

Diplomarbeit

# A Context Manager Exploiting Discourse Structure

CHRISTIAN HYING

Institut für Maschinelle Sprachverarbeitung  
Universität Stuttgart

In Zusammenarbeit mit

Man-Machine-Interfaces Group  
Sony International (Europe) GmbH  
Advanced Technology Center Stuttgart

Prüfer: Hans Kamp (IMS, Universität Stuttgart)  
Prüfer und Betreuer: Uli Heid (IMS, Universität Stuttgart)  
Betreuer: Sunna Torge (ATCS, SONY, Stuttgart)  
Jan van Kuppevelt (IMS, Universität Stuttgart)



## **Erklärung**

Hiermit versichere ich, dass ich meine Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Christian Hying



# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Problem . . . . .	3
1.2	Approach . . . . .	5
<b>2</b>	<b>General Description of the Dialogue System</b>	<b>7</b>
2.1	The Application Domain . . . . .	8
2.2	The User Task . . . . .	10
2.3	Task Complexity . . . . .	11
2.4	User Input . . . . .	12
2.5	The System Task . . . . .	16
2.6	System Output . . . . .	18
<b>3</b>	<b>Outline of the Extension</b>	<b>21</b>
3.1	An Example Dialogue . . . . .	21
3.2	Description of the Base System . . . . .	28
3.3	The New Architecture . . . . .	31
3.4	Summary . . . . .	35
<b>4</b>	<b>Basic Data Structures</b>	<b>37</b>
4.1	Feature Structures . . . . .	37
4.2	Modelling the Application Domain . . . . .	40
4.3	Communication about the Domain . . . . .	43
4.4	<i>syntactic/semantic</i> Representation . . . . .	44
<b>5</b>	<b>Structure of Dialogue</b>	<b>49</b>
5.1	Linguistic Structure . . . . .	50
5.2	Intentional Structure . . . . .	51
5.3	Attentional State . . . . .	59
5.4	Salience Structure . . . . .	61
<b>6</b>	<b>Anaphora Resolution</b>	<b>63</b>
6.1	Semantics of Nominals . . . . .	63
6.2	Anaphoric Expressions . . . . .	66
6.3	Anaphora Representation Structure . . . . .	67

## CONTENTS

---

6.4	Resolution of the Anaphoric Part . . . . .	69
6.5	Application of the Functional Part . . . . .	70
<b>7</b>	<b>Managing the Context</b>	<b>75</b>
7.1	Information State . . . . .	75
7.2	Intentions . . . . .	77
7.3	Information State Updates . . . . .	79
7.4	Summary of the Functions of the Context . . . . .	81
<b>8</b>	<b>Notes about the Implementation</b>	<b>83</b>
8.1	Parsing Module . . . . .	84
8.2	Anaphora Resolution . . . . .	85
8.3	Semantic Interpretation . . . . .	86
8.4	Context Manager . . . . .	86
8.5	Dialogue Manager . . . . .	87
8.6	Hints at Evaluation . . . . .	87
8.7	Summary . . . . .	88
<b>9</b>	<b>Future Work</b>	<b>91</b>
<b>10</b>	<b>Conclusions</b>	<b>99</b>
<b>A</b>	<b>Example Dialogue</b>	<b>103</b>

# Chapter 1

## Introduction

### 1.1 Problem

The problem to be solved by this diploma thesis is to add a dialogue memory to an existing dialogue system<sup>1</sup>. The task of this diploma thesis is ‘*to develop an appropriate concept for a dialogue memory, to implement the dialogue memory, and to exemplify the benefits of using the dialogue memory in the dialogue system*’.

The thesis starts off from a dialogue system which provides the functionality of a juke-box. The user can search a database of music recordings and select one piece which is then played back by the dialogue system. A typical session between a user and the dialogue system might look as follows:

- (1)
  - a. **User:** I want to listen to some *Jazz* from *Miles Davis*.
  - b. **System:** There are two titles: *Milestones* and *Decoy*.
  - c. **User:** Play *Milestones*.
  - d. The system plays the recording which has the genre *jazz*, the artist *Miles Davis*, and the title *Milestones*.

In this example the user specifies three distinct types of attributes which describe the recording he wants to listen to, namely the genre (*Jazz*), the artist (*Miles Davis*) and the title (*Milestones*) of the recording. The specification of these three attributes provides enough information to the dialogue system to uniquely identify a recording in the database.

Generally speaking, a dialogue memory is a component of a dialogue system whose purpose is to keep track of the dialogue and to provide relevant information to other components of the dialogue system. A dialogue memory can, for instance, provide the context to the *natural language understanding* unit of

---

<sup>1</sup>The dialogue system has been developed at SONY, ATCS, Stuttgart. For details, see Section 3.2

a dialogue system. Another example of an application of the dialogue memory is that it provides the context to an *interpretation unit* which determines the effect of a dialogue act (or of an utterance) on the task.

The *natural language understanding unit* (NLU) translates natural language expressions (e.g. utterances of the user) into semantic representations. The NLU needs information from the dialogue memory in order to resolve anaphoric expressions, such as pronouns (e.g. “he”), definite noun phrases (e.g. “the second artist”), and short forms of names (e.g. “Miles” as short form of the name “Miles Davis”). For instance, (2) shows a part of a dialogue which contains the pronoun “him” in utterance (2-c).

- (2) a. **User:** *Jazz* from *Miles Davis*.
- b. **System:** There are two titles: *Milestones* and *Decoy*.
- c. **User:** Is there *Pop* from **him**?

The context for understanding utterance (2-c) contains four objects: the titles *Milestones* and *Decoy* mentioned in utterance (2-b), the artist *Miles Davis* mentioned in utterance (2-a) and the genre *jazz* which is also mentioned in utterance (2-a). The pronoun “him” requires the antecedent to be male and singular which is only satisfied by the name of the artist “Miles Davis” occurring in utterance (2-a). Therefore, the actual denotation of the pronoun in (2-c) is the artist *Miles Davis*.

Other anaphoric expressions which are dealt with in this thesis are names, and definite noun phrases.

The *interpretation unit* determines the effect of an utterance on the state of the task. It can benefit from the information of the dialogue memory, when the user revises an attribute which he has specified already at an earlier stage of the dialogue.

The problem is to determine the effects of the revision. Does only the revised attribute have to be changed? Or are the value of some other attributes discarded, because they have been dependent on the former value of the revised attribute?

For example, (3) shows a the beginning of a dialogue in which the user revises one attribute. The user starts to search for a recording with the title ‘*Whiskey in the jar*’. The database contains more than one recording with that title, so the dialogue system offers the user to choose between two genres. In utterance (3-c) the user first specifies the genre *folk*, but since he is not satisfied with the choices presented to him in (3-d), he revises the genre to the second choice, which is *rock*.

- (3) a. **User:** Do you have the song *Whiskey in the jar*?
- b. **System:** Yes. Which genre would you like: *Rock* or *Folk*?
- c. **User:** *Folk*.
- d. **System:** There are two bands: *The Dubliners* and *The Pogues*.

e. **User:** I would like *Rock*.

At this point the dialogue system has two alternative ways to interpret the effect of utterance (3-e) on the state of the task. It can either mean that the user wants the system to forget about *folk*, but to keep the title “Whiskey in the jar”. In that case utterance (3-e) would yield a search for recordings with the title “Whiskey in the jar” from the genre *rock*. Or it can also mean that the user wants the system to forget about everything said so far. In that case utterance (3-e) would yield a search for recordings with the genre *rock*.

The first alternative takes into account that the attribute *genre* has been specified later than the attribute *title*. Therefore, the revision of the attribute *genre* does not affect the value of the attribute *title*.

The second alternative is an example of an approach where the values of some attributes are existentially dependent on the values of other attributes. For instance in the second alternative, it is assumed that the attribute type *title* existentially depends on the attribute type *genre*. Therefore, the title “Whiskey in the jar” is discarded, when the attribute of the type *genre* is revised.

In this thesis I propose a strategy which determines the way of interpreting the effect of an utterance based on discourse structure. (The first alternative would be taken by this strategy.) A discourse structure contains for example implicit information about the order of the utterances. The structure is provided by the dialogue memory.

## 1.2 Approach

The *context manager* implements the dialogue memory. It maintains an information state which models the information that is relevant in a dialogue at any given point (compare with dialogue systems of the TRINDI Project (Larsson and Traum 2000)). The implementation is described in Chapter 7. What the relevant information of a dialogue is, is determined by analysing the structure of the dialogue.

The structure of dialogues is analysed according to the theory of *discourse structure* proposed by Grosz and Sidner (1986). A dialogue is represented by a tripartite structure consisting of the structure of the actual utterances, the structure of the underlying intentions and a structure representing the focus of attention. Grosz and Sidner’s theory of discourse structure is reviewed in Chapter 5.

The computation of discourse structure relies on the coherence of an new utterance with the rest of the dialogue. The notion of coherence is based on simple semantics mapping natural language expressions onto lists of attribute-value pairs which are equivalent to semantic frames.

Anaphoric expressions are resolved by generalising the approach to resolve discourse deixis presented by Webber (1991). This generalised approach is combined with a matching algorithm operating on semantic representations.

Reviews of state-of-the-art approaches and state-of-the-art dialogue systems are distributed over the entire diploma thesis preceding the discussion of each major

component of the dialogue system. They can be found mainly in Chapter 2, Chapter 3, Chapter 5 and Section 6.2.

The diploma thesis describes the extension of the existing dialogue system. It is structured as follows.

The second chapter describes general properties of the dialogue system which hold for the existing system and for the extended system. Chapter 3 motivates the extensions and defines the target of the extension. Chapter 4 contains definitions of the basic data structures used in this thesis. It additionally contains descriptions of the structures used to represent the domain and linguistic expressions. Chapter 5 discusses the notion of discourse structure. The implementation of the anaphora resolution component is described in Chapter 6, and the implementation of the context manager in Chapter 7. The state of the implementation relative to the target is discussed in Chapter 8. In this chapter there are also a few remarks about an evaluation. Chapter 9 contains suggestions of how to proceed the implementation work and the development of the dialogue system. The final chapter briefly summarises this thesis.

The source code of the implementation has not been included, but Appendix A contains a transcription of the internal data structures used by the context manager to process an example dialogue (Table 3.1).

## Chapter 2

# General Description of the Dialogue System

The dialogue system JUKE provides the functionality of a juke-box; it serves as an interface to a database of music recordings and it controls a player application.

This chapter describes the dialogue system from a user's perspective; the description holds both for the basic system from which the diploma thesis starts off and for the extended system which is presented in the following chapters.

JUKE is fully controlled by natural language which the user types in via a keyboard. The system itself puts out natural language sentences and lists of database items on the display in order to communicate with the user.

JUKE originated from a speech understanding project (only available as internal report (Lucke 1998)) that focused on utterance processing, but did not investigate discourse phenomena. The work carried out within this thesis focuses the integration of contextual knowledge, in particular the knowledge of discourse context. The long term target of the development of JUKE is a multi-modal spoken language dialogue system which serves as an interface between users and a juke-box.

At the current stage, JUKE runs on a *Linux* PC or on a *Solaris* workstation. It requires the computer to have textual input and output facilities, for instance a keyboard and a text display.

The first section describes the application domain, the database containing music recordings. The second section outlines the task which the user has to carry out. Notes about the complexity of the user task can be found the third section. The input which the user can provide to JUKE is sketched in the fourth section. The last two sections describe the requirements to the system which ensure cooperative behaviour of the system with respect to solving the task and communicating with the user.

## 2.1 The Application Domain

The application behind the dialogue system consists of a music player and a database of files containing music data. Which entities constitute the basic entities of the dialogue system is mainly dependent on the content of the database and on the possible operations on the data. The domain of JUKE is similar to the one of the *EMBASSI* project (*EMBASSI Project 2002*) and (Bücher et al. 2001). This section describes the structure and the content of the database, the structure of the domain which is directly dependent on the database, and how the database is accessed.

**Database** The database of JUKE contains pointers to music recordings and additional information about each recording. It responds to queries with a set of items or with a set of particular attributes of items.

A database item is a tuple of five attributes:

*(id, genre, artist, title, file-location)*

Any item points at an audio file which stores a particular recording; the pointer is given by the last element of the tuple, i.e. *file-location*. The first element, *id*, contains a unique identifier of the item; thus, it can be used to represent the recording and the item itself. The other attributes express additional information about the recording; *genre* holds the name of the genre; *artist* holds the name of the composer, a performing artist, or the performing group; *title* contains the name of the title. Note, that this format does not support a distinction between, for instance, the composer and the performing artist.

The single attributes of the database items can be seen as having a denotation themselves. As said above, each unique identifier, i.e. the *id* element, refers to the particular database item which it is part of. A title name is the name of a recording, the title name by itself denotes the set of database items which contain exactly this title name. Accordingly, we have a notion of genres and artists: the genre (artist) denoted by a genre (an artist) name is the set of all database items which contain exactly this name in the genre (artist) slot.

**Domain Structure** JUKE's domain is a small music domain which shows little structure. *Domain structure* defines the dependencies between the information associated with the entities of the domain. Usually, it is obtained by decomposing the domain into a number of sectors which themselves may be hierarchically decomposed (Bernsen and Dybkjær 2000). In case the domain does not have a naturally given structure, one may impose structure onto it by inventing an appropriate hierarchy. Examples of invented hierarchies for music domains can be found in record shops, where the customer first has to find the top category, which is usually the genre, then the artist, and only then he can look for a particular title.

JUKE's domain comprises the following entities: one kind of object, three kinds of properties and one kind of action. First, *recordings* constitute the objects of the domain; second, there are *titles* which are the names of recordings; furthermore, there are names of *genres* and *artists*. As said above, genres and artists are represented by sets of database items. Third, the only action included in the domain is the action *play*, i.e. playing a recording.

The only domain structure that is given by the format of the database is that each recording is associated with exactly one title name, exactly one artist name, and exactly one genre name. We do not impose any other dependencies onto the domain; the properties of recordings, namely titles, artists and genres, are independent from each other.

**Queries** The entire database access can be formalised by the following predicate:

**query**(*type*, *recording*, *title*, *artist*, *genre*)

No argument can have more than one value; it is either unspecified, or filled with a symbol or a string. The first argument determines the format of the result. If *type* is one of the following five symbols `file`, `recording`, `title`, `artist` and `genre`, it only returns that element of the database item which corresponds to the symbol. In particular that means that the result is a list of file pointers, unique identifiers, title names, artist names, and the genre names, respectively. The other arguments of the query define the *search criteria*. If *recording* is specified as the symbol `x`, only those items are matched that have the unique identifier `x`; if *title* is specified as the string `"x"`, only those items are matched that have the title name `"x"`. The arguments *artist* and *title* are interpreted similarly.

The search criteria are always interpreted conjunctively. Disjunctive queries are not considered yet; it is not possible to formulate search criteria negatively, either. For instance, one cannot formulate a query that asks for 'everything except for jazz'.

In the following the search criteria are represented as a set of binary tuples. The first element of each tuple contains the type of a single search criterion and the second element the value. Types correspond to the above representation of database queries. They are taken from the symbols `recording`, `title`, `artist` and `genre`. For example, if the database is queried with the following search criteria,

{ `<title, "Yesterday">`, `<artist, "The Beatles">`, `<genre, rock>` }

it returns all database items that are classified as rock, have the title *Yesterday* and are associated with the band *The Beatles*.

## 2.2 The User Task

The user uses JUKE to select a title from the database in order to listen to it. The structure of the user task depends on the goals of the user and how they can be achieved (Flycht-Eriksson 1999). This section gives a description of the user task including the task structure which represents all possible sequences of possible operations the user may intend to carry out.

Apart from communicating with dialogue system, the user has to specify search criteria in order to navigate through the database. The search is finished when he has specified a particular recording. In general, the user can specify the following four search criteria: genre, artist, title, and recording. But there, he encounters two problems: first, he does not know the contents of the database; and second, he cannot directly refer to a recording by means of a name or a description. The only way to pick out a particular recording is to use its unique identifier, but unique identifiers are only for internal uses. So the user does not have direct access to them.

However, the user can indirectly refer to recordings by referring to entities presented by the dialogue system in earlier steps. Therefore, the user relies on the communication with the dialogue system in order to fulfil his task.

**Communication** JUKE uses two communication channels, whereas the user only uses one. The user is restricted to communicate with JUKE by natural language. The system addresses the user via natural language sentences which are displayed on the screen; additionally, it can display numbered lists. Usually, these lists show attribute values of database items, for instance, title names.

**Possible Operations** As noted above, the user can specify the following four search criteria: genre, artist, title, and recording. He can specify more than one criterion in each step. The specifications are interpreted relative to the context and lead to an update of the search criteria of the previous step. There are three different kinds of update: the new specification may replace all previous search criteria; it may overwrite some of them; or it may be added to them.

Although the user can specify the search criteria which determine the query result of the database, he cannot control action of what is done with the result of the database query. That means, that JUKE controls the presentation of the query result. For example, if the user asks for “all recordings of the genre jazz”, JUKE is free to present the artist names that correspond to the query result.

If the current search criteria uniquely specify a recording, it is played back. In all other cases JUKE presents the query result in an appropriate format to indicate to the user possible continuations which further restrict the search criteria.

**Task Structure** The task is *ill-structured* (Bernsen and Dybkjær 2000) because of the lack of domain structure; there is no naturally given hierarchy which orders the required information. Therefore, the user is free to decide which information he passes to the system at any point of the interaction; he does not

have to follow a particular order, nor does he have to follow the suggestions of the system in every step. Nevertheless, as noted above, he must pick up a suggestion of the system in case he wants to specify a recording.

The user task splits up into single steps. The task associated with each step is to specify search criteria. The last step is determined by the user specifying a recording.

## 2.3 Task Complexity

Before JUKE can play any music, the user has to fulfil his task, that is to specify a particular recording. JUKE tries to assist him by suggesting search criteria which are in line with his intentions. The term *Task Complexity* describes the complexity of the task itself, it does not include the complexity of the communication task. Task complexity is important for the design of dialogue systems, because interaction strategies and data structures which represent the relevant information of the dialogue and the task, have to be chosen according to its complexity, see (Bernsen et al. 1994) and (Allen, Byron, Dzikovska, Ferguson, Galescu and Stent 2001).

The complexity of the task is dependent on (i) the volume of information (i.e. the number of dialogue acts), which has to be exchanged between the user and the system, (ii) the structure of the task (ill structured vs. well structured) and (iii) the extent of mixed initiative which is involved Bernsen and Dybkjær (2000).

First, JUKE's task requires only the exchange of one piece of information, namely the information about one particular recording. But additionally, an optional information exchange takes place; the user gives information to the system regarding the search criteria; and the system informs the user about the content of the database. Thus, the volume of information which is exchanged between the user and the system directly depends on the structure and the content of the database. Since the structure is simple and size of the database of the current implementation of JUKE is rather small, the number of dialogue acts that have to be exchanged is also small.

Second, as noted in the previous section, the task is ill-structured. This means for the system that it has to expect any possible operation from the user at any time. Therefore, the task complexity correlates with the number of possible operations the user can specify. Since the task does not contain subtasks and JUKE does only support four possible operations, the task structure represents low overall task complexity.

Third, there is only little mixed initiative. JUKE rather resembles a command and control dialogue system than a free negotiation dialogue system. It is user directed, the system only reacts on the requests from the user. If the user specifies criteria which do not match anything in the database JUKE rejects the specification immediately. Otherwise, if the criteria match any entries, the user is immediately informed about the new query result. Thus, there is no need to

design complex domain plans, nor to consider the hidden beliefs, desires and intentions of the user.

All in all, the task itself shows a low complexity which is comparable to the tasks of the dialogue systems *EVAR* (Gallwitz et al. 1998), *OVIS* (van Noord et al. 1998) and the *Philips System* (Aust et al. 1995) all of which provide information about train time tables, and to the dialogue system *WHEELS*, which is an interface to a database of automobile classifieds (Meng et al. 1996).

## 2.4 User Input

The user controls JUKE via natural language. The user specifies search restrictions, describing them by means of names and properties, and by picking up suggestions from the display; in the latter case the user specifies the position of the suggestion which is displayed on the screen.

This section describes the fragment of natural language which qualifies as sensible possible user input. Most dialogue systems support spoken language as input, many recent research projects in dialogue systems aim at enabling multi-modal input, such that the system interprets the mimics, gestures, and other physical actions of the user, e.g. SMARTKOM (Wahlster et al. 2001). JUKE can only interpret a small fragment of natural language.

The user input is restricted in length and vocabulary. Any contribution the user enters into the system has to contain exactly one *utterance* which either consists of a full sentence or a nominal phrase. The following grammar rule shows the structure of an utterance. The question marks indicate optionality.

$$\text{UTT} \rightarrow (\text{GARB})? (\text{FP})? \text{OBJ} (\text{GARB})?$$

At the beginning and at the end of an utterance the user can include words which are taken to be *Garbage* and do not influence the analysis of the utterance. The rest of the utterance consists of a functional phrase (FP) containing subject and verb, and the the nominal object (OBJ). The object is mandatory for every utterance.

The examples below show valid utterances corresponding to the above rule:

- (4)
- a. jazz
  - b. jazz please
  - c. play jazz please
  - d. please play jazz

The structure of the categories GARB, and FP and OBJ is described by rules and examples in the remainder of this section.

The garbage category (GARB) only contains one word.

$$\text{GARB} \rightarrow \textit{please}$$

The category FP (*functional phrase*) contains all words of the sentence which do not belong to the object. In particular it contains the subject and the verb, it determines the sentence type and the tense of the sentence.

