes.

The ambition to model beliefs and intentions is appealing because it addresses concerns and questions that often puzzle people: *What does he want from me? What does she mean?* Knowing or extracting a user's intentions would certainly help a computer serve the user better. In addition, intention modeling recasts dialogue and interaction into a framework more general than other formalisms.

However, such a model may prove difficult to implement. It is first a philosophical challenge to figure out and describe what the beliefs of individuals are. Second, if beliefs or intentions could be completely described, it would be a second challenge to box them into a program and run them with a computer. Fortunately, most dialogue applications have a goal that is plainly prosaic, and simplifications can be made. We describe now a classical representation of user modeling introduced by Allen and Perrault (1980).

15.6.1 Representing Mental States

The idea of conversational systems based on belief, desire, and intention is to model the participants as processes – agents. The agents are human users as well as artificial ones, and each agent has a specific knowledge and the desire to complete an action. The agent's core is a representation of their mental states, which uses predicates aimed at describing their beliefs or knowledge spaces, and what they can do. Agents are modeled using operators such as:

- `want(A, X)`, which means that agent A wants to do X
- `can_do(A, X)`, which means that agent A can do X
- `believe(A, X)`, which means that agent A believes X
- `know(A, X)`, which means that agent A knows X

Since beliefs are personal, that is, individual, the definition of truth is no longer universal. For this reason, predicates have two arguments, the agent who is the believer and the proposition that is believed or known. This nonuniversal logic is called modal, and refers to the various modes of truth.

From these operators, some axioms can be derived such as:

```
(know(A, X), (X ⇒ Y)) ⇒ know(A, Y)
(believe(A, X), (X ⇒ Y)) ⇒ believe(A, Y)
(believe(A, X), X) ⇒ know(A, X)
```

Mental states can be different according to dialogue participants, whether they involve human beings together or humans and machines. Let us suppose that a patron goes to a restaurant, looks at the menu, and sees as main courses *cassoulet*, *pytt i panna*, and *Yorkshire pudding*. Let us also suppose that the restaurant is running out of *cassoulet*. When entering the restaurant, belief spaces of the patron and the waiter are different (Table 15.13).

A short dialogue between the waiter and the patron when ordering the meal will enable them to synchronize their belief spaces (Table 15.14).

Table 15.13. Belief spaces.

Patron's belief space	Waiter's belief space
$\exists x, cassoulet(x)$	
$\exists x, pytt\_i\_panna(x)$	$\exists x, pytt\_i\_panna(x)$
$\exists x, yorkshire\_pudding(x)$	$\exists x, yorkshire\_pudding(x)$

Patron: *I feel like a cassoulet*
Waiter: *Sorry sir, we have no more of it.*

Such an exchange is also called a grounding – that is, setting a common ground. Grounding is central to dialogue system design. The user must be sure that beliefs and knowledge are shared between her/him and the system. If not, misunderstanding would creep into many exchanges.

Mutual beliefs can be expressed as `believe(A, P)` $\wedge$ `believe(B, P)` $\wedge$ `believe(A, believe(B, P))` $\wedge$ `believe(B, believe(A, P))` $\wedge$ `believe(A, believe(B, believe(A, P)))`, etc. Such a infinite conjunction is denoted `mutually_believe(A, B, P)`.

Mutual beliefs should not be explicitly listed all the time at the risk of being tedious. Most of the time, the user knows that there is an artificial system behind the box and expects something very specific from it. However, the system has to make sure the user is aware of its knowledge and beliefs, for instance, using implicit confirmation each time the user provides information.

Table 15.14. Belief spaces after dialogue.

Patron's belief space	Waiter's belief space
$\exists x, pytt\_i\_panna(x)$	$\exists x, pytt\_i\_panna(x)$
$\exists x, yorkshire\_pudding(x)$	$\exists x, yorkshire\_pudding(x)$

Representing the corresponding beliefs and intentions using Prolog is straightforward (Tables 15.15 and 15.16)

Table 15.15. Beliefs in Prolog.

Patron's belief space	Waiter's belief space
`believe(patron('Pierre'), cassoulet(X))`	
`believe(patron('Pierre'), pytt_i_panna(X))`	`believe(waiter('Bill'), pytt_i_panna(X))`
`believe(patron('Pierre'), yorkshire_pudding(X))`	`believe(waiter('Bill'), yorkshire_pudding(X))`

Table 15.16. Intentions in Prolog.