Possible strings of the functional phrase are defined by the following rule. The curly brackets indicate alternative possibilities of strings.

$$\text{FP} \rightarrow \left\{ \begin{array}{l} \left\{ \begin{array}{l} \textit{is there} \\ \textit{do you have} \end{array} \right\} \\ \\ (\textit{can you})? \left\{ \begin{array}{l} \textit{show me} \\ \textit{display} \end{array} \right\} \\ \\ \left\{ \begin{array}{l} \textit{can I} \\ \textit{I want to} \\ \textit{I would like to} \end{array} \right\} \left\{ \begin{array}{l} \textit{have} \\ \textit{take} \\ \textit{listen to} \\ \textit{hear} \\ \textit{see} \\ \textit{choose} \\ \textit{select} \end{array} \right\} \\ \\ I \left\{ \begin{array}{l} \textit{want} \\ \textit{take} \\ \textit{choose} \\ \textit{select} \end{array} \right\} \end{array} \right\}$$

The following phrases show valid FP's according to this rule:

- (5)    a. is there  
       b. show me  
       c. I would like to listen to  
       d. I take

The first phrase (5-a) is an interrogative sentence; (5-b) introduces an imperative sentence and (5-c) and (5-d) are FP's for declarative sentences.

The category OBJ represents the nominal object of a sentence. Before we take a look at the rules that define the phrase OBJ we give the following examples for OBJ phrases:

- (6)    a. jazz  
       b. the second one  
       c. My Funny Valentine from Miles Davis  
       d. the second jazz recording My Funny Valentine from Miles Davis

These examples show that it is possible to use simple phrases in order to specify information (6-a), but also to put much information into one nominal object (6-d).

The following rules give a more detailed description of the coverage of the category OBJ. Alternative right-hand-sides of the rules are separated by semicolons. Behind each alternative there is given at least one example.

OBJ →	OBJPRO;	it, him, her
	DP;	some music
	DP PP;	some music with Miles Davis
	PN;	Miles Davis, jazz, My Funny Valentine
	PN PP;	Miles Davis from jazz
DP →	InPRO;	something, anything, everything
	(XDET)? { ADJ } PN }	some jazzy music jazzy music the Miles Davis recording the jazz title My Funny Valentine
	XDET { ADJ } PN }	a jazzy one
	(XDET) N CARD	the Miles Davis one the number three genre one
XDET →	<i>the</i> ORD;	the second, the first, the last
	DET	a, the, some
PP →	P PN;	from Miles Davis
	P OBJPRO;	by him
	P <i>the</i> ORD <i>one</i>	with the third one

The category OBJPRO denotes personal pronouns occurring as an object. The category P stands for one of the following prepositions: *of*, *with*, *by*, and *from*. The category InPRO denotes indefinite pronouns, such as *something*, *anything*, *etc.* Adjectives are represented by the category ADJ.

Proper names and short forms of proper names are subsumed by the category PN. This category directly correlates with the attributes of the database items. The values for genre, artist and title are analysed as proper names. The next rule only shows a tiny part of the actual rule:

$$\text{PN} \rightarrow \left\{ \begin{array}{c} \textit{jazz} \\ \textit{rock} \\ \dots \\ \textit{MilesDavis} \\ \textit{Miles} \\ \textit{Davis} \\ \dots \\ \textit{My Funny Valentine} \\ \dots \end{array} \right\}$$

The next rule contains a complete listing of the nouns (category N) which can be used in an utterance:

$$\text{N} \rightarrow \left\{ \begin{array}{c} \textit{music} \\ \textit{style} \\ \textit{genre} \\ \textit{artist} \\ \textit{title} \\ \textit{recording} \\ \textit{singer} \\ \textit{group} \\ \textit{band} \\ \textit{composer} \\ \textit{musician} \\ \textit{song} \\ \textit{cd} \\ \textit{record} \\ \textit{album} \\ \textit{item} \\ \textit{number} \\ \textit{entry} \end{array} \right\}$$

**Interpretation** The nominal objects receive an interpretation relative to the database. The parser augments the analysis of nominal objects with semantic information which determines the type of the denoted object. In particular, names of genres come with the information `TYPE = genre`. Names of artists and names of titles specify the type of the denotation to `artist` and `title`, respectively.

Therefore, we can interpret some nominal objects directly as a set of search criteria. For example the phrase

“some jazz from Wynton Marsalis”

can be interpreted as the following set of search criteria:

{(genre, jazz), (artist, “Wynton Marsalis”)}

The analysis and representation of nominal phrases will be presented in Section 4.4. Much more about the interpretation of nominal phrases will be given in Chapter 6.

## 2.5 The System Task

JUKE mediates between the user and the database. Its task splits up into three subtasks: the communicative task which is managing the communication with the user, the domain task which is keeping track of the user task, and controlling external applications (Flycht-Eriksson 1999).

**Communicative Task** The communicative task of the system is to act cooperatively in conversation. This comprises understanding the user, managing *discourse obligations* (Traum and Allen 1994) and (Bunt 1997), *turn taking*, e.g. (Bunt 1994), and *grounding* (Matheson et al. 2000) the process of adding information to the common ground. JUKE does not model the communication task explicitly, it rather solves the communication task by sticking to design principles for the dialogue manager and assumptions about the user (Bernsen et al. 1996), (Bernsen and Dybkjær 2000). This means that JUKE does not support clarification meta-communication in case of communication failure, the responsibility for ‘smooth’ communication has been shifted to the user.

Before the user uses JUKE he is informed about the words, the content and the structure of the utterances which the system can understand. In order to cope with turn management he is told that he holds the initiative all the time during the dialogue, but that he has to wait until the system has given some output.

JUKE supports the user in behaving cooperatively by giving *Feedback* and by using *Priming*. First, the user should be able to recognise immediately whether his intentions have been understood correctly. Therefore, the system displays the state of the task, so that the user can observe the effect of his utterances. Second, since the user is likely to pick up expressions from the system, the system should only output words, database items and sentences which it can understand itself. This sort of priming reminds the user of the words which the system can understand.

*Grounding* is dealt with by JUKE applying a strict interaction model. During the entire dialogue JUKE only specifies an intention once, namely at the beginning of the dialogue. This intention introduces the general task into the dialogue and it is assumed that the user accepts it, because it is the purpose of the whole dialogue. JUKE does not bring in any more intentions into the dialogue. All other new intentions are only introduced by the user. If the system accepts them they are taken to be common ground.

In order to manage *discourse obligations* JUKE does not put out sentences which would require the user to react in a particular way. Therefore, the output of the

system is restricted to two dialogue acts; *reject* the preceding intention and *inform* the user about the database. Only in the initial utterance the system is allowed to specify a *info-request* which serves as the basis of all subsequent requests from the user. The user is restricted to dialogue acts with a forward looking function (Core and Allen 1997), which JUKE always interprets as an *info-request*, i.e. a request for information.

Any dialogue system has to cope with taking initiative and reacting on the initiative of the user. JUKE always passes the initiative to the user. In the first utterance JUKE fixes the task which the user and the system are supposed to solve jointly. Then, it only responds to the user's requests. These measures reduce the complexity of the communicative task by making the dialogue excluding the first utterance fully *user directed*.

**Dialogue Model** The dialogue model generically describes how a dialogue can be constructed, see (Flycht-Eriksson 1999). According to the restrictions spelled out in the preceding paragraph we obtain the dialogue model for JUKE which is presented in figure 2.1.

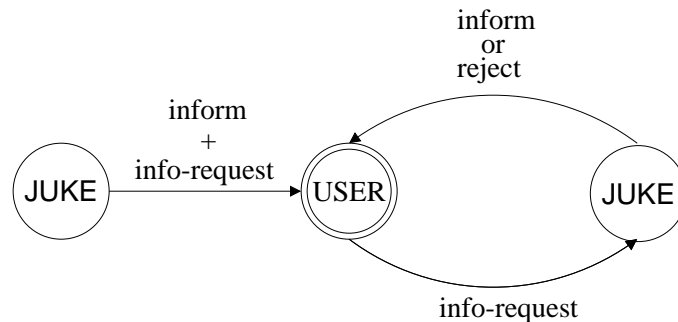


Figure 2.1: JUKE's dialogue model

The labels of the states indicate who has the turn; the transitions are labelled with communicative functions from the system's perspective; the double circle marks an end-state. JUKE starts the dialogue by stating the overall task and informing the user about possible options. It cannot control the end of the dialogue, because it always passes the turn back to the user. If the user wants to finish the session he should stop providing input to the system.

**Domain Task** The restrictions of the communicative task reduce the complexity of the overall system task to the *domain task* which includes (i) the interpretation of the user's utterances with respect to the domain task, (ii) controlling the database and the music player, and (iii) the production of appropriate output, which informs the user about the state of the task.

The task of understanding the user with respect to the domain task is limited to the utterances which are of the form described in Section 2.4. The first step

is carried out by the parser. But since the analysis of the parser still contains data structures which have to be resolved, for instance pronouns and definite noun phrases, these expressions have to be resolved in a second step. The last step is to understand the meaning of the utterance relative to the context. If it is understood, the system is able to compute the meaning of the dialogue.

The task of controlling the applications is part of the dialogue manager's duties. It queries the database, processes the results, and it controls a music player. The actions are carried out according to the state of the task and the state of the dialogue.

The system needs to produce cooperative output. This includes feedback on what the system has understood, feedback on what the system thinks the state of the task is, and assistance hinting at what the user can sensibly do in order to reach his goals. The output should conform to the principles and the dialogue model presented above.

## 2.6 System Output

The system puts out natural language sentences and numbered lists of database items. The output serves three functions: it informs the user about the current state of the task; it presents possible continuations of the task; and in case the last utterance of the user has to be rejected, the output should indicate that. This section gives a short overview of the behaviour of JUKE .

**Natural Language Output** The system displays natural language sentences to respond to the utterances of the user. Although questions require different responses than statements, JUKE does not distinguish between different types of sentences and speech acts. Everything the user contributes to the dialogue is interpreted as a request for information. Thus, the system complies with the request informing the user about the current state of the task, or it rejects the request by explicitly rejecting the state which the user intended to reach. For example, if the user and JUKE have agreed on the genre *hip-hop* and the user further requests recordings from *Frank Sinatra*, the system might either respond positively by

(7) Here are the hip-hop recordings from Frank Sinatra.

or it might reject the request by

(8) There are no hip-hop recordings from Frank Sinatra.

**Display** Apart from displaying natural language sentences JUKE is able to present numbered lists of text. The results of the database query are usually presented in this form. For example, after the user has told the system that he wants to listen to jazz, the system might reply with the following output:

- (9) “You can choose an artist from jazz.”
1. John Coltrane
  2. Miles Davis
  3. Klaus Graf
  4. Nils Lundgren
  5. Wynton Marsalis
  6. Maceo Parker

This output consists of a natural language sentence which serves as feedback and a numbered list which provides assistance.

**Feedback** The user must have the opportunity to verify that the system has understood the user’s input correctly (Bernsen and Dybkjær 2000). Ideally, the user also receives feedback on what the system believes the current state of the dialogue is. JUKE informs the user only about the state of the task which is equivalent to the current search restrictions. In (9) the natural language sentence paraphrases the state of the task, i.e. the database query.

**Assistance** The user has been given detailed instructions of how to use JUKE , but he is not (fully) aware of the contents of the database. By showing portions of the database JUKE suggests search criteria to the user which he can accept. Through this way the user gets to know the content of the database. The above example (9) shows a numbered list containing the artist names which correspond to the database query result. This list informs the user about possible search criteria which narrow down the current database search. Anyway, after the system responded with (9), the user can accept one of the suggestions, but he is also free to specify any other search criteria.



## Chapter 3

# Outline of the Extension

The description of the dialogue system JUKE that has been given in the previous chapter applies to both versions of JUKE: the old version (henceforth BASEJUKE), and the new version which has been developed in this framework (for this version we keep the name JUKE).

This chapter motivates the extension of BASEJUKE to JUKE. The main difference between both systems is that JUKE employs an explicit representation of the dialogue context. Thus it is capable of processing discourse related phenomena such as anaphora resolution. BASEJUKE lacks such an explicit representation. It implicitly stores the state of the task by means of a few variables. Therefore, BASEJUKE rather processes each utterance separately.

The first section gives an example dialogue and discusses the phenomena which typically require the capability of discourse processing. The second section describes BASEJUKE's utterance processing and discusses how it handles the discourse phenomena presented in the first section. The third section states the requirements for handling the phenomena and presents an architecture which is more suitable for discourse processing than the one of BASEJUKE.

### 3.1 An Example Dialogue

This section presents and analyses an invented example dialogue between a user and the dialogue system JUKE.

The dialogue is analysed with respect to its information structure and the interpretation of anaphoric expressions. The semantic content and the state of the task will be given in terms of search criteria (compare with Chapter 2).

#### Dependency of Information

By *dependency of information* we mean the dependency of one piece of information on another. A piece of information that depends on another piece of information is only valid concerning the meaning of the dialogue if this other

### 3.1. An Example Dialogue

User	System
(1)	Select a recording and I will play it for you. Here is a list of genres. <div style="border: 1px solid black; padding: 2px; display: inline-block;">           1. jazz   2.classical            3.pop   4.rock         </div>
(2) Is there anything from Maceo.	
(3)	I don't have an artist called Maceo. <div style="border: 1px solid black; padding: 2px; display: inline-block;">           1. jazz            2.classical            3.pop            4.rock         </div>
(4) I want a jazz song.	
(5)	Please, choose a jazz artist. <div style="border: 1px solid black; padding: 2px; display: inline-block;">           1. John Coltrane            2. Miles Davis            3. Klaus Graf            4. Nils Lundgren            5. Wynton Marsalis            6. Maceo Parker         </div>
(6) Maceo.	
(7)	I have these jazz recordings from Maceo Parker. <div style="border: 1px solid black; padding: 2px; display: inline-block;">           1. Black Widow            2. Coin Toss            3. Dial Maceo            4. My Baby Loves You         </div>
(8) Show me the second artist.	
(9)	Choose a jazz recording from Miles Davis. <div style="border: 1px solid black; padding: 2px; display: inline-block;">           1. Blue In Green            2. Milestones            3. John McLaughlin            4. Decoy         </div>
(10) Is there some pop from him?	
(11)	These are the pop recordings from Miles Davis. <div style="border: 1px solid black; padding: 2px; display: inline-block;">           1. All Of You            2. Some Day My Prince Will Come            3. My Funny Valentine         </div>
(12) Play the last one.	
(13)	OK. < play md6 >

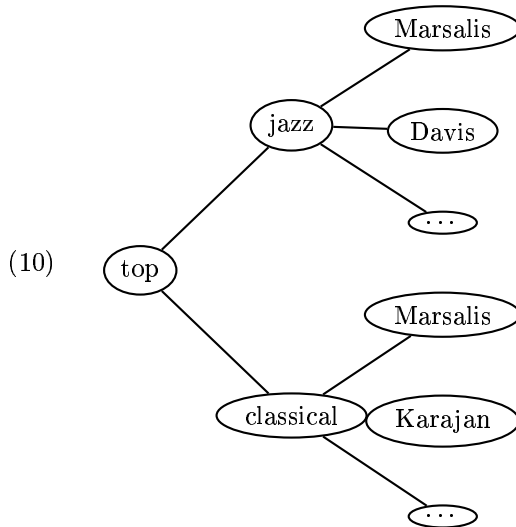
Table 3.1: Example Dialogue

piece is valid, too. The information of a new utterance which has been uttered in a dialogue at some point, can be dependent on the information that has already established, in such a way that the information of the new utterance is valid as long as the established information is valid.

In the dialogue shown in table 3.1 each user utterance contains information which aims at updating the search criteria. As presented below, the dialogue shows three possible ways of updating the state of the task with the new information. The new information is either (i) rejected by the system, or (ii) it is added incrementally to the current state, or (iii) it overwrites some parts of the current state. The dependency relation is important for the latter. Not only that one piece of information is overwritten by the new information that specifies the same search criterion, but also that all the dependent information is deleted.

**Static Dependency** Often the dependency relation is given by means of *domain structure*. In such cases the relation is statically determined holding between the distinct types of information.

In order to clarify the concept of static dependency of information, let us assume that the domain is structured as a hierarchy where the information about the artist depends on the information about the genre. An example of such a static hierarchy is given below:



This hierarchical domain distinguishes between the genres *jazz* and *classical*. The genres are further divided into artists.

Further assume that the user has already specified a particular genre and a particular artist, let's say *jazz* and *Marsalis*, respectively. If he then changes the genre to *classical*, all dependent information is abandoned. That means that the information that the artist is *Marsalis* is lost.

The use of static dependency is suitable for well structured domains (Bernsen

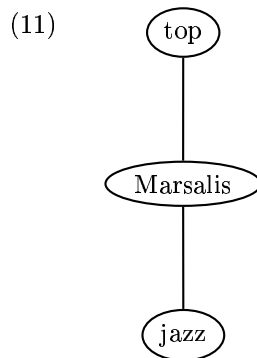
### 3.1. An Example Dialogue

---

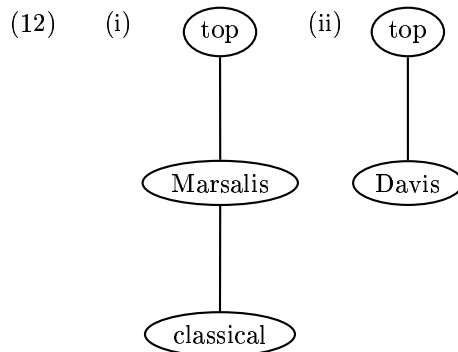
and Dybkjær 2000), but as noted in section 2.1 JUKE deals with an *ill-structured* domain. Therefore, we have decided to determine the dependency relation dynamically.

**Dynamic Dependency** We base the dependency relation of information on the dialogue itself. That means that the relation is determined dynamically based on the dialogue history - information that has been specified later is dependent on information that has been specified earlier. The user determines the order of the hierarchy implicitly while he is specifying search criteria. Therefore, it is not only possible to obtain the above hierarchy, see example (10), but also any other.

For example, if the user has first specified the artist *Marsalis* and then the genre *jazz*, then the information about the genre is dependent on the information about the artist.



Thus, changing the genre to *classical* only changes the genre, the information on the artist is kept, see (12-i). And if he further changes the artist to *Davis* the information about the genre is abandoned, (12-ii).



#### State of the Task

Let's now analyse the example dialogue according to the current state of the task. How do the user's utterances modify the search criteria?

When the dialogue starts, no search criteria are specified; the search space ranges over the entire database. The state of the task is represented by the empty set:

$$\{\}$$

**Rejection** If the system rejects an utterance of the user, it causes the information of the utterance not to be taken into the search criteria. In utterance (2) the user restricts the search space to items that contain an artist named “Maceo”; recall that the information that “Maceo” is a name of an artist is provided by the grammar. The state of the task is updated to the following search criteria:

$$\{\langle \text{artist, “Maceo”} \rangle\}$$

The system rejects this new piece of information by uttering utterance (3). It is rejected, because the database does not contain any item that matches this particular criterion (i.e.  $\langle \text{artist, “Maceo”} \rangle$ ). Note, that the database request with the above search criterion does not yield a result, because the database requires exact matching (compare with paragraph *Queries* on page 9 in Section 2.1. The following paragraph with the title *Addition* and the paragraph about anaphoric expressions on page 26 show successful use of the linguistic expression “Maceo”.

Anyway, the rejection causes restating the preceding state. So, the state of the task is reset to the empty set again:

$$\{\}$$

**Addition** The utterances (4) and (6) contain information which is added incrementally to the information that has already been established. In utterance (4) the user restricts the search space to items with a genre named “jazz”; then in utterance (6), he further restricts it to items that have an artist called “Maceo Parker”. The difference between both occurrences of “Maceo” in utterance (2) and utterance (4) will be explained below on page 26. After these two steps the state of the task is represented by the following search criteria

$$\{\langle \text{genre, jazz} \rangle, \langle \text{artist, “Maceo Parker”} \rangle\}$$

**Overwriting** Before the information of the utterances (8) and (10) is added to the search criteria, the system implicitly rejects some parts of the information which has already been established. The utterance (8) specifies the information that the artist should be named “Miles Davis” (see below). This information overwrites the previous information about the artist, so that the current search criteria restrict the search to database items with the genre named “jazz” and the artist named “Miles Davis”; or formally,

$$\{\langle \text{genre, jazz} \rangle, \langle \text{artist, “Miles Davis”} \rangle\}$$

Accordingly, the information of the utterance (10) overwrites the information about the genre. The new search criteria specify the genre “pop” and the artist “Miles Davis”:

$$\{\langle \text{genre, pop} \rangle, \langle \text{artist, “Miles Davis”} \rangle\}$$

Although the search criterion  $\langle \text{artist, “Miles Davis”} \rangle$  does not seem to be affected by the utterance (10), it is strictly speaking overwritten, for the pronoun “him” takes up the denotation “Miles Davis”.

The dialogue ends with the user selecting a unique database item. In utterance (12) he selects the recording with the unique identifier `rec_12345`. After the music player has been invoked with this recording, the task is finished.

### Anaphoric Expressions

The user’s utterances of the dialogue in table 3.1 contain pronouns, names, short forms of names and definite noun phrases. As many have pointed out, all these kinds of expressions can be interpreted as anaphoric expressions, (see (Krahmer and Piwek 2000) for an overview over anaphora).

Computing the meaning of an anaphoric expression is very much dependent on finding its antecedent. In case no suitable antecedent can be found in the context, (i) the expression does not have a meaning, or (ii) some suitable object has to be accommodated into the context (in the sense of (Lewis 1979)), or (iii) the expression receives an alternative interpretation. Here, we pursue the first and the latter approach. Pronouns which cannot be resolved do not get an interpretation. If names, their short forms, and definite descriptions cannot be resolved, they receive an exhaustive interpretation (see for example van Rooy (2001)). That means, they denote all objects which match the descriptive content of the expression.

Pronouns, names, short forms of names and noun phrases are used here to specify search criteria. Thus the denotation of anaphoric expressions is given in terms of search criteria (*cf* Section 2.1).

**Pronouns** The pronoun “him” is part of the indefinite expression in utterance (10). It is anaphoric on the name “Miles Davis” from the previous system utterance (4). The denotation of the pronoun is given by the set

$$\{\langle \text{genre, jazz} \rangle, \langle \text{artist, “Miles Davis”} \rangle\}$$

**Names and Short Forms** In the context of JUKE names and their short forms are taken to stand substitute for search criteria. We follow the analysis of names from van der Sandt (1992) and treat names as anaphoric expressions, such that they pick up the search criteria conveyed by that name at an earlier stage of the dialogue.

In the domain of JUKE the single attributes of the database items qualify as full names. That means that genre names, artist names, title names are *full names*. Short forms of names are all contiguous parts of those full names. For instance, the genre “disco pop” can be named by the short forms “disco” and “pop”. The felicity of using the short form is very much dependent on the situation; if there is no full name in the context which serves as an antecedent to such an short form, the short form cannot be resolved to a full name. Utterance (2) shows such a failure of the use of an short form, whereas it utterance (6) shows a felicitous usage.

In utterance (2) the short form cannot be resolved to a full name. Thus, it receives the alternative interpretation for names and their short forms. The descriptive content conveyed by the expression is interpreted exhaustively. Thus we obtain the search criteria

$$\{\langle \text{artist, “Maceo”} \rangle\}.$$

In utterance (6) the short form “Maceo” is matched by the full name “Maceo Parker” which is provided by the context at that point of the dialogue. In utterance (6) the name stands for the search criteria

$$\{\langle \text{genre, jazz} \rangle, \langle \text{artist, “Maceo Parker”} \rangle\}$$

In utterance (4) there is an occurrence of the full name “jazz”. The first genre name displayed in utterance (3) serves as its antecedent; it simply denotes the search criteria

$$\{\langle \text{genre, jazz} \rangle\}.$$

**Non-anaphoric Uses of Names** The natural language understanding component of JUKE allows the user to combine names and determiners to determiner phrases. In such cases the determiner simply overwrites the implicit determiner of the name and the name loses its own anaphoric force. While the definite determiner reassigns the anaphoric force to the name, indefinite determiners cause non-anaphoric uses of names. For instance, the expression “some Miles Davis” only specifies the search criterion  $\langle \text{artist, “Miles Davis”} \rangle$ ; likewise, the expression “some pop” denotes the criterion  $\langle \text{genre, pop} \rangle$ . In utterance (10) the denotation of the whole determiner phrase receives the following denotation after the pronoun “him” has been resolved as described above.

$$\{\langle \text{genre, pop} \rangle, \langle \text{artist, “Miles Davis”} \rangle\}$$

**Definite Noun Phrases** The first definite noun phrase “the second artist” in utterance (8) relates to the list on the display which has been presented to the user in the preceding step. It denotes the following search criteria:

$$\{\langle \text{genre, jazz} \rangle, \langle \text{artist, “Miles Davis”} \rangle\}$$

The second definite noun phrase, i.e. “the last one”, occurs in utterance (12). It denotes the following search criteria which uniquely specify a recording:

$$\{\langle \text{recording, rec.12345} \rangle, \langle \text{genre, pop} \rangle, \langle \text{artist, "Miles Davis"} \rangle, \langle \text{title, "My Funny Valentine"} \rangle\}$$

## 3.2 Description of the Base System

This section describes the starting point of the implementation that I have carried out in the framework of the diploma thesis. I will refer to the original state of JUKE by the name BASEJUKE. BASEJUKE provides the same task to the user as JUKE, so all descriptions that have been given about JUKE in chapter 2 also hold for BASEJUKE.

**Information Flow** BASEJUKE processes the input of the user in a pipeline as presented in figure 3.1. Since this processing model has been developed in the first place to work with spoken language input, it aims at providing an adequate treatment of ambiguity and uncertainty in utterances (Souvignier et al. 2000).

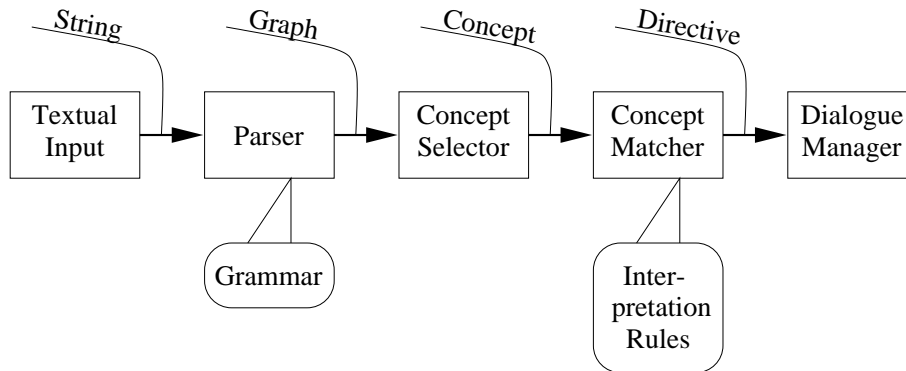


Figure 3.1: Input flow of BASEJUKE the base system

The input processing model of BASEJUKE can be divided into five processing parts each of which is represented by a box in figure 3.1. The results of each processing step which are passed to the next step are represented by the annotations of the arrows.

The first part reads in the *Textual Input* from the user. It puts out a string which is handed over to the second part. The *Parser* analyses the string according to a context free grammar. A hypothesis of a successful parse is represented by a feature structure which can be compared to *f-structure* of *Lexical Functional Grammar* (Kaplan and Bresnan 1982). The parser puts out a graph of alternative hypotheses which represent alternative interpretations of the input. The *Concept Selector* (*viz* the third part) selects one hypothesis from the graph. This hypothesis is a feature structure which is called the *Concept* that is con-

veyed by the input. The interpretation of the concept takes place in the *Concept Matcher* - the fourth part. The concept matcher tries to apply a set of *Interpretation Rules* to the concept; if a rule can be applied to the concept, it yields a *Directive* which the concept matcher sends to the final part, i.e. the *Dialogue Manager*.

**The Dialogue Manager** The dialogue manager processes the incoming directives. The structure of the dialogue manager is presented in figure 3.2.

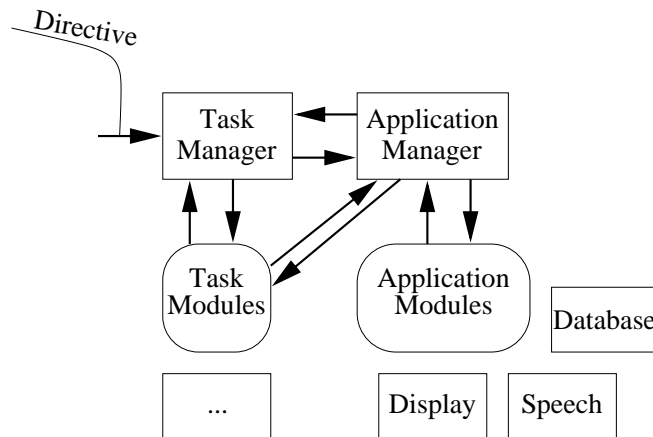


Figure 3.2: Structure of the Dialogue Manager

The dialogue manager consists of the *Task Manager* which attends to the user task and to the communication task, and of the *Application Manager* which controls the external applications and the output channels.

The main task of the task manager is to identify the incoming directive and to pass it to an appropriate *Task Module*. The task module initiates further action as for example requesting another procedure from the task manager or interacting with the application manager. The application manager controls a number of application modules which provide an interface to the external applications as for example the database and the display. The application manager analyses requests from the task manager and from the task modules, identifies an appropriate module which can handle the request the best and passes it to that module. It is also responsible to direct the answers of the requests to the right modules.

**Contextual Information** Natural language understanding in general requires contextual information in order to assign an appropriate meaning to utterances. Following the dynamic view on meaning which bases on the work from Stalnaker (1974), Kamp (1981) and Heim (1982) the meaning of utterances are their potential to update the context. Examples of dialogue systems which are strictly based on update semantics have been developed in the TRINDI Project

(TRINDI 2001) and (Larsson and Traum 2000); EMBASSI (Ludwig 2001) uses update semantics to some lesser extent.

Among the aspects of contextual information listed in (Bunt 1994) we focus on contextual information which represents the state of the task and an appropriate portion of the dialogue history including linguistic expressions, their meaning and their prominence at a given point.

While the pipeline architecture for processing user input does not provide any contextual information to the interpretation process, all parts of the dialogue manager have the capability of storing their state in local variables. It is only this way that dialogue management can benefit from contextual information. Frame-based and form-based dialogue systems also use this approach, e.g. OVIS van Noord et al. (1998), and WAXHOLM (Carlson 1996).

In BASEJUKE the task manager stores information about which *task modules* is currently active. So it holds some implicit knowledge about the state of the task and the dialogue. The task module that is responsible for keeping track of the user task stores the last database query. This information represents the information that has been established in the course of the dialogue; it represents the state of the task.

**Non-Incremental Processing of Information** In the example dialogue in table 3.1 the user specifies search criteria which update the current set of search criteria. In BASEJUKE we cannot really speak about an *update* because the criteria that have been specified in an utterance fully determine the current search criteria. That means, that BASEJUKE only responds to the last user request. It does not combine the information of a request with former requests. Therefore, BASEJUKE is not capable of dealing properly with the information of the utterances (6) and (8), which is actually supposed to be added to the former state of the task.

**Treatment of Anaphoric Expressions** The example dialogue in table 3.1 contains anaphoric expressions. In this paragraph we want to discuss how BASEJUKE treats those expressions and anaphoric expressions in general.

The main problem of BASEJUKE with respect to anaphora resolution is that it does not have any information about previous natural language utterances. Therefore BASEJUKE's concept matcher and the dialogue manager employ special interpretation rules and directives in order to cope with anaphoric expressions. The main strategy pursued by BASEJUKE is to infer the meaning of an anaphoric expression on the basis of the last database query.

Pronouns cannot be handled in general, BASEJUKE can only infer the meaning of "it" and "them", both are interpreted to denote the whole set of items from the last database query. For example, if the system displays recordings both utterances "Play it." and "Play them." cause all recordings to be played. The pronoun "him" in (10) of the example dialogue cannot be interpreted.

BASEJUKE directly translates full names into search criteria which is actually equivalent to what has been called the *non-anaphoric use of names* in section

3.1. Furthermore, BASEJUKE does not expand short forms of names to full names.

Definite noun phrases can only be resolved by BASEJUKE if they contain an ordinal determiner. In such cases it repeats the last database query; the denotation of the noun phrase is the attribute value whose position within the query result corresponds to the position specified by the ordinal determiner. The definite noun phrase “the last one” in utterance (12) from table 3.1 is interpreted correctly, because according to the discussion in section 3.1 it is interpreted as the recording which is shown on the last position on the screen in the system utterance (11). An example of an incorrect interpretation is given by the definite noun phrase in utterance (8) from table 3.1, i.e. “the second artist”. BASEJUKE would repeat the database query of which the result is presented to the user in utterance (7) and take the second item from the result, i.e. the database item that contains the recording “Coin Toss”. The final interpretation of the noun phrase is the artist “Maceo Parker” which has been determined by taking the artist attribute from that database item.

### 3.3 The New Architecture

While processing user input JUKE consults contextual information about the discourse and the current situation. The pipeline architecture of BASEJUKE will have to undergo changes in so far that it will be augmented with an information flow that is directed backwards. This section describes the framework of the implementation that is developed in this thesis. In particular, I will present the requirements for the components of JUKE which are involved in the processing of user input, discussing the functionality of the components and the difference to their equivalents in BASEJUKE .

Luperfoy et al. (1998) present an architecture where there are three components for discourse processing: the *Context Tracker* resolves contextual dependencies, *Pragmatic Adaption* translates the resolved representation of the utterance into commands which can be directly sent to external applications. *Dialogue Management* mainly focuses on coping with the communicative task; furthermore it is in control of the other components.

Allen, Ferguson and Stent (2001) also present an architecture which shows three distinct components that deal with discourse processing. The *Interpretation Manager* resolves contextual dependencies in cooperation with the *Discourse Context*. It hands the resolved representation over to the *Behavioural Agent* which controls any further processing; the *Behavioural Agent* combines the functionality of *Pragmatic Adaption* and *Dialogue Management* as presented above (Luperfoy et al. 1998).

Whereas dialogue management is central to the two former approaches, Larson and Traum (2000) present an architecture which is set around the *Total Information State*. The total information state is a data structure which stores all the data that is required for the dialogue system. In spite communicating directly to each other all components communicate through the total informa-

tion state; they read it and they manipulate it. It contains the triggers and the context for every operation. Thus, the *Information State* which contains the discourse context is the central component of discourse processing. Both the *Interpretation Module* which resolves contextual dependencies, and the *Dialogue Move Engine* which provides most of the dialogue management are dependent on the discourse context.

Summarising, the first two approaches process an utterance by passing interpretations of the utterance from the input via a contextual resolution component to the dialogue managing component, whereas in the latter approach the components process an utterance by operating on the context.

### Requirements for the Context

The example dialogue in table 3.1 contains some expressions which are not processed appropriately by BASEJUKE. In order to handle them appropriately the context component of JUKE will have to provide information about the state of the task and information about entities which qualify as antecedents for anaphoric expressions. The requirements for the context determine the structure of the content of the context component and they also define the operations which change the context.

**Dependency of Information** JUKE employs a dependency relation between the pieces of information. The relation is determined by the structure of the dialogue. The information that has been established by a dialogue can be represented in three different ways: (i) explicit representations of the discourse, e.g. DRT (Kamp and Reyle 1993), SDRT (Asher 1993); (ii) information states which are updated by utterances, e.g. (Heim 1982), DPL (Groenendijk and Stokhof 1991), (Cooper 1997). Information states are represented in dialogue systems as sets of logical formulas, e.g. EMBASSI (Ludwig et al. 2000), and as attribute-value structures, e.g. EVAR (Gallwitz et al. 1998), OVIS (van Noord et al. 1998), WAXHOLM (Carlson 1996). Larsson and Traum (2000) present a combination of logical formulas and attribute-value structures. (iii) A third alternative is to use both explicit discourse representations and information states. Grosz and Sidner (1986) for instance propose to analyse discourses by a tripartite structure containing the structure of linguistic expressions, the structure of the underlying purposes (or intentions), and the state of the attention changing in the course of the dialogue. Verbmobil (Wahlster 2000) also represents discourses by three distinct, but interacting structures: intentional structure, thematic structure and referential structure.

In order to model the dependency relation between pieces of information, information state approaches rely on backtracking mechanisms that restate former information states from the history, compare with (Ludwig et al. 2000). Discourse representation approaches can model the dependency relation by discourse structure.

We follow the approach proposed by Grosz and Sidner (1986), because it provides an explicit representation of the dependency relation between the pieces

of information that constitute the current information of the dialogue. Furthermore, it is simpler to implement than explicit discourse representation approaches.

**Anaphoric Expressions** The resolution of anaphoric expressions, in particular pronouns, names, and definite noun phrases, requires the discourse context to provide a *Saliency Structure*. The *Saliency Structure* is a structure which contains all objects that qualify as an antecedent to an anaphoric expression. The structure has to provide measures which serve as the basis to rank these objects according to their saliency. The problem is to determine saliency and accessibility (Webber 1991), i.e. whether an object is a member of the saliency structure at all. Despite the possibility of deriving saliency from features provided by the surface of the discourse, e.g. (Müller et al. 2002), (Ge et al. 1998), (McCarthy and Lehnert 1995), we have decided to compute the ranking of the saliency structure on the basis of discourse structure. One reason is that it is needed for the dependency relation of information already. We follow Webber (1991) in order to determine salient objects in a discourse analysed according to Grosz and Sidner (1986). Another important point to base anaphora resolution on discourse structure is that the heuristics of surface approaches usually rely on big data collections which were not available for the thesis at all.

**Dialogue Acts** Dialogue acts update the context. JUKE processes three dialogue acts: `inform`, `info-request` and `reject`:

The dialogue act `inform` pushes the attention of the user to information about the state of the task, i.e. the current database request, or about possible continuations of the dialogue. The dialogue act `info-request` updates the current database request with the information of the request, i.e. a set of search criteria. And the dialogue act `reject` rejects the contribution that has been made by a preceding *info-request*. It restates the database query from the state before the rejected request has been made.

While all three dialogue acts can be used by the system, the user is restricted to the use of the dialogue act `info-request` (compare with Section 2.5, Figure 2.1).

## Architecture

In order to integrate anaphora resolution and a context manager into JUKE, the architecture of BASEJUKE (see Figure 3.1) is replaced by a new architecture which resembles the one of current state-of-the-art dialogue systems (see (Luperfoy et al. 1998), (Allen, Ferguson and Stent 2001), and (Larsson and Traum 2000)) and satisfies the requirements presented above. This architecture is shown in Figure 3.3.

Since BASEJUKE and JUKE are only stages in the development of a sophisticated dialogue system most of the components will be changed in the future or they even might be replaced completely. The latter will in particular be the case for

BASEJUKE's dialogue manager, for the *Concept Selector* and for the *Concept Matcher*. Nevertheless, the anaphora resolution component should work with the basic components of BASEJUKE but also with future components. For this reason, the decision has been made to implement the anaphora resolution as a "plug and play" component which works on the parser output and provides output in the same format.

That means that anaphora resolution has to operate on the syntactic/semantic format described in section 4.4 which represents a combination of syntactic and semantic features. This format imposes some restrictions on the implementation of the anaphora resolution such that it can only replace anaphoric expressions by other expressions which are free from contextual dependencies. (See Chapter 6).

The architecture of JUKE is shown in figure 3.3. It splits up into three main independent compartments: *Natural Language Understanding*, *Context Manager* and *Dialogue Manager*.

*Natural Language Understanding* processes utterances in a pipeline where each sub-component manipulates the utterance and passes it to its successor. This design keeps the base architecture of BASEJUKE and integrates anaphora resolution according to the architectures of (Seneff et al. 1998) and (Allen, Ferguson and Stent 2001). The *Dialogue Manager* rather processes the context instead of an utterance which is in line with the design from Larsson and Traum (2000). The big arrows represent the information flow for input and output processing which are in principle independent from each other.

Each component of the architecture has to provide a certain functionality which is defined by the input and output interfaces.

**Parser** The parser is the same as in BASEJUKE. It requires textual input and puts out a graph of hypotheses.

**Anaphora Resolution** The anaphora resolution component reads in the output from the parser and puts out a graph in the same format. It requests a *Saliency Structure* (cf Section 5.4) from the *Context Manager* which enables it to replace the anaphoric expressions that occur in the input by resolved expressions, i.e. expressions that are independent from the context.

**Semantic Interpretation** This component can be identified with the combination of the *Concept Selector* and the *Concept Matcher* from BASEJUKE, because it picks out a single hypothesis from the graph and computes the final representation of the utterance. The utterance is sent to the *Context Manager* in the form of a dialogue act.

**Context Manager** The *Context manager* accepts dialogue acts from the *Semantic Interpretation* component and from the *Dialogue Manager*. Dialogue acts cause an update of the context changing the representation

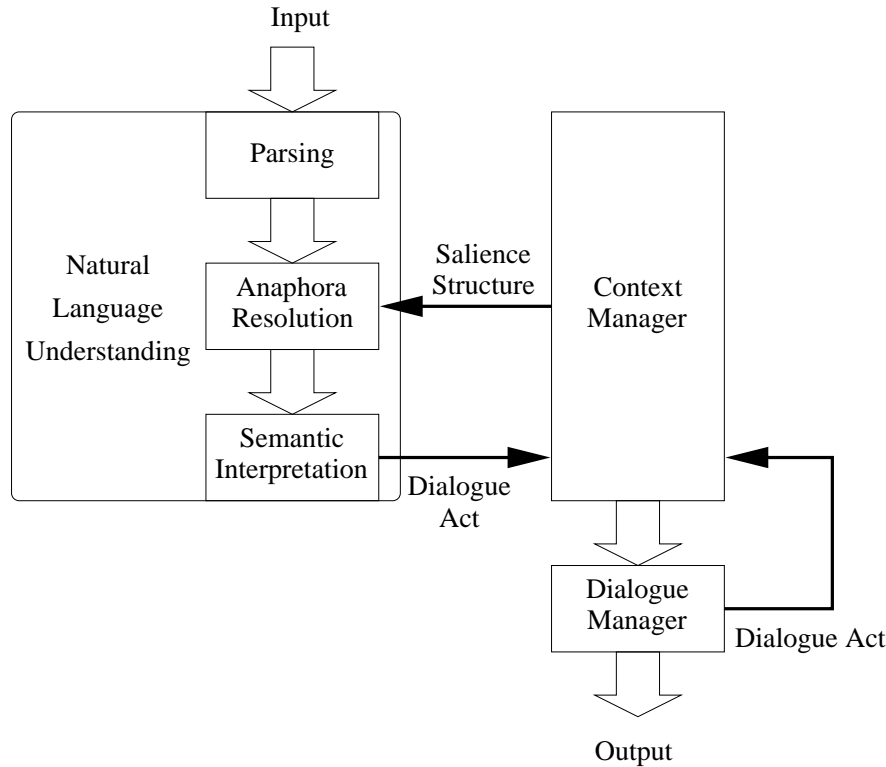


Figure 3.3: Architecture of JUKE

of the discourse structure and the table for salient objects. The *Context Manager* provides the anaphora resolution component with a *Salience Structure* containing the objects which are available for anaphoric uptake. Furthermore, it sends the state of the task and state of the dialogue to the *Dialogue Manager*.

**Dialogue Manager** The *Dialogue Manager* determines further action on the basis of the state of the task. It must report any output adequately to the *Context Manager* in the form of a dialogue act.

### 3.4 Summary

The extension of BASEJUKE comprises a change of the architecture. The components *Anaphora Resolution* and *Context Manager* have no corresponding components in the architecture of BASEJUKE.

The main target of the rest of this thesis is to describe these two components. Parts of the other components which belong to JUKE are described wherever it is necessary.

### 3.4. Summary

---

The extension of BASEJUKE to JUKE aims at the construction of a context representation at runtime which is used to provide possible antecedents for anaphora resolution and to provide task structure where the domain itself lacks structure.

## Chapter 4

# Basic Data Structures

Domain, discourse and context are modelled by feature structures. The representation format of feature structures is described in the first section; the second section contains the formal description of the domain. In the third section we present the semantic representations that are used in order to denote the objects of the domain. The last section presents a representation format which is used to represent linguistic expressions.