Patron's intentions	Waiter's intentions
`intend(patron('Pierre'), (cassoulet(X), order(X)))`	`intend(waiter('Bill'), take_order(X))`

Finally, modal operators can be used to transcribe speech acts. For instance, the act of informing can be associated to the operator `inform(A, B, P)` (`A` informs `B` of `P`), which will be applied with the following preconditions and effects:

- preconditions: `know(A, P)`, `want(A, inform(A, B, P))`
- effects: `know(B, P)`

The operator `request(A, B, P)` can be modeled as:

- preconditions: `want(A, request(A, B, P))`, `believe(A, can_do(B, P))`
- effects: `believe(A, want(B, P))`

15.6.2 The STRIPS Planning Algorithm

Mental state consistency is usually controlled using a planning algorithm. Planning has been extensively studied, and we introduce here the STRIPS algorithm (Fikes and Nilsson 1971, Nilsson 1998). STRIPS considers planning as a search problem given an initial and a final state. It uses rules describing an action – corresponding here to the operators – with preconditions and postconditions. Postconditions are divided into an add and a delete list reflecting facts new to the world and facts to be removed.

```
%strips_rule(+operator, +preconditions, +add_list,
%  +delete_list).

strips_rule(inform(A, B, P), [know(A, P), want(A,
  inform(A, B, P))], [believe(B, P)], []).
```

Mental states or world state are described by lists of facts. The `apply/3` predicate applies an operator to a `State` that results in a `NewState`, subtracting facts to be deleted and adding facts to be added:

```
% apply(+Action, +State, -NewState)

apply(Action, State, NewState):-
  strips_rule(Action, _, _, DeleteList),
  subtract(State, DeleteList, TempState),
  strips_rule(Action, _, AddList, _),
  union(AddList, TempState, NewState).
```

where `subtract/3` and `union/3` are predicates built-in in most Prologs. They define set subtraction and union. `subtract(+Set, +Delete, -Result)` deletes all elements of list `Delete` from list `Set`, resulting in `Result`. `union(+Set1, +Set2, -Result)` makes the union of `Set1` and `Set2`, removing duplicates and resulting in `Result`.

STRIPS represents possible states as nodes of a graph (Fig. 15.3). Each node consists of a list of facts. The actions enable movement from one node to another, adding or deleting facts when the preconditions are met. Knowing an initial and a final list of facts representing the initial state and the goal, the problem is stated as finding the action plan that modifies the world, adding or deleting facts so that the initial state is transformed into the final one. This is a search problem, where STRIPS traverses a graph to find the plan actions as follows (Nilsson 1998, pp. 376–379).

- Repeat while `Goals` are not a subset of the current `State`,
  1. Select a `Goal` from the `Goals` that is not already in the current `State`.
  2. Find a STRIP rule whose `Action` adds `Goal` to the current `State` and make sure that `Action` has not already been done.
  3. `Action` may not be possible if the `Preconditions` are not met. Therefore, use STRIPS to solve recursively `Preconditions`. This results in the intermediary state 1 `(InterState1)`.
  4. Apply `Action` to add and delete facts from the intermediary state 1. This results in the intermediary state 2 `(InterState2)`.
  5. Recursively solve the rest of `Goals`.

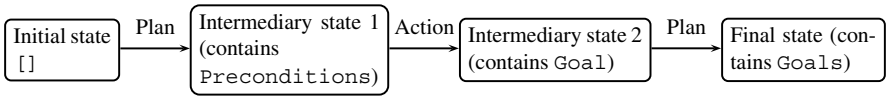

Fig. 15.3. The STRIPS schemata.

STRIPS in Prolog uses the initial state and the goals as inputs and produces the final state and the plan, where the goals must be a subset of the final state. We need some auxiliary variables to carry out the computation. The Prolog recursive rule builds the `Plan` in a reverse order, adding the current `Action` to the head of the list, and unifies it to the `FinalPlan` variable when the goal is satisfied. Prolog also builds `State`, and unifies it to `FinalState` in a same way. To avoid possible infinite loops when finding preconditions to an action, the rule keeps a copy of the corresponding plan and prohibits the repetition of actions.

```
strips(Goals, InitState, FinState, Plan) :-
  strips(Goals, InitState, [],[], FinState, RevPlan),
  reverse(RevPlan, Plan).

% strips(+Goals, +State, +Plan, +PrecondPlan,
```

```
%  -FinalState, -FinalPlan)

strips(Goals, State, Plan, _, State, Plan) :-
  subset(Goals, State).
strips(Goals, State, Plan, PrecondPlan, FinalState,
    FinalPlan) :-
  member(Goal, Goals),    %Select a goal
  \+ member(Goal, State),
  strips_rule(Action, _, AddList, _), %Find an action
  member(Goal, AddList),
  \+ member(Action, PrecondPlan),
      % Find preconditions
  strips_rule(Action, Preconditions, _, _),
  % Get the FirstPlan and InterState1
  % to achieve preconditions
  strips(Preconditions, State, Plan,
    [Action | PrecondPlan], InterState1, FirstPlan),
  % Apply Action to the world
  apply(Action, InterState1, InterState2),
  % From FirstPlan move forward
  strips(Goals, InterState2, [Action | FirstPlan],
    PrecondPlan, FinalState, FinalPlan).
```