### 4.1 Feature Structures

Feature structures and similar formats are widely used to model linguistic structures, e.g. (Pollard and Sag 1994) and (Cooper 1997); feature structures will also be used in this thesis for representation and implementation purposes. The format supports easy implementation and transformation to other representation formats, such as XML (Bray et al. 2000) or database queries (*cf* Section 4.2). This section describes the format used in this thesis including some relations that hold between feature structures and operations on them.

The type of a feature is defined by the feature name followed by a colon followed by the type. I will use the naming convention that the names of feature structure types end on the two letters *FS* which stands for *feature structure*. For example, (13-a) defines the feature structure type *Example-FS*. (13-b) is an instance of this type.

- (13) a. *Example-FS*  
          [TITLE : String]  
      b. [TITLE “Someday My Prince Will Come”]

In principle all feature structures are based on the fundamental types shown in figure 4.1.

Type	Description
AVM	is short for attribute value matrix which is the type of any feature structure.
Stack	is the type of a stack containing feature structures. A stack provides the operations <code>push</code> , <code>peek</code> , <code>pop</code> and some typical list operations.
Symbol	is type of any symbol;
String	is the type of strings;
Integer	ranges over all negative and positive integer numbers including zero;
Boolean	has the instances <code>true</code> and <code>false</code> .

Figure 4.1: Primitive Types

The value of a feature is represented by a variable that points to a particular feature structure followed by a dot and the name of the feature followed by two brackets resembling a function call.

For example suppose that the variable *A* holds the feature structure (13-b), then the `TITLE` feature of *A* is denoted by the expression:

```
A.TITLE
```

The value of a feature is expressed by the function named identically as the feature itself. Thus the function call

```
A.TITLE()
```

returns the string “Someday My Prince Will Come”. If a feature is unspecified, the function returns the constant `null`. The features of embedded feature structures are addressed similarly. Suppose the feature structure *A* is embedded in another feature structure, say *B*, under the feature `CONTENT`, then we obtain the value of *A.TITLE* by using the expression:

```
B.CONTENT.TITLE()
```

A feature can be set by using the `set` operation. As an alternative one may also use the more readable expression where a value is assigned to a feature by means of the `:=` operator. The following two lines both do the same:

```
A.TITLE().set("My Ship")  
A.TITLE := "My Ship"
```

The feature `TITLE` of the feature structure *A* is set to the string “My Ship”.

## Relations

Two important notions about feature structures are *Equality* and *Subsumption*.

### Definition 4.1.1 *Equality of Feature Structures*

Two feature structures *A* and *B* are equal if and only if they both have the same set of features and for each feature *F* of that set holds: the value *A.F()* equals the value *B.F()*.

### Definition 4.1.2 *Subsumption of a Feature Structure*

*A* and *B* are feature structures. *A* subsumes *B* if and only if for each feature *F* of *A* holds:

*B* has a feature named *F* and

1. *A.F()* is unspecified; or
2. if *A.F()* and *B.F()* are feature structures, then *A.F()* subsumes *B.F()*; or
3. if *A.F()* and *B.F()* are Strings, then if *A.F()* is equal to some part of *B.F()*;  
or
4. *A.F()* equals *B.F()*.

For instance, the feature structure (14-a) subsumes the feature structure (14-b).

$$(14) \quad \begin{array}{l} \text{a.} \quad \left[ \begin{array}{l} \text{A } x \\ \text{B } \text{"x"} \\ \text{C} \end{array} \right] \\ \text{b.} \quad \left[ \begin{array}{l} \text{A } x \\ \text{B } \text{"xx"} \\ \text{C } xxx \end{array} \right] \end{array}$$

We also need the notion of *underspecification of feature structures* which is equivalent to the subsumption of feature structures. That means a feature structure *A* *underspecifies* a feature structure *B*, if and only if *A* subsumes *B*.

## Operations

The types `Symbol` and `String` provide conversion into each other. The type `Feature Structure` permits the operations *unification* and *intersection* with other feature structures.

The function `toString` transforms a `Symbol` into a `String`, thereby substituting each underscore by a whitespace. The reverse operation exists as well. Any string provides the function `toSymbol` which returns the string as a symbol having substituted every whitespace by an underscore. Remember the example above where the feature `TITLE` of *A* has been set to the string "My Ship". The function call

A.TITLE.toSymbol()

returns the value

My\_Ship

The operations *unification* and *intersection* are only defined between features structures of the same type.

The *unification* of the feature structures A and B is a feature structure C which is of the same type as A and B, such that both A subsume C and B subsumes C, and for every feature structure D holds: if A subsume D and B subsumes D, then C subsumes D.

If the unification operation is successful, it yields a more specific feature structure.

The *intersection* of the feature structures A and B is a feature structure C of the same type as A and B such that C subsumes A and and C subsumes B, and for every feature structure D holds: if D subsumes A and and D subsumes B, then D subsumes C.

Intersection yields a feature structure containing only those values which are common to both initial feature structures. The result always subsumes the operands.

For example given the following two feature structures

$$(15) \quad \begin{bmatrix} A & x \\ B & y \\ C & \end{bmatrix} \text{ and } \begin{bmatrix} A & x \\ B & \\ C & z \end{bmatrix}$$

Their unification is given in (16-a) their intersection in (16-b).

$$(16) \quad \begin{array}{l} \text{a. unification of (15)} \\ \quad \begin{bmatrix} A & x \\ B & y \\ C & z \end{bmatrix} \\ \text{b. intersection of (15)} \\ \quad \begin{bmatrix} A & x \\ B & \\ C & \end{bmatrix} \end{array}$$

## 4.2 Modelling the Application Domain

The database is the application of the dialogue system. *Recordings* are the only sort of objects, *genres*, *artists* and *titles* are properties of the objects. The dialogue system has to be able to query for these entities and to read them

from the database. Thus, the system's representations of the entities of the domain receive their interpretation by relating them to database queries and the corresponding results.

**Search Criteria** The database is queried by a command which requires a fixed set of search criteria as its arguments (*cf* Section 2.1) - one argument can specify a recording and three arguments can specify properties. Thus, we can represent the search criteria by a feature structure of the following type:

$$(17) \quad \textit{Search-Criteria-FS} \left[ \begin{array}{l} \text{DESC} : \left[ \begin{array}{l} \text{ARTIST} : \text{String} \\ \text{TITLE} : \text{String} \\ \text{GENRE} : \text{String} \end{array} \right] \\ \text{REF} : \text{Symbol} \end{array} \right]$$

The search criteria specifying properties are grouped together under the DESC feature (*Description*). The REF feature expresses the search criterion specifying a particular recording by holding a unique identifier.

We can distinguish between two different interpretations of these search criteria: (i) an *extensional* interpretation yields the set of all database items which match the criteria; whereas, (ii) an *intensional* interpretation yields a set of search criteria of the above format. The notion of intensional interpretation bases on the notion of extensional interpretation.

**Definition 4.2.1** *Intensional Interpretation*

Given a feature structure  $A$  of the type *Search-Criteria-FS*. Every set  $B$  of feature structures of the type *Search-Criteria-FS* qualifies an *intensional interpretation* of  $A$  iff

1. the extensional interpretation of each member of  $B$  is subset of the extensional interpretation of  $A$ ; and
2. the unification of the extensional interpretation of all members of  $B$  is identical to the extensional interpretation of  $A$ .

Intensional interpretation provides a notion of decomposition of the search criteria. For example, the following feature structure denotes all database items which contain the genre *rock*.

$$(18) \quad \left[ \begin{array}{l} \text{DESC} \left[ \begin{array}{l} \text{ARTIST} \\ \text{TITLE} \\ \text{GENRE} \text{ "rock"} \end{array} \right] \\ \text{REF} \end{array} \right]$$

Suppose, the database only contains six recordings which fall under the genre *rock*, three recordings from the band *The Beatles* and three from the band *The Rolling Stones*.

The extensional interpretation of (18) yields the set consisting of the six corresponding database items. One of many possible intensional interpretations is the following set:

$$(19) \left\{ \left[ \begin{array}{l} \text{DESC} \\ \text{REF} \end{array} \left[ \begin{array}{l} \text{ARTIST} \text{ "The Beatles"} \\ \text{TITLE} \\ \text{GENRE} \text{ "rock"} \end{array} \right] \right], \left[ \begin{array}{l} \text{DESC} \\ \text{REF} \end{array} \left[ \begin{array}{l} \text{ARTIST} \text{ "The Rolling Stones"} \\ \text{TITLE} \\ \text{GENRE} \text{ "rock"} \end{array} \right] \right] \right\}$$

**Database Query** In order to systematically retrieve intensional interpretations the database does not return the whole database item, but only the value of one attribute of the item. Thus the database query requires one additional argument which specifies the type of the attribute which should be returned. The feature structure representing a database query has the following type

$$(20) \text{ DB-Query-FS} \left[ \begin{array}{l} \text{TYPE} : \text{Symbol} \\ \text{DESC} : \left[ \begin{array}{l} \text{ARTIST} : \text{String} \\ \text{TITLE} : \text{String} \\ \text{GENRE} : \text{String} \end{array} \right] \\ \text{REF} : \text{Symbol} \end{array} \right]$$

Corresponding to the restrictions of the query command, the values of the TYPE feature are restricted to `recording`, `title`, `artist` and `genre`.

The result of such a database query is a set of those attributes which correspond to the TYPE feature and belong to an item matching the search criteria.

For example, the database query which aims at decomposing the feature structure given in (18) into the feature structures shown in (19) is represented by the following feature structure:

$$(21) \left[ \begin{array}{l} \text{TYPE} \text{ artist} \\ \text{DESC} \left[ \begin{array}{l} \text{ARTIST} \\ \text{TITLE} \\ \text{GENRE} \text{ "rock"} \end{array} \right] \\ \text{REF} \end{array} \right]$$

If we keep the assumptions of the above example, this query yields the set with the two names of the artists which are associated with the six database items:

{“The Beatles”, “The Rolling Stones” }

### 4.3 Communication about the Domain

When communicating about the domain the dialogue participants needs to denote the objects and the properties of the domain.

The objects which can be denoted by natural language expressions occur in distinct manifestations on different levels; all manifestations of one object are tightly connected with each other. We use the term *Domain Object* for the actual objects of the domain; the term *Semantic Object* denotes a semantic description possibly referring to a domain object; a *Discourse Object* is a representation containing all relevant information of the natural language expression, most notably agreement information and its denotation which is a semantic object. *Visual Objects* play the same role in the visual context as discourse objects play in the discourse context, e.g. (McGlashan 1996); they represent the symbols and expressions from the visual context containing at least their denotation which is represented by a semantic object.

**Discourse Object** On the level of natural language discourse any object is represented as a *Discourse Object*. It is associated with syntactic and semantic information. While syntactic information comprises that kind of information which is associated with the surface expression, semantic information is independent from the realization in natural language, it contains descriptive and referring information.

$$(22) \quad \textit{Discourse-Object-FS} \left[ \begin{array}{l} \text{AGR} : \left[ \begin{array}{l} \text{NUM} : \text{Symbol} \\ \text{GEN} : \text{Symbol} \end{array} \right] \\ \text{SEM} : \textit{Semantic-Object-FS} \end{array} \right]$$

A *Discourse-Object-FS* comprises agreement information from the natural language expression and its denotation which is a semantic object. The latter is hold by the SEM feature. The former is hold by the AGR feature. The AGR feature contains a feature structure holding information about the number (i.e. the NUM feature) and the gender (i.e. the GEN feature). The description of the type *Complex-NP-FS* in the next section contains some more notes about the range of the values in the AGR feature.

**Visual Object** Visual objects are simply represented as semantic objects because the properties of visual symbols are not considered to be relevant for this implementation.

**Semantic Object** On the semantic level any object denoted by a natural language expression or a visual symbol is represented as a *Semantic Object*.

A semantic object is supposed to be the description of a property or a domain object specifying its type, its properties and possibly the reference to a particular domain object.

$$(23) \quad \textit{Semantic-Object-FS} \\ \left[ \begin{array}{l} \text{TYPE} : \text{Symbol} \\ \text{DESC} : \left[ \begin{array}{l} \text{ARTIST} : \text{String} \\ \text{TITLE} : \text{String} \\ \text{GENRE} : \text{String} \end{array} \right] \\ \text{REF} : \text{Symbol} \end{array} \right]$$

This type of feature structure is equivalent to the type which represents database queries. The interpretation of feature structures of this type is therefore straight forward, see section 4.2.

**Domain Object** A *Domain Object* is represented as a unique identifier. In this case it is a symbol which uniquely identifies an object of the database. The only objects of the domain are recordings.

## 4.4 *syntactic/semantic* Representation

This section describes the basic linguistic elements of a dialogue which the user can use in order to communicate to the dialogue system. The natural language input of the user is analysed by the parser which puts out a graph containing alternative analyses of the input string. The analyses are represented in the *syntactic/semantic* representation format, i.e. feature structures containing syntactic and semantic information. These feature structures resemble the f-structures of *Lexical Functional Grammar* to some extent (Kaplan and Bresnan 1982).

**Turn, Utterance and Sentence** The user's turn is exactly one utterance consisting of a sentence (*cf* user-input). Thus, dialogue turn, utterance and sentence are all represented by the following type:

$$(24) \quad \textit{Sentence-FS} \\ \left[ \begin{array}{l} \text{MODE} : \text{Symbol} \\ \text{PRED} : \text{Symbol} \\ \text{SUBJ} : \text{Symbol} \\ \text{OBJ} : \textit{Complex-NP-FS} \end{array} \right]$$

The feature `MODE` expresses the sentence type, `PRED` holds the predicate, `SUBJ` the subject and `OBJ` the object. While the first three features have symbols as their values, the object is represented by a complex feature.

If a sentence is elliptical some features remain unspecified. For instance, a sentence only containing a nominal phrase is represented by an instance of

*Sentence-FS* with unspecified values for the features *MODE*, *PRED* and *SUBJ*.

The most important part of the sentences used with the dialogue system is the argument the verb, i.e. the object, therefore we will show the representations of all types of nominals which can serve as an argument.

**Noun Phrases** Noun phrases are represented by a feature structure of the type *Complex-NP-FS*. It is defined as follows:

$$(25) \quad \textit{Complex-NP-FS} \left[ \begin{array}{l} \text{TYPE} \quad : \text{ Symbol} \\ \text{PRED} \quad : \text{ Symbol} \\ \text{SPEC} \quad : \text{ Symbol} \\ \text{QUANT} \quad : \text{ Symbol} \\ \text{AGR} \quad : \left[ \begin{array}{l} \text{NUM} \quad : \text{ Symbol} \\ \text{GEN} \quad : \text{ Symbol} \end{array} \right] \\ \text{ADJ} \quad : \text{ list of } \textit{Simple-NP-FS} \end{array} \right]$$

Although the five features are not typed strictly the system uses particular value ranges for each feature. The first two features, *TYPE* and *PRED*, bear descriptive semantic information; the *TYPE* feature can take the values *type*, *genre*, *artist*, *title*, and *recording* each of which determines the meaning of the *PRED* feature. The most obvious divide can be drawn between the *TYPE* feature having the value *type* and the other values. In the first case the *PRED* feature takes the value of the other types, namely *genre*, *artist*, *title*, and *recording*; in all other cases the *PRED* feature will be a symbol representing a name, strictly speaking the name of a genre, an artist, or a title, or a unique identifier standing for a recording.

The two features *SPEC* and *QUANT* contain functional semantic information. The specifier feature *SPEC* encodes the information given by a determiner or a quantifier. It can take one of the following values: *unspec*, *spec*, *ANA*, *last*, or a positive integer 1, 2, and so on. The *unspec* value represents an indefinite determiner as for example “a”, and “some”. The value *spec* encodes the definite determiner; i.e. the determiner “the”, and the null determiner which is implicitly assumed for names.

Ordinal determiners modifying the definite determiner “the” are reflected by the *SPEC* feature taking a positive integer or the value *last*. The quantity feature *QUANT* takes the information of cardinal numbers modifying the central determiner. It will be set to 1 for most of the examples discussed in this essay, because the system is not capable of processing plural expressions yet, and the quantity of singular expressions is one. Pronouns are marked by the specifier *ANA*.

The last two features, *AGR* and *ADJ*, reflect syntactic information from the surface and about dependencies. The agreement feature *AGR* takes a feature structure containing the features *NUM* for number and *GEN* for gender. The number ranges

over singular, *sg*, and plural, *pl*, the gender can take one of the values *masc*, *fem* and *neut* standing for masculine, feminine and neuter, respectively. The *ADJ* feature holds the representations of all adjuncts of the noun phrase, such as modifying adjectives and prepositional phrases. The value of the feature is a list of feature structures of the type *Simple-NP-FS* which is shown below in (26). It is almost identical to the type *Complex-NP-FS* except that it does not have an *ADJ* feature.

$$(26) \quad \textit{Simple-NP-FS} \quad \left[ \begin{array}{l} \text{TYPE} \quad : \quad \text{Symbol} \\ \text{PRED} \quad : \quad \text{Symbol} \\ \text{SPEC} \quad : \quad \text{Symbol} \\ \text{QUANT} \quad : \quad \text{Symbol} \\ \text{AGR} \quad : \quad \left[ \begin{array}{l} \text{NUM} \quad : \quad \text{Symbol} \\ \text{GEN} \quad : \quad \text{Symbol} \end{array} \right] \end{array} \right]$$

*Simple objects* occurring as an adjunct are always interpreted conjunctively; that means, the semantic properties given by the adjuncts are added to the semantic properties of the head of the noun phrase.

**Pronouns** As mentioned before, pronouns are marked by the specifier value *ANA*. Since they do not contain descriptive content the *TYPE* feature and the *PRED* feature are mostly unspecified, as exemplified in (27). They have agreement features, but they never have any adjuncts.

$$(27) \quad \text{“it”} \quad \left[ \begin{array}{l} \text{TYPE} \\ \text{PRED} \\ \text{SPEC} \quad \text{ANA} \\ \text{QUANT} \quad 1 \\ \text{AGR} \quad \left[ \begin{array}{l} \text{NUM} \quad \text{sg} \\ \text{GEN} \quad \text{neut} \end{array} \right] \\ \text{ADJ} \quad () \end{array} \right]$$

The representation of the masculine and feminine pronouns deviates from this scheme. Since the format lacks some kind of sortal restriction for the difference between *human* and *non-human*, the representation of these pronouns postulates the type *artist* which is only feasible because the type *artist* and the property *human* are equivalent within the small range of the semantics used in the system.

$$(28) \quad \text{“he”/”she”}$$

$$\left[ \begin{array}{ll} \text{TYPE} & \text{artist} \\ \text{PRED} & \\ \text{SPEC} & \text{ANA} \\ \text{QUANT} & 1 \\ \text{AGR} & \left[ \begin{array}{ll} \text{NUM} & \text{sg} \\ \text{GEN} & \text{masc/fem} \end{array} \right] \\ \text{ADJ} & () \end{array} \right]$$

**Definite Noun Phrases** The head of a noun phrase is typically of the type `type` and the `PRED` feature contains the name of the type. For example:

(29) “the title”

$$\left[ \begin{array}{ll} \text{TYPE} & \text{type} \\ \text{PRED} & \text{title} \\ \text{AGR} & \left[ \begin{array}{ll} \text{NUM} & \text{sg} \\ \text{GEN} & \text{neut} \end{array} \right] \\ \text{QUANT} & 1 \\ \text{SPEC} & \text{spec} \\ \text{ADJ} & () \end{array} \right]$$

The definite determiner “the” is expressed by the feature value pair `SPEC = spec`.

**Complex Determiners** If the definite determiner is modified by an ordinal number or by a modifier like “last” and “first”, the `SPEC` feature takes the value of the modifier.

(30) “the last title”

$$\left[ \begin{array}{ll} \text{TYPE} & \text{type} \\ \text{PRED} & \text{title} \\ \text{AGR} & \left[ \begin{array}{ll} \text{NUM} & \text{sg} \\ \text{GEN} & \text{neut} \end{array} \right] \\ \text{QUANT} & 1 \\ \text{SPEC} & \text{last} \\ \text{ADJ} & () \end{array} \right]$$

**Proper Names** The analysis of proper names corresponds to the analysis of definite noun phrases, besides names never being of the type `type`. Their type is determined by the type of the object which bears the name. The name in the following example is intended to refer to the jazz musician John Coltrane, thus the `TYPE` feature is set to `artist`.

- (31) “John Coltrane”
- $$\left[ \begin{array}{ll} \text{TYPE} & \text{artist} \\ \text{PRED} & \text{john\_coltrane} \\ \text{AGR} & \left[ \begin{array}{ll} \text{NUM} & \text{sg} \\ \text{GEN} & \text{masc} \end{array} \right] \\ \text{QUANT} & 1 \\ \text{SPEC} & \text{spec} \end{array} \right]$$

**Complex Noun Phrases** All the above examples from (27) to (31), show representations of noun phrases without any adjuncts. The next example is a representation of a complex noun phrase. It is modified by the the noun “jazz” and by the prepositional phrase “from John Coltrane”.

- (32) “the second jazz title from John Coltrane”
- $$\left[ \begin{array}{ll} \text{TYPE} & \text{type} \\ \text{PRED} & \text{title} \\ \text{AGR} & \left[ \begin{array}{ll} \text{NUM} & \text{sg} \\ \text{GEN} & \text{neut} \end{array} \right] \\ \text{QUANT} & 1 \\ \text{SPEC} & 2 \\ \text{ADJ} & \left( \left[ \begin{array}{ll} \text{TYPE} & \text{genre} \\ \text{PRED} & \text{jazz} \\ \text{AGR} & \left[ \begin{array}{ll} \text{NUM} & \text{sg} \\ \text{GEN} & \text{neut} \end{array} \right] \\ \text{QUANT} & 1 \\ \text{SPEC} & \text{spec} \end{array} \right] \right. \\ & \left. \left[ \begin{array}{ll} \text{TYPE} & \text{artist} \\ \text{PRED} & \text{john\_coltrane} \\ \text{AGR} & \left[ \begin{array}{ll} \text{NUM} & \text{sg} \\ \text{GEN} & \text{masc} \end{array} \right] \\ \text{QUANT} & 1 \\ \text{SPEC} & \text{spec} \end{array} \right] \right) \end{array} \right]$$

Modifiers are stored in the adjunct list which is held by the ADJ feature. Here, analyses of the modifying noun and the prepositional phrase are stored in the adjunct list.

## Chapter 5

# Structure of Dialogue

A dialogue is a sequence of connected dialogue acts, for example spoken or written utterances, gestures and other actions that are meaningful to one of the dialogue participants (Grosz and Sidner 1986). The structure of the dialogue and the meaning of single dialogue acts and of the dialogue as a whole are important to motivate the structure of a context component that has to provide appropriate information to the anaphora resolution component of a dialogue system and to the dialogue manager.

Recent approaches analyse dialogues according to *Conversational Game Theory* as games e.g. (Hulstijn 2000), (Larsson and Traum 2000), (Ginzburg 1996), (Carletta et al. 1997). The participants of the dialogue play the dialogue game by making moves, so called *Dialogue Moves* which are equivalent with *Dialogue Acts*. The rules of the game are defined by the dialogue model in figure 2.1. The state of the game defines the state of the dialogue and the state of the task. This state is also referred to as *information state*, e.g. (Cooper 1997), (Matheson et al. 2000). After each dialogue move (i.e. dialogue act) the state of the game changes, that means that dialogue acts update the information state, see Section 7.

Although our understanding of interaction between the dialogue participants relies on the above approaches, we rather use the terminology of (Grosz and Sidner 1986) which in our opinion is more appropriate than the game metaphor when talking about discourse structure.

**Discourse Structure** Discourse structure is an important factor for determining the information that is conveyed by a dialogue. It provides information for the resolution of referring and anaphoric expressions, see for example (Grosz and Sidner 1986) exemplifying the interpretation of definite noun phrases, (Webber 1991) explaining discourse deictic expressions, and (Kamp and Reyle 1993) showing the resolution of pronouns. Furthermore, discourse structure determines how a new utterance relates to the other parts of the discourse (see for example (Grosz 1995), (Lascares and Asher 1999)). For instance, *Rhetorical Structure Theory* (Mann and Thompson 1987) and *SDRT*

(Lascarides and Asher 1993) use a variety of discourse relations to capture the relations among the parts of a discourse. (Grosz and Sidner 1986) and (Webber 1991) use two relations which in essence correspond to *subordination* and *co-ordination*.

Our understanding of discourse structure is due to Grosz and Sidner (1986) which divide a discourse into three distinct but interacting components: the structure of the segments containing the actual utterances (*linguistic structure*), the structure about the intentions conveyed by the utterances (*intentional structure*) and information about the focus of attention payed to the entities that have been introduced into the dialogue (*attentional state*).

The *linguistic structure* provides a notion of discourse structure for it divides the discourse into *discourse segments* which contain utterances and other discourse segments. The *intentional structure* contains the intentions that underly single utterances, discourse segments and the whole discourse. Furthermore, it holds the relations which hold among the intentions of a discourse. The *attentional state* represents the focus of attention. It serves as summary of the dialogue summarising the information, the objects and the topics which are of interest at some point of the dialogue. The attentional state provides the basis to derive a notion of *salience of objects*.

The following sections provide detailed explanations of these three parts which constitute the structure of a discourse.

## 5.1 Linguistic Structure

Following Grosz and Sidner (1986) the *Linguistic Structure* of a dialogue is a structure imposed over the sequence of utterances that comprise a dialogue. The utterances of a discourse are naturally aggregated into *Discourse Segments*. Such discourse segments are taken to be recursive structures consisting of utterances as the minimal elements and embedded discourse segments. The linguistic structure consists of of discourse segments and an embedding relationship between them.

The example dialogue (table 3.1) is structured as shown in table 5.1. It consists of thirteen utterances which are aggregated into seven discourse segments. The abbreviations ‘S’ and ‘U’ stand for the dialogue participants ‘System’ and ‘User’, respectively.

As Webber (1991) points out, the embedding relationship between discourse segments can be captured by a tree structure. She identifies two basic relations between discourse segments: *parent-of* and *right-sibling-of*. A discourse segment  $DS_A$  is the parent of a discourse segment  $DS_B$  if  $DS_A$  is directly embedded under  $DS_A$ . A discourse segments  $DS_C$  is the right sibling of a discourse segment  $DS_D$  if  $DS_C$  and  $DS_D$  are have the same parent segment and  $DS_C$  immediately precedes  $DS_D$ . Taking utterances as minimal discourse segments we receive the tree representation of the linguistic structure shown in figure 5.1. We also call this structure the *Dialogue Tree*.

Following Grosz and Sidner (1986) the segmentation of the discourse is deter-

Table 5.1: Linguistic Structure

$DS_1$	[		(1) S:	Select a recording and I will play it for you. Here is a list of genres.	
		$DS_2$	[	(2) U:	Is there anything from Maceo.
				(3) S:	I don't have an artist called Maceo.
		$DS_3$	[	(4) U:	I want a jazz song.
				(5) S:	Please, choose a jazz artist.
		$DS_4$	[	(6) U:	Maceo.
				(7) S:	I have these jazz recordings from Maceo Parker.
		$DS_5$	[	(8) U:	Show me the second artist.
				(9) S:	Choose a jazz recording from Miles Davis.
		$DS_6$	[	(10) U:	Is there some pop from him?
				(11) S:	These are the pop recordings from Miles Davis.
		$DS_7$	[	(12) U:	Play the last one.
				(13) S:	OK.

mined on the basis of the intentions that underly the utterances. The relations between the segments are surface reflection of the relationships among the elements of the intentional structure, i.e. the intentions. For this reason, the motivation for segmenting the example dialogue as presented in this section will be left to the following chapter, where we discuss the intentional structure of the example dialogue.

## 5.2 Intentional Structure

Following Grosz and Sidner (1986) the intentional structure controls the discourse structure. Intentions and a small number of relationships between them provide the basic elements of the intentional structure. Grosz and Sidner call the intentions that underly discourse segments *discourse segment purposes* or DSP's.

Furthermore, they propose two relations to hold between intentions: *dominance* and *satisfaction-precedence*. Dominance is a subordinating relation – if the DSP of one segment serves to satisfy that of another, the latter *dominates* the former. If the order in which the DSP's are satisfied is relevant, the satisfaction-precedence relation comes into play. Satisfaction-precedence can be identified with coordination in a sequence – if one DSP must be satisfied before another (in order to satisfy some larger purpose), then the former *satisfaction-precedes* the latter.

In order to clarify the notion of satisfaction-precedence and dominance, we see

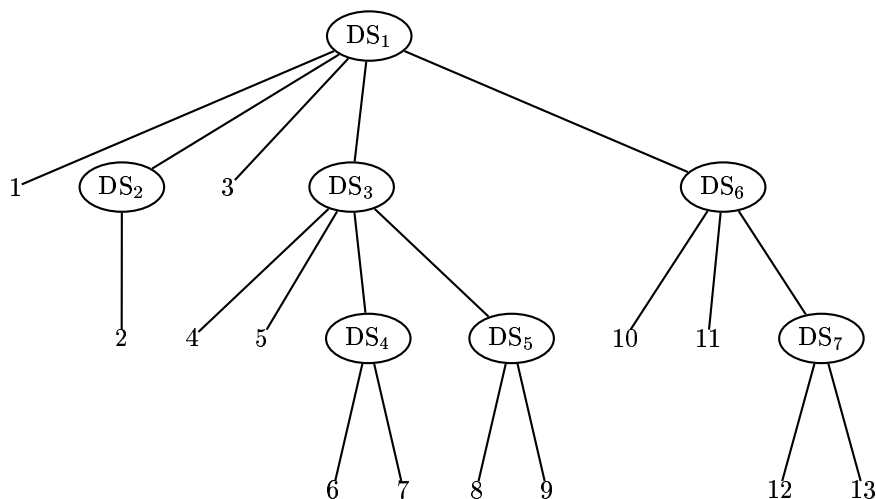


Figure 5.1: Dialogue Tree

in Table 5.2 the intentions which underlie the utterances of the example dialogue. When an utterance does not have an underlying intention, this is indicated by a dash.

The intention that underlies utterances (6) seems to contain more information than one would expect on first sight. This is due to the treatment of anaphoric expressions presented in Section 3.1 and described in detail in Chapter 6. The utterance (6) only contains the anaphoric expression “Maceo”. It is resolved to something that can be paraphrased by the expression “the artist Maceo Parker from the genre jazz”. Therefore, the single word “Maceo” expresses the complex intention “to take something from the genre jazz and from the artist Maceo Parker” as shown in (6) in Table 5.2.

In order to give examples for dominance and satisfaction-precedence from the terminology of Grosz and Sidner (1986), we take look at the intentions underlying utterances (1), (12), (4) and (6). The intention underlying utterance (1) *dominates* the intention underlying utterance (12), because the intention *to take the the recording rec\_12345 with the properties ...* satisfies the intention *to play a recording*.

Intention (4) *satisfaction-precedes* intention (6) because (4) must be satisfied before (6). This observation is a trivial one, because (6) is the conjunction of (4) and another intention. If one could not satisfy the intention “to take something from the genre jazz”, there would be no way to satisfy the conjunctive intention “to take something from the genre jazz *and* the artist Maceo Parker”.

The relations identified by Grosz and Sidner (namely dominance and satisfaction-precedence) base on a notion the satisfaction of intentions. Instead of using these two types of relations I employ one single type of relation holding between intentions. It is related to the former two relationships, but it bases on a notion of coherence and growth of information.

- 
- (1) S: the intention to play a recording;
  - (2) U: the intention to take something from the artist Maceo;
  - (3) S: the intention to reject the previous intention;
  - (4) U: the intention to take something from the genre jazz;
  - (5) S: -
  - (6) U: the intention to take something from the genre jazz  
and from the artist Maceo Parker;
  - (7) S: -
  - (8) U: the intention to take something from the genre jazz  
and from the artist Miles Davis;
  - (9) S: -
  - (10) U: the intention to take something from the genre pop  
and from the artist Miles Davis;
  - (11) S: -
  - (12) U: the intention to take the recording rec\_12345  
with the genre pop, the artist Miles Davis,  
and the title My Funny Valentine;
  - (13) S: -

Table 5.2: Intentions underlying the utterances.

**The Role of Intentions** In order to separate my use of intentions from Grosz and Sidner’s discourse segment purpose, I will use a different terminology. Single intentions that are conveyed in an utterance are *Utterance Intentions*. Utterance intentions contribute to *Dialogue Segment Intentions* and to the *Discourse Intention*.

An *Utterance Intention* is the intention that underlies an utterance, without regarding the other intentions that have been specified in the dialogue so far. The utterance intentions of the example dialogue are shown in Table 5.2.

A *Dialogue Segment Intention* is the intention which continuously valid throughout the entire dialogue segment it belongs to. It is a property of the dialogue segment itself, and part of the common ground<sup>1</sup>. All dialogue participants are aware of it. Therefore, dialogue segment intentions are *joint intentions*, that means they are jointly pursued by all dialogue participants. Dialogue segment intentions are identical to Grosz and Sidner’s DSP’s and more generally to discourse level intentions. One may also identify a dialogue segment intention with the topic of that dialogue segment.

The dialogue segment intention is determined by the utterance intention which initiates the segment; or in Webber’s terminology we can say that the intention of a dialogue segment is determined by its left-most child. Table 5.3 shows the

---

<sup>1</sup>compare with the grounding strategy described in Section 2.5

Discourse Segment	Intention
DS <sub>1</sub>	the joint intention to play a recording
DS <sub>2</sub>	the joint intention to take something from the artist Maceo
DS <sub>3</sub>	the joint intention to take something from the genre jazz
DS <sub>4</sub>	the joint intention to take something from the genre jazz and from the artist Maceo Parker
DS <sub>5</sub>	the joint intention to take something from the genre jazz and from the artist Miles Davis
DS <sub>6</sub>	the joint intention to take something from the genre pop and from the artist Miles Davis
DS <sub>7</sub>	the joint intention to take the recording rec_12345 with the genre pop, the artist Miles Davis, and the title My Funny Valentine

Table 5.3: Dialogue Segment Intentions.

dialogue segment intentions of the example dialogue.

A *Discourse Intention* is the overall intention at a given point in the dialogue which is composed out of the relevant dialogue segment intentions. Discourse intentions express the current state of task at any given point. They are updated with every utterance. Thus, we can associate each utterance of a dialogue with the corresponding discourse intention. Discourse intentions represent the intention which is jointly pursued by the dialogue participants at any given point of the dialogue. Table 5.4 shows the discourse intentions for the example dialogue. The term ‘ $I_{rec}(S)$ ’ is an abbreviation for the intention ‘to play a recording that has the properties which are specified by the set of search criteria<sup>2</sup>  $S$ ’.

Webber (1991) introduced the notion *right-frontier* in order to talk about the discourse segments which are relevant at a given point in the dialogue. We can use this notion to describe the computation of discourse intentions. The discourse intention which belongs to a given utterance, is the conjunction of the dialogue segment intentions which belong to the *right-frontier* after the utterance has been uttered. For instance, the discourse intention for utterance (7) is the conjunction of the dialogue segment intentions of DS<sub>1</sub>, DS<sub>3</sub> and DS<sub>4</sub>. The corresponding discourse tree is shown in Table 5.5. The discourse segments of the right frontier are marked by thick lines.

We have seen that intentions are associated with three different types of structural elements. (i) Intentions underly utterances, (ii) they underly dialogue segments, and (iii) they express the joint intention at any given point of the dialogue.

---

<sup>2</sup>see Section 2.1

	(1) $I_{rec}(\{\})$		
$DS_2$ [	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="border-top: 1px solid black; padding-top: 2px;">(2) <math>I_{rec}(\{\langle \text{artist, "Maceo"} \rangle\})</math></td></tr> <tr><td style="border-top: 1px solid black; padding-top: 2px;">(3) <math>I_{rec}(\{\})</math></td></tr> </table>	(2) $I_{rec}(\{\langle \text{artist, "Maceo"} \rangle\})$	(3) $I_{rec}(\{\})$
(2) $I_{rec}(\{\langle \text{artist, "Maceo"} \rangle\})$			
(3) $I_{rec}(\{\})$			
$DS_3$ [	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding-top: 2px;">(4) <math>I_{rec}(\{\langle \text{genre, jazz} \rangle\})</math></td></tr> <tr><td style="padding-top: 2px;">(5) (no change)</td></tr> </table>	(4) $I_{rec}(\{\langle \text{genre, jazz} \rangle\})$	(5) (no change)
(4) $I_{rec}(\{\langle \text{genre, jazz} \rangle\})$			
(5) (no change)			
$DS_4$ [	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="border-top: 1px solid black; padding-top: 2px;">(6) <math>I_{rec}(\{\langle \text{genre, jazz} \rangle, \langle \text{artist, "Maceo Parker"} \rangle\})</math></td></tr> <tr><td style="padding-top: 2px;">(7) (no change)</td></tr> </table>	(6) $I_{rec}(\{\langle \text{genre, jazz} \rangle, \langle \text{artist, "Maceo Parker"} \rangle\})$	(7) (no change)
(6) $I_{rec}(\{\langle \text{genre, jazz} \rangle, \langle \text{artist, "Maceo Parker"} \rangle\})$			
(7) (no change)			
$DS_5$ [	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="border-top: 1px solid black; padding-top: 2px;">(8) <math>I_{rec}(\{\langle \text{genre, jazz} \rangle, \langle \text{artist, "Miles Davis"} \rangle\})</math></td></tr> <tr><td style="padding-top: 2px;">(9) (no change)</td></tr> </table>	(8) $I_{rec}(\{\langle \text{genre, jazz} \rangle, \langle \text{artist, "Miles Davis"} \rangle\})$	(9) (no change)
(8) $I_{rec}(\{\langle \text{genre, jazz} \rangle, \langle \text{artist, "Miles Davis"} \rangle\})$			
(9) (no change)			
$DS_6$ [	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding-top: 2px;">(10) <math>I_{rec}(\{\langle \text{genre, pop} \rangle, \langle \text{artist, "Miles Davis"} \rangle\})</math></td></tr> <tr><td style="padding-top: 2px;">(11) (no change)</td></tr> </table>	(10) $I_{rec}(\{\langle \text{genre, pop} \rangle, \langle \text{artist, "Miles Davis"} \rangle\})$	(11) (no change)
(10) $I_{rec}(\{\langle \text{genre, pop} \rangle, \langle \text{artist, "Miles Davis"} \rangle\})$			
(11) (no change)			
$DS_7$ [	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="border-top: 1px solid black; padding-top: 2px;">(12) <math>I_{rec}(\{\langle \text{recording, rec\_12345} \rangle, \langle \text{genre, pop} \rangle, \langle \text{artist, "Miles Davis"} \rangle, \langle \text{title, "My Funny Valentine"} \rangle\})</math></td></tr> <tr><td style="padding-top: 2px;">(13) (no change)</td></tr> </table>	(12) $I_{rec}(\{\langle \text{recording, rec\_12345} \rangle, \langle \text{genre, pop} \rangle, \langle \text{artist, "Miles Davis"} \rangle, \langle \text{title, "My Funny Valentine"} \rangle\})$	(13) (no change)
(12) $I_{rec}(\{\langle \text{recording, rec\_12345} \rangle, \langle \text{genre, pop} \rangle, \langle \text{artist, "Miles Davis"} \rangle, \langle \text{title, "My Funny Valentine"} \rangle\})$			
(13) (no change)			

Table 5.4: Discourse Intentions

**Properties of Intentions** The intentions that underly the elements of the linguistic structure have different properties. Some specify the type of the action which is intended to be the result of the dialogue (e.g. play a recording), some specify properties of the object which the action is intended to be applied to. There are intentions which require other intentions to be specified and some which can be used alone.

In general, I want to distinguish between two action intentions and argument intentions. Furthermore, each of these intentions can be complex or simple. I start with the description of the latter distinction, because it is a technical distinction and applies to both of the former types.

The information of any intention is twofold: it has a set of *preconditions* and a *result*. The *preconditions* of an intention specify which other intentions have to be satisfied before the intention itself can be satisfied. After an intention has been satisfied, the *result* provides the information that the intention contributes to the dialogue. For the computation of intentions I refer to Chapter 7.

Intentions that have preconditions I call *complex intentions*, and intentions that do not have preconditions *simple intentions*. Simple intentions are satisfied by their own force; they always have a result. When complex intentions are introduced into the discourse they are not satisfied and they do not have a result. They only receive their result after all their preconditions have been satisfied.

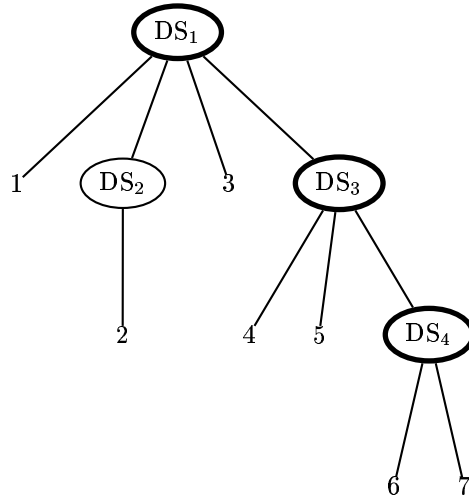


Table 5.5: Right Frontier

Now, I want to explain the distinction between argument and action intentions. These intentions are distinguished by means of the type of information they specify.

*Action intentions* specify the type of action. They are intentions to perform a particular action on arguments of a particular type. In the actual domain there is only one action intention, namely

to intend to play a recording

This action intention is also associated with  $DS_1$  in the example dialogue. It is complex because it requires another argument to be specified before it can be satisfied itself.

*Argument Intentions* are intentions concerning the arguments of the intended action. The domain allows the user to specify an argument among the following four types: *genre*, *artist*, *title* and *recording*. This is an example of an argument intention;

to intend to do something with an object that has a genre with the name “classical music”

This intention is simple. Its result provides the information that the genre should have the value *classical*. All initial utterances of the dialogue segments  $DS_2$  to  $DS_7$  introduce argument intentions into the dialogue.

**Information and Coherence** Intentions express the information which the participants provide to advance the task of the dialogue. The intentions that are used and can be understood by Juke according to the domain as it is described

in Section 2.1. The action intention *play a recording* is mapped onto the action *play*. The argument intentions are mapped onto search criteria.

Mapping intention onto the domain enables us to compute whether one intention is coherent with another. In general an intention is coherent with the another intention if the semantic representations of both intentions can be unified. There are three different cases: (i) the coherence of two action intentions, (ii) the coherence of two argument intentions, and (iii) the coherence of an action intention and an argument intention.

First, two action intentions are coherent if the feature structures which represent them can be unified. Second, two argument intentions are coherent if they satisfy the following two conditions: (a) the argument types are distinct, and (b) the unification of the semantic objects<sup>3</sup> yields a valid semantic object. Third, an action intention is always coherent with an argument intention.

According to the mapping of intentions onto the domain, we receive the following interpretations for utterance intentions, dialogue segment intentions, and discourse intentions: The information conveyed by the intention that underlies an utterance is the information that is specified by that utterance, for example an action and some search criteria. The information that is conveyed by an intention underlying a dialogue segment comprises the action and the search criteria which are valid throughout the entire segment. And finally, the information of a discourse intention is the information of the state of the task (compare with Section 3.1) stating the type of action and the search criteria which have been fixed by the dialogue at any given point.

**Segmentation and Dominance** The segmentation of the discourse and the embedding relationship between the segments is controlled by the intentional structure. The dialogue segments and their relations to the segments of the preceding segments are computed by means of coherence between utterance intentions and discourse intentions.

Any new utterance has to be assigned to a dialogue segment

1. If the utterance rejects the preceding intention by means of a dialogue act, the dialogue segment of the immediately preceding utterance is closed.
2. If the utterance intention of the utterance is unspecified the utterance is put into the currently open dialogue segment.
3. If the utterance intention is coherent with the current discourse intention, it initiates a new dialogue segment.
4. If the utterance intention of the utterance is not coherent with the current discourse intention, it initiates a new dialogue segment and closes the dialogue segment which is currently open.