15.6.3 Causality

Planning provides a good operation model for dialogue and for other discourse phenomena such as **causality**. Causality occurs when some sentences are logically chained and are part of a same demonstration. Consider, for instance, these sentences:

Hedgehogs are back. Caterpillars shiver.

The second sentence is a consequence of the first. Causality can be related to a logical demonstration but also depends on time-ordered events. Causal rules represent specific events, which will result in certain facts or effects. Usually, they also require preconditions. They can be expressed in Prolog using predicates whose structure is similar to:

```
causes(preconditions, event, effects).
```

As we can see, this is also closely related to planning. The `causes` predicate means that if the `preconditions` are met, and if an `event` occurs, then we will have `effects`.

Many sentences involve sequences of actions – plans – that are temporally chained. For instance:

Phileas the hedgehog was thirsty. He went out to have a pint.

These two sentences correspond to an action, which is followed by another, the second one being a consequence of the first one. For both examples, discourse understanding can be restated as a plan recognition problem.

15.7 Further Reading

Speech acts theory in dialogue is mostly known from the works of Austin (1962) and Searle (1969, 1979), although Bühler (1934) pioneered it. Searle and Vanderveken (1985) describe a logical model of illocutionary acts as well as a list of English verbs classified according to Searle's ontology. Vanderveken (1988) expands this work to French verbs. Foundations of belief and intention modeling in dialogue are due to Hintikka (1962). Carberry (1990) provides accounts to plan recognition in dialogue.

EVAR and the SUNDIAL projects have been a source of valuable research and publications about spoken dialogue processing. Bilange (1992) gives an excellent overview of dialogue processing techniques and application examples. Other works include those of Andry (1992), Mast (1993), Eckert (1996), and Sagerer (1990). The TRAINS project (Allen et al. 1995) is another example of elaborate dialogue processing. Many applications, such as train reservation systems, are now available commercially.

Planning includes a large number of applications and has spurred many algorithms. In computational linguistics, it occurs within the frameworks of temporal reasoning, intention modeling, and other forms of constraint-based reasoning. Bratko (2001) gives a short introduction to planning and a collection of Prolog programs. Russell and Norvig (2003) provides another introduction to planning.

Exercises

15.1. Write a dialogue program using Prolog clauses – no DCG rules – asking a couple of questions and accepting yes or no answers only. Collect all the answers and print them out at the end of the session.

15.2. Write a dialogue program using Prolog clauses – no DCG rules – reproducing the dialogue of Fig. 15.1. Collect all the answers and print them out at the end of the session.

15.3. Write verbs in a language you know corresponding to Searle's ontology of illocutionary classes: assertives, directives, commissives, declaratives, and expressives.

15.4. Rewrite Exercice 15.1 using SUNDIAL's speech act predicates in Table 15.8.

15.5. Rewrite Exercice 15.2 using SUNDIAL's speech act predicates in Table 15.8.

15.6. The DCG dialogue rules described in Sect. 15.5.3 are not robust. Make a parallel with sentence parsing and give examples where they would fail and why.

15.7. Modify the rules of Sect. 15.5.3 so that they would never fail but recover and start again.

15.8. Write an automaton in Prolog to model EVAR's main phases accepting a legal sequence of speech acts, such as `[S_GREETING, U_REQ_INFO, ...]`.

15.9. Modify the EVAR automaton of Exercice 15.8 to be interactive. Design questions and messages from the system and possible answers from the user. Replace the user and system turns with them.

15.10. Modify the EVAR automaton of Exercices 15.8 and 15.9 and use the SUNDIAL speech acts. Make the system work so that you have a more or less realistic dialogue.