---

<sup>3</sup>see Section 4.3 and Section 7.2

If an utterance initiates a new dialogue segment the dialogue segment has to be positioned into another dialogue segment or on top level. As noted above, Grosz and Sidner (1986) employ two relationships that hold among DSP's (or dialogue segment intentions), namely *satisfaction-precedence* and *dominance*. These relations control the position of the dialogue segments within the linguistic structure.

I deviate from there approach by employing only one relationship instead of two. Within the domain and the communication described in Chapter 2 it is possible to eliminate the relation *satisfaction-precedence*. To do so, we do relate dialogue segment intentions to the discourse intention at the beginning of the dialogue segment. This discourse intention is the conjunction of all dialogue segments which embed the dialogue segment. Or if we look at the dialogue tree we can say that it is the conjunction of all discourse segment intentions which are on the path from the dialogue segment in question to the root node including the root node itself.

Additionally, we have to refine the notion of dominance. In the following it is defined by means of coherence.

**Definition 5.2.1** *Dominance*

A discourse intention  $DI_1$  which is associated with the initial utterance of a discourse segment  $DS_1$  *dominates* a dialogue segment intention  $I_2$  *iff*

1.  $I_2$  is coherent with  $DI_1$ ; and
2.  $DS_1$  is open (has not been closed);
3. there is no discourse segment embedded within  $DS_1$ , such that it is open and the discourse intention associated with its initial utterance is coherent with  $I_2$ .

This definition of dominance can be used to define a notion of dominance between dialogue segment intentions.

**Definition 5.2.2** *Dominance between Dialogue Segment Intentions*

A dialogue segment intention  $I_1$  which underlies a dialogue segment  $DS_1$  *dominates* a dialogue segment intention  $I_2$  *iff* the discourse intention associated with the initial utterance of  $DS_1$  dominates  $I_2$ .

Furthermore, we need a notion of dominance of utterance intentions.

**Definition 5.2.3** *Dominance of Utterance Intentions*

The dialogue segment intention  $I_1$  which underlies a dialogue segment  $DS_1$ , *dominates* the utterance intentions of all utterances which belong to  $DS_1$ .

As said above, these three definition of dominance substitute the relations *satisfaction-precedence* and *dominance* proposed by (Grosz and Sidner 1986).

The intentional structure of a dialogue consists of utterance intentions, dialogue segment intentions and the dominance relation. Since the intentional structure controls the linguistic structure, both structures are isomorphic to each other. For instance, the intentional structure of the example dialogue is isomorphic to the dialogue tree shown in figure 5.1.

### 5.3 Attentional State

According to Grosz and Sidner (1986) the attentional state is an abstraction of the participants' focus of attention in the course of discourse. It contains information about the objects, properties, relations, and discourse intentions that are most salient at any given point<sup>4</sup>. It serves to summarise information from previous utterances crucial for processing subsequent ones, thus avoiding the need for keeping a complete history of the discourse. It is important to recognise that the attentional state is associated with the discourse itself modelling the focus of attention from an objective perspective, so that it is not a property of the dialogue participants modelling their cognitive states. The context model developed in the framework of this thesis has to provide information for at least two tasks: (i) the intention of each utterance has to be recognised and positioned at the right place in the intentional structure, and (ii) pronouns, definite descriptions, and names have to be interpreted according to previous utterances and the visual context. Both of these tasks deal with the interpretation of utterances, and thus the information required to solve them is part of the attentional state.

First, let's turn to the first task. An utterance that comes with a new intention has to be integrated somehow into the intentional structure. To do so, it is necessary to find the immediately dominating discourse intention in the dominance-precedence hierarchy, that means we have to check whether there is a discourse intention, such that the new intention satisfaction-precedes it, or is dominated by it. The rest of this section will present an approach to the integration of intentions in detail.

The second task is a part of the process of anaphora resolution. The context contains the objects which are in the current focus of attention. In order to resolve anaphoric expressions, one has to distinguish between various levels of salience. Therefore, the context provides a *Salience Structure*, a structure containing the objects which are relevant at a given time, including information about the modality by which they entered the dialogue and information about recency. The details of determining this structure are given in section 5.4; anaphora resolution is described in section 6.

The attentional state consists of three components: a focus stack, an action intention, and an argument intention.

The *Focus Stack* (see below) contains the relevant parts of the discourse. The actual dialogue segment intention is the composite of the action intention and the argument intention. So far, the argument intention is computed by unifying

---

<sup>4</sup>Examples are given in Appendix A

all intentions contained in the focus stack. The following section gives a detailed description of the focus stack and its parts and how it changes in the course of discourse.

**Focus Stack** The focus stack maps the hierarchical structure of the discourse given by the dominance-precedence hierarchy onto the information provided by the discourse. Following the processing model of Grosz and Sidner the relevant information of a discourse is stored in a stack structure, the so called *Focus Stack*. The focus stack consists of elements which store the relevant contribution of each dialogue segment; these elements are called *Focus Spaces*. A focus space contains the intention contributed by the utterance and three focus lists which I will explain later. Examples of focus stack, focus spaces and intentions are given in Appendix A.

The definition of dominance-precedence holds between dialogue segment intentions, but focus spaces only contain the utterance intention which they contributes to the dialogue segment intention. The actual dialogue segment intention is computed on the basis of the utterance intentions of all focus spaces contained in the focus stack. This enables us to compute all dialogue segment intentions which are predecessors of the current one. The term *predecessor* is meant to denote any dialogue segment intention which subordinates the current one relative to the dominance-precedence hierarchy. This helps us to interpret utterances that bring new intentions into the discourse, because any new intention is either immediately dominated by the current dialogue segment intention or by one of its predecessors.

The decision is made according to a simple heuristic; adding the new intention to a dialogue segment intention must yield a coherent intention; starting with the current dialogue segment intention and proceeding with its predecessors, it is checked whether the new intention can be added coherently. I explain the integration of a new intention in detail along the utterance (10) in Table 5.1. Before “Is there some pop from him” is uttered, the focus stack contains three focus spaces each associated with its own dialogue segment; the first focus space is associated with  $DS_1$ , it specifies the intention to play a recording; the second focus space which is linked with  $DS_3$  contains an argument intention which specifies the genre *jazz*; and the top focus space, associated with  $DS_5$ , specifies the argument intention that the artist of the argument is *Miles Davis*. At this stage the dialogue segment intention can be paraphrased as the following intention; “the dialogue participants intend to play a jazz recording from Miles Davis”.

The new intention introduced into the dialogue by the utterance (10) is the intention that the argument is associated with the genre *pop* and with the artist *Miles Davis*: let’s give this intention the name  $I_{10}$ . The next step integrates a focus space containing  $I_{10}$  into the focus stack. It cannot be pushed onto the top of the focus stack because in that case the actual argument intention would become incoherent; and it can not be pushed onto the stack either after having popped the top focus space from the stack. The reason for that is the

incoherence of the focus stack; neither the dialogue segment intention associated with  $DS_5$  nor the one associated with  $DS_3$  are coherent with  $I_{10}$ , because they both specify the genre jazz, whereas  $I_{10}$  specifies the genre pop. However, the dialogue segment intention of  $DS_1$  is coherent with  $I_{10}$ , so the first two focus spaces are popped from the focus stack and the new focus space containing  $I_{10}$  is pushed onto the top. This yields a new dialogue segment intention which can be paraphrased as “the discourse participants intend to play a pop recording from Miles Davis”.

The impact on the intentional structure can be read right from the focus stack; the dialogue segment intention associated with  $DS_1$  immediately dominates  $I_{10}$ .

**Focus Space** The focus space comprises the contribution of a particular dialogue segment to the attentional state. Besides an intention, the focus space mainly contains *Focus Lists* which provide objects for each type of attention. These objects constitute a part of the discourse context which is important to determine the interpretation of a variety of natural language expressions. Anaphoric expressions are not exclusively anaphoric on natural language expressions which have been mentioned in the discourse, but also on objects that have occurred in the visual context; one may therefore distinguish between *Discourse Attention* and *Visual Attention*.

Furthermore, some studies show that anaphoric expressions can have antecedents which don't have to occur explicitly; these antecedents have rather to be inferred from the explicitly mentioned objects and from the current topic of the discourse. This can be found in the literature under the term *Bridging*, e.g. (Asher and Lascarides 1997).

To account for all three ways of introducing objects into the attentional state, each focus space has three different types of focus lists: (i) discourse focus which is a list of discourse objects, (ii) visual focus which is a list of visual objects, and (iii) inferred focus, a list of discourse objects.

Every time one of the two other lists is changed, the inferred focus is computed anew, because different objects in the focus of visual or discourse attention support different inferences. The system infers abstractions from the current visual focus such that the inferred focus list includes the genres, artists and title names which more than one of the currently displayed objects have in common.

## 5.4 Saliency Structure

The *Saliency Structure* contains the objects which serve as antecedents for nominal anaphoric expressions. It orders these objects according to discourse structure and to the kind of attention they belong to.

The saliency of objects is determined by the saliency of the focus lists in which they occur. Focus lists receive their saliency through the structure of the focus spaces and the position of the according focus space within the focus stack.

The top element of the stack is called the highest focus space. The saliency order of focus spaces is defined as follows: the higher a focus space in the

stack, the higher is its saliency. A fixed saliency ordering is imposed over the structure of focus spaces: Each focus space contains three focus lists which define the current focus: these are (i) the discourse focus containing discourse objects which have occurred in conversation; (ii) the visual focus containing the semantic representation of those objects which are currently displayed on the screen; and (iii) the inferred focus comprising discourse objects which can be inferred from both other focus lists.

The saliency order of the focus lists within a focus space is motivated by (Kehler 2000). Anaphoric expressions prefer to take up the reference to objects which are currently selected (*in focus* in (Gundel et al. 1993)) over objects which are visible but not selected (*activated*).

Since the dialogue system uses visual objects only to inform the user about further alternative steps, visual objects are not selected, but *activated*. The objects contained in the discourse focus and in the inferred focus describe the current state; so they express selected objects, that means that they are *in focus*. This yields the following order within a focus space: The objects of the discourse focus are most salient, the inferred objects receive a lower saliency, and the lowest saliency within a focus space is assigned to the objects of the visual focus. There is no finer graduation, all objects of the same focus list have the same saliency.

## Chapter 6

# Anaphora Resolution

Anaphoric expressions are natural language expressions which determine their meaning according to the meaning of their antecedent. As stated in section 3.3 we implement anaphora resolution as a substitution of unresolved representations by resolved representations.

We restrict ourselves to nominal anaphora, so we first present the semantics of the relevant nominals. The second section gives a brief overview over anaphoric expressions. The third section describes the implementation of anaphoric expressions. The following two sections present the process of nominal anaphora resolution as it is implemented in JUKE.

### 6.1 Semantics of Nominals

This section presents the semantic representations of the nominal phrases presented in section 4.4. All kind of nominal phrases are analysed as feature structures of the type *Discourse-Object-FS*.

**Simple Noun Phrases** The translation from the syntactic/semantic format to the semantic format is straight forward for *Simple-NP-FS*. Suppose, we want to construct the semantic representation  $O$  from the noun phrase  $NP$  where  $O$  is a feature structure of the type *Discourse-Object-FS* and  $NP$  a feature structure of the type *Simple-NP-FS*. The construction of  $O$  proceeds in four steps: First, copy the agreement features. Second, if the head of the NP is a noun, set the type of the object according to the noun. Third, if the type of the NP is `recording` it is translated as a title referring to a particular recording. Otherwise, the head of the NP is a name of a particular type, and so set the corresponding property to that name after having transformed the symbol to a string. These steps can be formalised as follows:

1.  $O.AGR := NP.AGR()$ ;
2. if  $NP.TYPE() = \text{type}$ , then  $O.SEM.TYPE := NP.PRED()$ ;

3. if  $\text{NP.TYPE}() = \text{recording}$ , then  $\text{O.SEM.TYPE} := \text{title}$ , and  $\text{O.SEM.REF} := \text{NP.PRED}()$ ;
4. if neither  $\text{NP.TYPE}() = \text{type}$ , nor  $\text{NP.TYPE}() = \text{recording}$ , then  $\text{O.SEM.TYPE} := \text{NP.TYPE}$ , and
  - if  $\text{NP.PRED}() = \text{genre}$ , then  $\text{O.SEM.DESC.GENRE} := \text{NP.PRED.toString}()$ ; or
  - if  $\text{NP.PRED}() = \text{artist}$ , then  $\text{O.SEM.DESC.ARTIST} := \text{NP.PRED.toString}()$ ; or
  - if  $\text{NP.PRED}() = \text{title}$ , then  $\text{O.SEM.DESC.TITLE} := \text{NP.PRED.toString}()$ .

All translations of a simple noun phrase yield an underspecified semantic representation. For example, the name “John Coltrane” of which the syntactic/semantic representation is shown in (31) in Section 4.4, is translated to this discourse object representation:

$$(33) \left[ \begin{array}{l} \text{AGR} \\ \\ \text{SEM} \\ \\ \text{REF} \end{array} \left[ \begin{array}{l} \left[ \begin{array}{l} \text{NUM} \quad \text{sg} \\ \text{GEN} \quad \text{masc} \end{array} \right] \\ \\ \left[ \begin{array}{l} \text{TYPE} \quad \text{artist} \\ \text{DESC} \quad \left[ \begin{array}{l} \text{ARTIST} \quad \text{“john coltrane”} \\ \text{TITLE} \\ \text{GENRE} \end{array} \right] \end{array} \right] \\ \\ \text{REF} \end{array} \right] \right]$$

**Complex Noun Phrases** Translating a complex noun phrase, say  $NP$ , to a discourse object  $O$  is a bit more complicated than the translation of simple noun phrases, because the adjuncts of a complex noun phrase contribute to the semantics of the discourse object  $O$ , too. First of all, each adjunct is translated from the syntactic/semantic representation into a semantic representation. Second, for each adjunct integrate its corresponding semantic representation,  $A$ , into the main representation  $O$ ; those steps are described below. Third, at this point all adjuncts have been processed, and so the rest of the translation follows the steps of the translation of a simple noun phrase overwriting some of the features which have been set in step two.

But now let’s turn to the integration of the adjuncts into the discourse object  $O$ . Each adjunct needs to be translated to a discourse object; since every adjunct is of the type *Simple-NP-FS*, the translation follows the steps described earlier. For each of these discourse objects  $A$ , check if  $A$  is of the type `title` and if it also has a specified value for the `REF` feature. If so, copy the `REF` feature. If the type of  $A$  is either `genre`, `artist` or `title` copy the corresponding descriptive feature from  $A$  to  $O$ .

The description below repeats all steps of the integration of a discourse object  $A$  into the representation of a discourse object  $O$  in a more formal way:

1. If  $A.SEM.TYPE() = \text{title}$  and not  $A.SEM.REF() = \text{null}$ , then  
 $O.SEM.REF := A.SEM.REF()$ .
2. If  $A.SEM.TYPE() = \text{genre}$ , then  
 $O.SEM.DESC.GENRE := A.SEM.DESC.GENRE()$ ; or  
 if  $A.SEM.TYPE() = \text{artist}$ , then  
 $O.SEM.DESC.ARTIST := A.SEM.DESC.ARTIST()$ ; or  
 if  $A.SEM.TYPE() = \text{title}$ , then  
 $O.SEM.DESC.TITLE := A.SEM.DESC.TITLE()$ .

It should be noted that the translation from the syntactic/semantic format to the semantic format holds a systematical loss of information because it copies only that information of an adjunct which corresponds to its type.

Let me exemplify the translation process from a syntactic/semantic representation to a semantic representation by the example given in section 5.1 example (32), which is a representation of the noun phrase “the second jazz title from John Coltrane”. After the integration of the adjuncts we obtain a preliminary representation of a discourse object which looks as follows:

$$(34) \left[ \begin{array}{l} \text{AGR} \\ \text{SEM} \\ \text{REF} \end{array} \left[ \begin{array}{l} \left[ \begin{array}{l} \text{NUM} \\ \text{GEN} \end{array} \right] \\ \left[ \begin{array}{l} \text{TYPE} \\ \text{DESC} \\ \text{REF} \end{array} \right] \left[ \begin{array}{l} \left[ \begin{array}{l} \text{ARTIST} \quad \text{“john coltrane”} \\ \text{TITLE} \\ \text{GENRE} \quad \text{“jazz”} \end{array} \right] \end{array} \right. \end{array} \right. \right]$$

This preliminary representation of a discourse object lacks a type and agreement features. The translation of the head of the noun phrase is added to the above representation, such that we obtain the final result:

$$(35) \left[ \begin{array}{l} \text{AGR} \\ \text{SEM} \\ \text{REF} \end{array} \left[ \begin{array}{l} \left[ \begin{array}{l} \text{NUM} \quad \text{sg} \\ \text{GEN} \quad \text{neut} \end{array} \right] \\ \left[ \begin{array}{l} \text{TYPE} \quad \text{title} \\ \text{DESC} \\ \text{REF} \end{array} \right] \left[ \begin{array}{l} \left[ \begin{array}{l} \text{ARTIST} \quad \text{“john coltrane”} \\ \text{TITLE} \\ \text{GENRE} \quad \text{“jazz”} \end{array} \right] \end{array} \right. \end{array} \right. \right]$$

This structure represents a single title which is associated with an artist named John Coltrane and a genre named Jazz; there is no information about the title name, nor about a particular recording.

## 6.2 Anaphoric Expressions

Anaphoric expressions receive their meaning by picking up entities that have entered the context of the dialogue at an earlier stage. The system (partially) covers the resolution of the following kinds of anaphoric expressions: pronouns, names, and definite descriptions, all of which have noun phrases or visual objects as antecedents.

In order to provide ‘plug-and-play’ facility for the anaphora resolution component (compare with Section 3.3), the anaphora resolution process has to operate on the syntactic/semantic format (*cf* Section 4.4). The use of this format imposes the restriction on the implementation of the anaphora resolution process that it has to replace anaphoric expressions by other expressions which are free from contextual dependencies. (See Chapter 6). We therefore take a view on anaphora that they stand proxy for other expressions. This view is a generalisation of the *E-type* approach to pronouns (Evans 1980) and (Neale 1990). It is contrary to dynamic semantic approaches to anaphora where anaphora are bound by the context (Kamp and Reyle 1993), (van der Sandt 1992), (Groenendijk and Stokhof 1991), VERBMOBIL (Bos and Heine 2000) and DENK (Kievit et al. 2001).

The anaphora resolution process determines the accessibility of possible antecedents following the approach from Webber (1991). Only those objects are accessible which belong to a discourse segment from the right frontier of the discourse structure tree (compare with Section 5.1).

In our approach the focus stack correlates with the right frontier of the tree structure. Therefore, the focus stack determines the accessibility of objects. An object is accessible from an anaphoric expressions if it is contained in the focus stack at that point of time when the anaphoric expression is resolved. It is important to note the simultaneity of accessibility and containment in the focus stack, because manipulating the focus stack takes place after the anaphoric expressions have been resolved. Therefore, the focus stack can contain certain objects while the anaphoric expressions of an utterance are resolved. But these objects do not need to be contained in the focus stack anymore after that utterance has been integrated into the focus stack. For instance, utterance (10) of the example dialogue shown in the appendix in Section A.10 contains the pronoun “him”. Determining the antecedent of the pronoun yields the expression “Miles Davis” (see in the appendix Section A.10.1). The expression “Miles Davis” is included in the dialogue segment  $DS_5$  (*cf* Table 5.1). The focus space associated with  $DS_5$  is part of the focus stack when utterance (10) is processed by the anaphora resolution component. But it is removed from the focus stack after the focus space associated with utterance (10) is pushed integrated into the focus stack (see Appendix, Section A.10.2).

We split up anaphoric expressions into *Anaphoric Semantic Content* and *Specifier*. Like most approaches to anaphora resolution the *Anaphoric Semantic Content* determines the properties according to which the list of possible antecedents is computed. It contains a semantic description which has to be satisfied by the antecedent (e.g. Kamp (2001) and Beaver (1997)). The *specifier*

provides disambiguation: it determine the actual antecedent by specifying the position relative to the list of possible antecedents.

In the syntactic/semantic format anaphoric expressions are marked by the SPEC feature having one of the following values: `spec`, `ANA`, `last`, or a positive integer. While only representations of the types *Discourse-Object-FS* and *Semantic-Object-FS* express the anaphoric semantic content, the syntactic/semantic format bears important information about the specifier.

For this reason we need one further representation format, one that mediates between the syntactic/semantic format and the types *Discourse-Object-FS* and *Semantic-Object-FS* in order to provide the information which is necessary for anaphora resolution.

### 6.3 Anaphora Representation Structure

The *anaphora representation structure* (ARS) represents an anaphoric expression. It mediates between the requirements of anaphora resolution and the syntactic/semantic format. It splits up into three parts: an anaphoric part holding the anaphoric content and its type, a functional part defining the specifier and a dependency part containing all embedded anaphoric expressions on which the main anaphoric expression is dependent.

**Anaphoric Part** The anaphoric part of an anaphoric expression can be described by a feature structure of the following type:

$$(36) \quad \left[ \begin{array}{l} \text{TYPE} \quad : \text{symbol} \\ \text{CONT} \quad : \text{Discourse-Object-FS} \end{array} \right]$$

It is determined by the dependency part and by two features from the syntactic/semantic representation, namely the TYPE feature and the PRED feature which hold the semantic information of a noun phrase. The type of the anaphoric part declared by the TYPE feature controls the accessibility of focus spaces; it is either set to `general`, `discourse` or `visual`. General anaphoric parts marked by `general` can be resolved relative to the whole focus space; whereas discourse ARS marked by `discourse` are restricted to discourse focus and inferred focus. Visual ARS indicated by TYPE = `visual` only relate to the visual focus. The anaphoric content represented under the feature CONT is an underspecified discourse object which contributes to the anaphora resolution process in the following way: every possible antecedent of the anaphoric expression is a specific instance of the anaphoric content, or in other words, the anaphoric content is an underspecification of every possible antecedent.

**Functional Part** The functional part of an anaphoric expression defines the specifier which acts as a filter of the list of possible antecedents. Here, the specifier controls the selection of possible antecedents, when there are more than one most salient possible antecedents. A possible antecedent satisfies the

definiteness conditions if its position among all most salient possible antecedents lies within the range defined by the specifier.

Acceptable user input (*cf* Section 2.4) comprises definite noun phrases with complex determiners which determine ranges like “the first two” and positions as for example “the second”. According to these facilities the functional part of an anaphoric expression has the following form:

$$(37) \quad \begin{bmatrix} \text{FROM} & : & \text{integer} \\ \text{TO} & & : & \text{integer} \end{bmatrix}$$

The features FROM and TO define the starting position and the end position of the list holding the actual antecedents which are part of the list of the most salient possible antecedents. The first element of a list is referred to as position zero, the second as position one, and so on. It is also possible to specify positions from the end of the list; minus one denotes the last element, minus two the preceding element, and so on. In case one of the values falls out of the range of the list of the possible antecedents, the resolution process fails. The functional part of an anaphoric expression is fully determined by features SPEC and QUANT of the syntactic/semantic representation.

**Dependency Part** The dependency part of an anaphoric expression is a list containing anaphoric expressions which are derivable from the adjunct list of the syntactic/semantic representation. In other words, every member of the ADJ list which is identified as an anaphoric expression is translated into an ARS and added to the dependency list.

**General Definite Descriptions** Definite descriptions and names are interpreted as anaphoric expressions. The specifier SPEC = *spec* triggers an exhaustive interpretation; that means all most salient possible antecedents constitute the antecedent for the anaphoric expression. We obtain an exhaustive interpretation of the anaphoric part by setting FROM to zero and TO to minus one, so that they cover the whole range from the first element to the last element. General definite descriptions are marked by the simple determiner “the”. They are represented by feature structures of the following form:

$$(38) \quad \begin{array}{l} \text{“the”} \\ \text{a. functional part:} \\ \quad \begin{bmatrix} \text{FROM} & 0 \\ \text{TO} & -1 \end{bmatrix} \\ \text{b. anaphoric part:} \\ \quad \begin{bmatrix} \text{TYPE} & \text{general} \\ \text{CONT} & \textit{Discourse-Object-FS} \end{bmatrix} \end{array}$$

It should be clear that singular anaphoric expressions with an exhaustive interpretation can only be resolved felicitously if there is exactly one most salient

possible antecedent. The anaphoric part of general definite descriptions and names contains an ARS with `TYPE = general`, its content is obtained by translating the syntactic/semantic representation to a semantic representation.

**Pronouns** Pronouns are marked by the feature value pair `SPEC = ANA`. They are treated similar to definite descriptions except that the anaphoric part is a discourse ARS. It has the type `discourse`, blocking all objects from the visual focus as possible antecedents.

**Ordinal Definite Descriptions** In contrast to general definite descriptions, ordinal definite descriptions have a complex determiner composed from the definite determiner and an ordinal determiner. The `SPEC` feature is either a positive integer or the symbol `last` specifying the position in a list; the position is encoded in the features `FROM` and `TO`. Here, ordinal definite descriptions receive a special interpretation, the system always relates them to the visual focus, which means, that the anaphoric part is a visual ARS.

For example, a specified definite description with the complex determiner “the second” has the following representation:

- (39) “the second”
- a. functional part
 

FROM	1
TO	1
  - b. anaphoric part
 

TYPE	visual
CONT	<i>Discourse-Object-FS</i>

The functional part specifies the second element from all most salient possible antecedents. The anaphoric part restricts the resolution process to the visual focus.

## 6.4 Resolution of the Anaphoric Part

The resolution of the anaphoric part determines the most salient possible antecedents of an anaphoric expression from the choice that is provided by the salience list (*cf* section 5.4). It is carried out as a search for specific instances of the anaphoric expression’s content.

**Types of Anaphora Representation Structures** There are three types of ARS: general ARS, discourse ARS, and visual ARS. The type of the ARS determines the type of focus which can provide possible antecedents.

General ARS can be resolved over the whole range of a focus space; discourse ARS can only be resolved relative to discourse focus and inferred focus, whereas visual ARS can be resolved to objects of the visual focus only.

The intention behind this categorisation is that we restrict the range of possible antecedents from the linguistic form of an anaphoric expressions. For example, pronouns are taken to be anaphoric on items that have occurred as natural language expressions, whereas ordinal definite descriptions, e.g. *the first one* are taken to be anaphoric on items that have been presented on the display.

**Resolution** ARS resolution is carried out as a search of the salience hierarchy for specific instances of an underspecified object representation. The search space is defined by the type of the ARS and by the the salience order. The search algorithm starts with the most salient focus list. Depending on the type of the ARS the current focus list is searched for appropriate instances or it is skipped. If the current focus list is compatible with the type of the anaphoric part, it is searched for feature structures of which the representation of the ARS is an underspecification. We distinguish between focus lists of *Discourse-Object-FS* and *Semantic-Object-FS*: the first are searched for underspecifications of the ARS, while the latter are only searched for underspecifications of the SEM part. If the search failed or the focus list has been skipped, no specific instances have been found; and so the algorithm rejects the current focus list and repeats all steps with the next focus list in the salience order. The algorithm terminates, either if the search was successful, then it returns a list with all specific instances as a result, or if all focus spaces which occur in the salience order have been checked, but the search has failed, then the algorithm returns an empty list.

## 6.5 Application of the Functional Part

The functional part of an anaphoric expression holds the definiteness conditions which control which of the possible antecedents is the actual antecedent if any at all. It is applied to the result of the resolution of the anaphoric part determining the antecedent. The result is a feature structure of the type *Discourse-Object-FS*.

Independently of the actual algorithm which is implemented in the functional part, we can distinguish between three possible outcomes: (i) the resolution failed, (ii) the anaphoric expression has been resolved uniquely, and (iii) the resolution has been ambiguous. In the first and the third case, no unique antecedent has been found; the original anaphoric part should be returned as a result in order to provide as much certain information as possible to dependent anaphoric expressions, as described below under the paragraph “Nested Anaphoric Expressions”. In the second case, a unique antecedent has been found. It is either of the type *Discourse-Object-FS* or *Semantic-Object-FS*. In order to provide always a result of the same type, the antecedent is presented as a feature structure of the type *Discourse-Object-FS*. If the feature structure uniquely identified as the antecedent is of the type *Semantic-Object-FS*, it is translated to a *Discourse-Object-FS* by copying the original anaphoric part and overwriting the SEM feature of the original anaphoric part with it.

In general the functional part of an anaphoric expression specifies a sublist of the resolution result by pointing at the first and the last member of the sublist. Thus we can further specify the conditions for the anaphoric expression having a unique antecedent. If the process of ARS resolution returns an empty list, no possible antecedent has been found. If the features `FROM` and `T0` point at positions which fall out of the range of the resolution result, no antecedent could be determined. If they fall within the range, they determine a sublist holding the antecedent. The number feature of the original anaphoric part has to be coherent with the sublist. That means, the setting `AGR.NUM=sg` requires the sublist to contain exactly one member, whereas `AGR.NUM=p1` requires it to contain at least two members.

With respect to singular expressions there are only two relevant configurations of functional parts; exhaustive interpretation (see Structure (38) in Section 6.3), and ordinal specifications exemplified in (39) in the Section 6.3. An example for an exhaustive interpretation is the functional part of a general definite description like “the title”. The `FROM` feature points to the first element of the result of the ARS resolution and the `T0` feature points to the last element. In order to provide a unique antecedent for the singular expression “the title” that result has to be a list consisting of exactly one feature structure. Ordinal definite descriptions like “the second title” point at the second element of the result of the ARS resolution. Therefore, that result needs to have at least two elements.

**Nested Anaphoric Expressions** So far we have only attended to the resolution of anaphoric expression conveyed by simple noun phrases. This section describes the resolution process in case one anaphoric expression is embedded within another. The approach commonly taken in the literature is to resolve the embedded anaphoric expressions before resolving the embedding expressions, e.g. (van der Sandt 1992). Therefore, all dependencies must be resolved before ARS resolution and application of the functional part can take place. The dependency part holds a list with embedded unresolved anaphoric expressions. These expression have to be resolved and integrated into the anaphoric part before the resolution of the main anaphoric expression starts. Each resolution result is integrated into the anaphoric content in the same way as semantic representations of adjuncts are integrated into the semantic representation of a complex noun phrase. Here the the question arises of how to treat anaphoric expressions which could not be resolved. In these cases the system falls back to accommodation of the unresolved anaphoric expression. That means, the anaphoric content of the embedded anaphoric expression is treated as if it was resolved already.

**Example** of a nested anaphoric expression

(40) “the last title from Coltrane”

I want to give an example which illustrates the single steps of the resolution algorithm. The ordinal definite description “the last title from Coltrane” is split up in three parts (41), (42) and (43)

The main anaphoric part shown below in (41) is derived from the TYPE and PRED features of the main noun phrase and the adjunct.

$$(41) \left[ \begin{array}{l} \text{TYPE} \quad \text{visual} \\ \\ \\ \text{CONT} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \quad \text{sg} \\ \text{GEN} \quad \text{neut} \end{array} \right] \\ \\ \text{SEM} \left[ \begin{array}{l} \text{TYPE} \quad \text{title} \\ \text{DESC} \left[ \begin{array}{l} \text{ARTIST} \\ \text{TITLE} \quad \text{“coltrane”} \\ \text{GENRE} \end{array} \right] \end{array} \right] \\ \\ \text{REF} \end{array} \right] \end{array} \right] \end{array} \right]$$

This is the main functional part:

$$(42) \left[ \begin{array}{l} \text{FROM} \quad -1 \\ \text{TO} \quad -1 \end{array} \right]$$

The dependency part is a list with the anaphoric expression derived from the name “Coltrane”.

$$(43) \quad \begin{array}{l} \text{a. anaphoric part of “Coltrane”} \\ \left[ \begin{array}{l} \text{TYPE} \quad \text{general} \\ \\ \\ \text{CONT} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \quad \text{sg} \\ \text{GEN} \quad \text{masc} \end{array} \right] \\ \\ \text{SEM} \left[ \begin{array}{l} \text{TYPE} \quad \text{artist} \\ \text{DESC} \left[ \begin{array}{l} \text{ARTIST} \quad \text{“coltrane”} \\ \text{TITLE} \\ \text{GENRE} \end{array} \right] \end{array} \right] \\ \\ \text{REF} \end{array} \right] \end{array} \right] \\ \\ \text{b. functional part of “Coltrane”}: \\ \left[ \begin{array}{l} \text{FROM} \quad 0 \\ \text{TO} \quad -1 \end{array} \right] \\ \\ \text{c. “Coltrane” has no dependencies.} \end{array}$$

First, the dependencies have to be resolved. Suppose, that the context is such that the inferred focus list contains exactly one specific instance of the anaphoric part (43-a), so the resolution algorithm returns the following list consisting of one element:

$$(44) \left( \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \quad \text{sg} \\ \text{GEN} \quad \text{masc} \end{array} \right] \\ \\ \text{SEM} \left[ \begin{array}{l} \text{TYPE} \quad \text{artist} \\ \text{DESC} \left[ \begin{array}{l} \text{ARTIST} \quad \text{"john coltrane"} \\ \text{TITLE} \\ \text{GENRE} \end{array} \right] \\ \text{REF} \end{array} \right] \end{array} \right] \right)$$

The application of the functional part (43-b) to the result (44) yields the only element of (44), because exhaustivity and the number condition are satisfied. The embedded anaphoric expression “Coltrane” has been resolved to a discourse object denoting the artist “John Coltrane”. After having integrated this result into the main anaphoric expression the main anaphoric part has changed from (41) to (45):

$$(45) \left[ \begin{array}{l} \text{TYPE} \quad \text{visual} \\ \\ \text{CONT} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \quad \text{sg} \\ \text{GEN} \quad \text{neut} \end{array} \right] \\ \\ \text{SEM} \left[ \begin{array}{l} \text{TYPE} \quad \text{title} \\ \text{DESC} \left[ \begin{array}{l} \text{ARTIST} \quad \text{"john coltrane"} \\ \text{TITLE} \\ \text{GENRE} \end{array} \right] \\ \text{REF} \end{array} \right] \end{array} \right] \end{array} \right]$$

At this stage the main anaphoric part has been updated with the information that the artist of the antecedent has the name “john coltrane”. Suppose, that the display shows a list of jazz recordings, four of which are from Miles Davis and three from John Coltrane, namely “Giant Steps”, “Mr P.C.” and “Blue Train”. The resolution algorithm continues with the resolution of the main anaphoric part: Its content underspecifies all three recordings from John Coltrane, so we obtain a list with three possible antecedents. Applying the functional part (42) to this list yields a sublist starting from the last element up to the last element. That element is the object representation of the title “Blue Train”, it may look as shown below:

$$(46) \left[ \begin{array}{l} \text{TYPE} \quad \text{title} \\ \\ \text{DESC} \left[ \begin{array}{l} \text{ARTIST} \quad \text{"john coltrane"} \\ \text{TITLE} \quad \text{"blue train"} \\ \text{GENRE} \quad \text{"jazz"} \end{array} \right] \\ \text{REF} \quad \text{rec12345} \end{array} \right]$$

We obtain the final result by setting the SEM feature of the the anaphoric content by (46). The anaphoric expression “the last title from Coltrane” has been resolved to the following discourse object:

AGR	NUM	sg	]	
	GEN	neut		
SEM	TYPE	title	]	
	DESC	ARTIST		“john coltrane”
		TITLE		“blue train”
		GENRE		“jazz”
	REF	rec12345		

## Chapter 7

# Managing the Context

The context manager provides (i) information about the state of the task and about the state of the dialogue to the dialogue manager and (ii) a salience structure to the anaphora resolution component. The context is modelled by an information state which holds two attentional states, the current one and the most recent one. It should be noted, that the appendix A contains a transcription of the internal feature structures from the processing of the example dialogue in Table 3.1. The transcription provides an extensive example containing instances of all the structures described in this section.

### 7.1 Information State

The information state models the state of the dialogue, (compare with (Larsson and Traum 2000)). It does not keep track of the entire discourse, only the currently relevant portion of the discourse and the one from the immediately preceding step are kept. Thus the information state consists of two representations of an attentional state: the current attentional state, and the previous attentional state which provides the immediately preceding part of the dialogue history.

$$(47) \quad \begin{array}{l} \textit{Information-State-FS} \\ \left[ \begin{array}{l} \textit{ATT} : \textit{Attentional-State-FS} \\ \textit{HIST} : \textit{Attentional-State-FS} \end{array} \right] \end{array}$$

The brief history in the feature HIST is needed to undo steps which have removed information from the attentional state. For instance, let's look at utterance (x) of the example dialogue (see Table 3.1 and Appendix A, in particular page 119). Right before utterance (x) the attentional state contains the specification of the search criteria  $\langle \textit{genre}, \textit{jazz} \rangle$  and  $\langle \textit{artist}, \textit{Miles Davis} \rangle$ . After the user has uttered "Is there any pop from him?" the attentional state changed to specify the search criteria  $\langle \textit{genre}, \textit{pop} \rangle$  and  $\langle \textit{artist}, \textit{Miles Davis} \rangle$ . The information that

the genre was set to jazz at an earlier stage is not contained in the attentional state anymore. Although the system accepts this specification in the example dialogue, suppose that it would reject it. In such a case, the step of the user could be cancelled by restoring the previous attentional state which is provided by the history. The lost search criterion would have been restored, as well.

**Attentional State** The attentional state consists of three components: a focus stack, an action intention, and an argument intention, all of which are represented by feature structures.

$$(48) \quad \textit{Attentional-State-FS} \left[ \begin{array}{l} \text{FOCUS} \quad : \quad \textit{List of Focus-Space-FS} \\ \text{ACT} \quad \quad : \quad \textit{Action-Intention-FS} \\ \text{ARG} \quad \quad : \quad \textit{Argument-Intention-FS} \end{array} \right]$$

The feature FOCUS holds a representation of the *Focus Stack*. Here, it is a list of feature structures representing focus spaces (see below). The actual dialogue segment intention is the composite of the action intention and the argument intention, which are held by the features ACT and ARG, respectively. The argument intention is computed by unifying all argument intentions, which are contained in the focus stack.

**Focus Space** Every focus space contains the utterance intention and all relevant objects that have been introduced into the dialogue by a dialogue act.

$$(49) \quad \textit{Focus-Space-FS} \left[ \begin{array}{l} \text{INTENTION} \quad : \quad \textit{Intention-FS} \\ \text{DISC} \quad \quad : \quad \textit{List of Discourse-Object-FS} \\ \text{VIS} \quad \quad \quad : \quad \textit{List of Semantic-Object-FS} \\ \text{INF} \quad \quad \quad : \quad \textit{List of Discourse-Object-FS} \end{array} \right]$$

The INTENTION feature holds the utterance intention; the other features DISC, VIS and INF hold the focus lists *Discourse Focus*, *Visual Focus* and *Inferred Focus*, respectively. The *Discourse Focus* representation is a list of discourse object representations, i.e. feature structures of the type *Discourse-Object-FS*. The *Visual Focus* is a list of visual objects; they are represented as *Semantic-Object-FS*.

The *Inferred Focus* holds a list of discourse objects, i.e. *Discourse-Object-FS*, which abstract over the other two focus lists. The objects contained in the inferred focus list are the discourse objects which denote the properties that the entities of one focus list have in common. For example, if the system only displays recordings which all belong to the genre jazz, then the inferred focus list contains a discourse object which denotes the property *jazz*.

## 7.2 Intentions

Intentions control the intentional structure and thereby the structure of the attentional state. They are represented by feature structures of the following format:

$$(50) \quad \textit{Intention-FS} \quad \left[ \begin{array}{ll} \text{TYPE} & : \text{ Symbol} \\ \text{PRED} & : \text{ Symbol} \\ \text{RESULT} & : \text{ AVM} \\ \text{PRECOND} & : \text{ list of Symbols } s_1 \dots s_n \\ s_1 & : \text{ AVM} \\ \dots & \\ s_n & : \text{ AVM} \end{array} \right]$$

The TYPE feature expresses the distinction between action intentions and argument intentions; the possible values are `action` and `argument`. Corresponding to this feature the above feature structure type can be specialised to *Action-Intention-FS* and *Argument-Intention-FS*.

The PRED feature specifies the predicable information of an intention. This information includes the relation that holds between two objects or the relation between an object and a name; this relation has a corresponding relation in the database, in terms of first order predicate logic it is the same as a predicate.

Preconditions of an intention are held by the feature PRECOND. They consist of an ordered set of symbols each of which is the value of another intention's PRED feature. The preconditions specify which other intentions have to be satisfied before the intention itself can be satisfied. For example, in order to play a recording one has to specify a particular recording; in such a case, the intention `play` has the precondition `recording`. For each precondition there is a feature which holds the result of the precondition after it has been satisfied.

The RESULT feature is either unspecified in case the intention is not satisfied yet, or otherwise, when the intention is satisfied, the RESULT feature holds a feature structure. While the feature PRED holds the relation specified by an intention, this feature specifies the entities which stand in that particular relation.

The system only uses one action intention, namely `play`, and four argument intentions, namely `genre`, `artist`, `title` and `recording`. All four argument intentions are primitive intentions, an example for the intention to specify a particular recording is shown below:

$$(51) \quad \text{to intend that the argument is the jazz recording Blue Train from John Coltrane with the identifier } \textit{rec12345}$$

TYPE	argument												
PRED	recording												
RESULT	<table style="border-collapse: collapse;"> <tr> <td style="border-right: 1px solid black; padding: 5px;">TYPE</td> <td style="padding: 5px;">title</td> </tr> <tr> <td style="border-right: 1px solid black; padding: 5px;">DESC</td> <td style="padding: 5px;"> <table style="border-collapse: collapse;"> <tr> <td style="border-right: 1px solid black; padding: 5px;">ARTIST</td> <td style="padding: 5px;">"john coltrane"</td> </tr> <tr> <td style="border-right: 1px solid black; padding: 5px;">TITLE</td> <td style="padding: 5px;">"blue train"</td> </tr> <tr> <td style="border-right: 1px solid black; padding: 5px;">GENRE</td> <td style="padding: 5px;">"jazz"</td> </tr> </table> </td> </tr> <tr> <td style="border-right: 1px solid black; padding: 5px;">REF</td> <td style="padding: 5px;">rec12345</td> </tr> </table>	TYPE	title	DESC	<table style="border-collapse: collapse;"> <tr> <td style="border-right: 1px solid black; padding: 5px;">ARTIST</td> <td style="padding: 5px;">"john coltrane"</td> </tr> <tr> <td style="border-right: 1px solid black; padding: 5px;">TITLE</td> <td style="padding: 5px;">"blue train"</td> </tr> <tr> <td style="border-right: 1px solid black; padding: 5px;">GENRE</td> <td style="padding: 5px;">"jazz"</td> </tr> </table>	ARTIST	"john coltrane"	TITLE	"blue train"	GENRE	"jazz"	REF	rec12345
TYPE	title												
DESC	<table style="border-collapse: collapse;"> <tr> <td style="border-right: 1px solid black; padding: 5px;">ARTIST</td> <td style="padding: 5px;">"john coltrane"</td> </tr> <tr> <td style="border-right: 1px solid black; padding: 5px;">TITLE</td> <td style="padding: 5px;">"blue train"</td> </tr> <tr> <td style="border-right: 1px solid black; padding: 5px;">GENRE</td> <td style="padding: 5px;">"jazz"</td> </tr> </table>	ARTIST	"john coltrane"	TITLE	"blue train"	GENRE	"jazz"						
ARTIST	"john coltrane"												
TITLE	"blue train"												
GENRE	"jazz"												
REF	rec12345												
PRECOND	()												

The above structure represents the intention to do something with object that is uniquely identified by the symbol `rec123456` and meets the following description: its type is `title`, its artist is `John Coltrane`, its title name is `Blue Train` and its genre is `Jazz`.

The only action intention used by the system so far is a complex intention; it specifies the type of the action (i.e. `play`) and the required type of the argument (i.e. `recording`):

(52) `intend to play a recording`

TYPE	action
PRED	play
RESULT	
PRECOND	( <code>recording</code> )
	<code>recording</code>

This structure represents the intention to play a recording; it has not been specified yet which recording has to be played, so the `recording` feature and the `RESULT` feature are unspecified.

**Intention Satisfaction** An intention is satisfied if (i) it has no preconditions and (ii) it has a specified result. Simple intentions are satisfied trivially. Complex intentions have preconditions which need to be satisfied before the actual intention can be satisfied. If one of the preconditions is satisfied it is removed from the set of preconditions and its result is integrated into the actual intention by copying it to the feature with the same name as the intention itself. If the last precondition of an intention is satisfied the intention itself is satisfied and is assigned a result. The result is computed by a particular rule which exists for each complex intention.

For example, suppose the intention (51) is taken to satisfy the precondition in (52). First, the symbol `recording` is removed from the preconditions and the feature `recording` is set to the result of (51). We obtain a preliminary representation of an intention which does not have any preconditions, but still has no result yet. Second, the rule for the action intention "play-recording" computes the result. It simply copies the `recording` feature to the `RESULT`

feature. The satisfied intention is represented below:

- (53) to intend that the argument is the jazz recording Blue Train from John Coltrane with the identifier *rec12345*

TYPE	action								
PRED	play								
RESULT	[1]								
PRECOND	()								
		TYPE	title						
		DESC	<table style="border-collapse: collapse; margin-left: 10px;"> <tr> <td style="border-right: 1px solid black; padding-right: 10px;">ARTIST</td> <td style="padding-left: 10px;">"john coltrane"</td> </tr> <tr> <td style="border-right: 1px solid black; padding-right: 10px;">TITLE</td> <td style="padding-left: 10px;">"blue train"</td> </tr> <tr> <td style="border-right: 1px solid black; padding-right: 10px;">GENRE</td> <td style="padding-left: 10px;">"jazz"</td> </tr> </table>	ARTIST	"john coltrane"	TITLE	"blue train"	GENRE	"jazz"
ARTIST	"john coltrane"								
TITLE	"blue train"								
GENRE	"jazz"								
recording	[1]	REF	rec12345						

### 7.3 Information State Updates

Dialogue acts manipulate the intentional structure of the dialogue and they introduce new objects. Both effects cause an update of the information state. According to the interaction described in section 2.5, the context can be updated by three different types of dialogue acts: *inform*, *info-request*, and *reject*. Dialogue acts are represented by feature structures of the following type:

- (54) *Dialogue-Act-FS*
- |           |   |   |        |   |                                    |        |   |                                   |
|-----------|---|---|--------|---|------------------------------------|--------|---|-----------------------------------|
| SPEAKER   | : | Symbol  |        |   |                                    |        |   |                                   |
| INT-ACK   | : | Symbol  |        |   |                                    |        |   |                                   |
| INTENTION | : | <table style="border-collapse: collapse; margin-left: 10px;"> <tr> <td style="border-right: 1px solid black; padding-right: 10px;">ACTION</td> <td style="padding-left: 10px;">:</td> <td style="padding-left: 10px;"><i>Action-Intention-FS</i></td> </tr> <tr> <td style="border-right: 1px solid black; padding-right: 10px;">ARGS</td> <td style="padding-left: 10px;">:</td> <td style="padding-left: 10px;"><i>Argument-Intention-FS</i></td> </tr> </table>              | ACTION | : | <i>Action-Intention-FS</i>         | ARGS   | : | <i>Argument-Intention-FS</i>      |
| ACTION    | : | <i>Action-Intention-FS</i>  |        |   |                                    |        |   |                                   |
| ARGS      | : | <i>Argument-Intention-FS</i>  |        |   |                                    |        |   |                                   |
| CONTENT   | : | <table style="border-collapse: collapse; margin-left: 10px;"> <tr> <td style="border-right: 1px solid black; padding-right: 10px;">DISC</td> <td style="padding-left: 10px;">:</td> <td style="padding-left: 10px;">List of <i>Discourse-Object-FS</i></td> </tr> <tr> <td style="border-right: 1px solid black; padding-right: 10px;">VISUAL</td> <td style="padding-left: 10px;">:</td> <td style="padding-left: 10px;">List of <i>Semantic-Object-FS</i></td> </tr> </table> | DISC   | : | List of <i>Discourse-Object-FS</i> | VISUAL | : | List of <i>Semantic-Object-FS</i> |
| DISC      | : | List of <i>Discourse-Object-FS</i>  |        |   |                                    |        |   |                                   |
| VISUAL    | : | List of <i>Semantic-Object-FS</i>   |        |   |                                    |        |   |                                   |

Dialogue acts are associated with one of the dialogue participants who has initiated the dialogue act - the **SPEAKER** feature codes this kind of information, and we will use the term *speaker* to refer to the initiating dialogue participant.

Furthermore, dialogue acts either acknowledge the current discourse intention or they reject it. The feature **INT-ACK** (*viz* intention acknowledge) provides information about the speaker's attitude to the current discourse intention. If he rejects it, the value is **reject**; all other values mean acceptance.

The third feature holds the intentions that are specified by the dialogue act. This format allows the speaker to specify an action intention and an argument intention within one dialogue act.

A dialogue act can introduce objects into the discourse in two ways: it can introduce discourse objects which are then uttered in a natural language sentence

and visual objects which are displayed on the screen. The **CONTENT** feature holds two lists to express this type of information: a list of *Discourse-Object-FSs* and a list of *Semantic-Object-FSs*.

The above representation format for dialogue acts is suitable to represent the dialogue acts used in this thesis. The following three types have to be seen as the *Core Dialogue Acts*, because each of them addresses one particular feature of the dialogue act representation.

**inform** The dialogue act **inform** is used to introduce objects into the dialogue. It has appropriate values for the **CONTENT** feature.

**info-request** The dialogue act **info-request** is used to specify an intention manipulating the intentional structure. It is called “info-request” because all intentions aim at querying the database. This dialogue act is marked by appropriate values for the **INTENTION** feature.

**reject** The dialogue act **reject** rejects the current discourse intention. The feature **INT-ACK** is set to **reject**.

Dialogues which use this representation format are not restricted to these three dialogue acts; it is possible to combine the three features of the representation.

**Update** Dialogue acts update the information state; the effect of a dialogue act solely depend on the values of the features of the dialogue act representation. The update procedure starts with processing the **INT-ACK** feature, it proceeds with **INTENTION** feature, and ends with processing the **CONTENT** feature.

**INT-ACK** The update of the information state starts with checking whether the dialogue participant who produced the dialogue act accepts or rejects the current discourse intention.

If the feature **INT-ACK** has the value **reject**, the last attentional state is restored by copying the **HIST** feature of the information state to the **ATT** feature. In all other cases, the state of the task is taken to be accepted completely and the current attentional state is kept for further processing.

**INTENTION** Before the dialogue act updates the focus stack the history is updated, because the current discourse intention is going to be changed by the update. For this reason, the whole attentional state is copied to the history; that means, the feature **ATT** of the information state is copied to the feature **HIST**.

If the user specifies an intention, a new focus space is created and the intention is copied to this new focus space. In cases where the dialogue act contains both an argument intention and an action intention, the argument intention is copied to the focus space and the action intention is directly copied to the action intention of the attentional state.

The integration of the new focus space is controlled by the dominance-precedence relation which is based on the coherence of intentions. If the intention of the new focus space is coherent with all other intentions in the focus stack, the focus space is pushed onto the top of the stack. If this is not the case, as many focus spaces are popped from the focus stack until the unification of all its intentions become coherent with the intention of the new focus space; only then it is pushed onto the focus stack.

The new intention possibly satisfies a precondition of an intention that has been introduced earlier. In order to satisfy a precondition the intention has to be satisfied itself; if so, the focus stack is searched top down for an unsatisfied intention which has a precondition that can be satisfied by the new intention. The first precondition that is found is going to be satisfied by it, compare section 7.2. If satisfying the precondition results in the satisfaction of the whole intention, this intention can satisfy another precondition again, and so forth.

**CONTENT** The objects that a dialogue act introduces into the discourse overwrite the according focus lists in the top focus space: The list contained in `CONTENT.DISC` becomes the discourse focus list (*viz* the feature `DISC`) and the list contained in `CONTENT.VIS` becomes the visual focus list (*viz* the feature `VIS`) of the focus space.

In order to complete the focus space the inference procedure is called; it computes the inferred focus (*viz* the feature `INF`). With these steps we obtain the focus space that corresponds to the dialogue act.

After updating the focus space the discourse intention is adapted by computing the argument intention of the attentional state.

## 7.4 Summary of the Functions of the Context

This section contains a short summary of the content of the context and a list of dialogue system components which can benefit from it.

The context manager keeps track of the dialogue in order to provide a salience structure to the anaphora resolution component and to inform the dialogue manager about the state of the task and the state of the dialogue such that it can initiate further action. Apart from these two main tasks the context manager contains information which is useful to other general components of a dialogue system.

**Content of the Context** The context is the central repository for the contextual information needed to process utterances at each point in the dialogue. It only contains the relevant part of the discourse structure, thus obviating the need for keeping a history of the whole dialogue.

According to the analysis of discourse structure, the context manager receives knowledge about the availability of objects to serve as an antecedent. Discourse

structure also allows the context manager to decide which pieces of information that have been specified during the dialogue, contribute to the state of the task. Apart from the state of the task the context manager stores the previous state relative to the intentional structure in order to be able to revoke the last step.

**Anaphora Resolution Component** The salience structure sent to the anaphora resolution component is computed as described in section 5.4. Basically it is the focus stack without the information about intentions. The objects are ordered according to their location in the focus stack: the top focus space receives the highest salience; within a focus space, the discourse focus is ranked higher than the inferred focus and the visual focus receives the lowest salience.

The salience structure contains semantic representations of objects that have been introduced into the dialogue. The anaphora resolution component uses this information to assign meaning to contextual dependent natural language expressions, such as pronouns, abbreviations of names, and definite descriptions.

**Dialogue Manager** The input to the dialogue manager is the state of the task. For the current purpose, the state of the task is sufficiently expressed by the current discourse intention for it can be directly translated into a database query. In order to provide the dialogue manager with the state of the dialogue, the context manager can submit a restricted copy of the focus stack containing the single utterance intentions.

**Others** The context contains information which is beneficial to other components typically included in dialogue systems. In particular I want to point at natural language generation and speech recognisers.

The salience structure provided by the context can be used as the basis for generating pronouns, names and definite descriptions.

The salience structure and the state of the task can be used to compute the expectations of the speech recogniser regarding the natural language input of the user. Since most of the information the user has about the task has been provided by system output, most of it has entered the context at some point. The focus stack contains a lot of information about possible continuations of the task which tie up to the current state of task. Thus, the context manager provides information for making predictions about the next user utterance.

## Chapter 8

# Notes about the Implementation

The scope of the diploma thesis comprises the implementation of some parts of the architecture shown in figure 3.3, and repeated for the reader's convenience in figure 8.1. This chapter describes the current state of the implementation as of July 2002: it shows where the implementation follows the architecture, and where it deviates from it. Furthermore, it points out some implicit assumptions which one should be aware of if one wanted to improve the implementation.

The main part of the functionality presented in this diploma thesis is implemented in the modules *Context Manager*, and *Anaphora Resolution*. The *Semantic Interpretation* module implements the functionality which is required to process the example dialogue in table 3.1. The *Dialogue Manager* is simulated by some programs which receive and print out the output of the *Context Manager*, and provide it with input. The *Parsing* module has been taken over from the previous system, i.e. BASEJUKE.

All the implementation work has been done in the programming language *Java*. The modules are independent processes which communicate with each other by passing messages to each other. The fundamental procedures providing the communication have been taken from the *EMBASSI* project (*EMBASSI Project 2002*). All messages are first sent to a server. The server syntactically checks the messages and forwards them to their proper destination.

The syntax of the messages is defined by the syntax of the *Knowledge Query and Manipulation Language (KQML)* (Labrou and Finin 1997) and the *XML* specification (Bray et al. 2000). In particular we use the Java package *Koala Object Markup Language (KOML)* (Dyade 1999) to translate Java objects into the correct *XML* format.

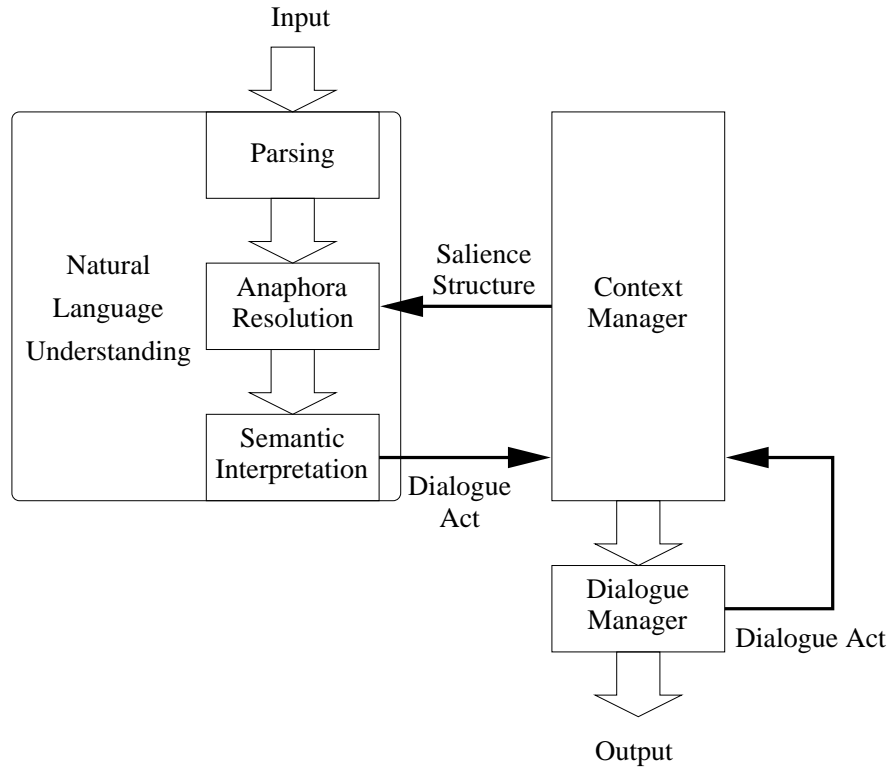


Figure 8.1: Architecture of JUKE (repeated table, see 3.3)

## 8.1 Parsing Module

The parsing module has been taken without any modifications from the predecessor BASEJUKE. It is a parser for context free grammars which can contain rules that are annotated with semantic information. The parser analyses strings and puts out a list of alternative analyses which span over the whole string. Each analysis consists of a syntactic category, i.e. the left hand side of each grammar rule, and a feature structure containing the annotations. Since the annotations represent both syntactic and semantic information, the representation format of the single analyses is called the *syntactic/semantic format*.

The grammar currently used yields analyses as described in Chapter 2, section 2.4, except that it does not provide information about the gender and the number of noun phrases in all cases. This drawback is currently amended by some add-hoc add-ons in the anaphora resolution component.

The current implementation deviates from the architecture to the extent that the parser puts out lists of alternative analyses, instead of putting out a graph

which additionally contains partial analyses of the input string. This deviation is due to the grammar which is used by the parser. It only analyses the full input string. Therefore, the output of the parser can be seen as a list of alternative analyses.

## 8.2 Anaphora Resolution

The anaphora resolution component works on the output of the parsing module which is a list of the possible analyses. It filters out those analyses which contain anaphoric elements and substitutes them by *resolved* expressions which are free from anaphoric dependencies.

The current implementation provides three main functions: (i) it identifies all anaphoric elements contained in the analyses, and translates them from the syntactic/semantic format into the intermediate format, i.e. a format which represents anaphoric structures. (ii) It initiates a resolution for each anaphoric structure by requesting the resolution from the context manager. (iii) It receives the resolved structures from the context manager, translates them into the syntactic/semantic format and replaces every anaphoric element in the analyses by its resolved counterpart.

The current implementation supports the identification of anaphoric noun phrases. In particular it recognises proper names (e.g. *Maceo Parker*), abbreviations of names (e.g. *Maceo* for *Maceo Parker*), ordinal definite noun phrases (e.g. *the first artist*), and definite noun phrases (e.g. *the title from Maceo Parker*).

The anaphora resolution component makes two assumptions which are only feasible because of the very limited setting of the domain. (i) Since the parsing module does not always provide gender and number information the anaphora resolution component carries the implicit assumption that every noun phrase is singular and masculine if not specified otherwise. Furthermore, (ii) the anaphora resolution component takes the semantic property of being human to be equivalent with being of the type `artist`. This second assumption is made to carry over semantic information from the syntactic level onto the semantic level: for the personal pronouns “he” and “she” it is assumed that they need antecedents which have the property of being human. This is necessary, because so far, the information about the gender and the number is not available for every object at every stage in the anaphora resolution process.

Chapter 9 contains hints about ways to discard these assumptions. In particular it says what to change and where the changes have to be made.

The implementation of the anaphora resolution component deviates from the target architecture in that the resolution of anaphoric structures is not carried out by the anaphora resolution component itself, but it is delegated to the context manager. This is a drawback compared to the proposal of the architecture, because it requires bidirectional communication which is more complicated

than uni-directional communication. Furthermore, it is much more time consuming that the anaphora resolution component requests the resolution of each anaphoric expression from an external component than to compute it by itself. Chapter 9 shows the steps which are needed to achieve a state of the implementation which conforms to the architecture.

## 8.3 Semantic Interpretation

The *Semantic Interpretation* component determines the final result of the language understanding process. It selects one analysis from the list of possible alternatives which are put out by the anaphora resolution component and sends it - as a dialogue act - to the context manager.

The current implementation of this component realises only a little part of the functionality it actually should provide: It mainly translates noun phrases from the syntactic/semantic format into the format of dialogue acts. Its two main limitations are that (i) it always takes the first analysis from all possible alternatives. And (ii) because of the restricted model of communication it always translates the analysis into a dialogue act of the type *specify intention*. Nevertheless, the interfaces to the anaphora resolution component and to the *context manager* conform to the requirements of the architecture.

## 8.4 Context Manager

The current implementation of the context manager integrates dialogue acts from any dialogue participant into its model of the dialogue. On the basis of this model it computes the state of the task and the dialogue which it sends to the dialogue manager. Furthermore, it resolves anaphoric structure (in the format of intermediate representations) on request.

The context manager copes with mixed initiative. That means, that it integrates dialogue acts without requiring the dialogue participants to make their contributions in a certain order. After every update with a dialogue act the context manager sends the state of the task and the dialogue to the dialogue manager. The state is represented by the current discourse intention and a partial copy of the focus stack only containing the single utterance intentions. So far, the semantic processing of the context manager is tied to the feature structure type *Semantic-Object-FS*. That means, that it can process objects of the type recording and properties of the types genre, artist, title. Section 9 gives hints about extending the semantic expressiveness.

Contrary to the architecture shown in figure 8.1, in the current implementation the context manager resolves anaphoric structures. They are resolved relative to the focus stack and the resolved structures are sent to the anaphora resolution component. Chapter 9 briefly describes how the salience structure can be derived from the focus stack and how the algorithms for anaphora resolution can be taken out of the context manager.

## 8.5 Dialogue Manager

So far, the system does not contain a dialogue management component. The integration of BASEJUKE's a dialogue manager requires many little changes, but it does not pose serious problems (see Chapter 9). Since the integration of a dialogue manager does not belong to the tasks of the diploma thesis, the input and output capabilities of the dialogue manager are simulated by dummy components which use the interfaces required by the specification of the architecture. The input to the dialogue manager is sent to a component which writes the data onto the hard-disk of the computer. The output of the dialogue manager is simulated by a program which allows the user to choose among 7 predefined dialogue acts (the dialogue acts of the example dialogue) and to send them to the context manager.

## 8.6 Hints at Evaluation

The evaluation of dialogue systems gives the justification or the reasons for the acceptance (or rejection) of certain techniques used in those dialogue systems. An evaluation of JUKE with respect to the work I have done goes well beyond the scope of the diploma thesis. Nevertheless, I want to hint at such an evaluation naming the aspects which should be evaluated and suggesting feasible methods. An evaluation should aim at two distinct aspects: It should evaluate (i) the models and algorithms presented here, and (ii) the effects which the extension from BASEJUKE to JUKE has on the overall performance of the dialogue system.

The evaluation of the discourse model and the segmentation process can be carried out on data from various origins: systematically constructed use cases; transcriptions of human-human interaction solving the same task; transcriptions of *Wizard-of-Oz* experiments; and transcriptions from projects aiming at a similar task in a similar domain (*EMBASSI (EMBASSI Project 2002)*, *SMARTKOM* (Wahlster et al. 2001)).

In order to provide the basis for the evaluation of the context manager test material comprising a stimulus and one or more expected results has to be produced in the present case. Each utterance of the data has to be annotated with the utterance intention and the discourse intention associated with the utterance. The evaluation of the anaphora resolution component requires each anaphoric expression in the data to be annotated with its denotation. Furthermore, one should annotate each utterance with the attentional state that has been reached after the utterance has been interpreted.

The model of discourse structure should be evaluated with respect to its capability of representing the state of the task and the state of the dialogue: Does the attentional state contain the information which is needed to generate an appropriate response? Does it contain enough information to interpret the immediately following utterance at any stage of the dialogue?

The process of automatic segmentation of the discourse is based on a notion of coherence of intentions with the current state of the task. It should be evaluated whether it identifies the correct segments.

Determining the correct relations between the discourse segments is crucial for obtaining the correct discourse structure. This can be evaluated by comparing the predicted discourse intention with the annotated discourse intention. The predicted discourse attention can be obtained by feeding the context manager with the annotated attentional state and a dialogue act containing the annotated utterance intention. The computed discourse attention should be equal to the annotated one.

The anaphora resolution algorithm should be evaluated according to its correctness in determining the semantic denotation of an anaphoric expression. Again one should feed the context manager with annotated attentional states. The anaphora resolution component has to be fed with the corresponding utterances. The evaluation is carried out by comparing the denotation of the resolved expression with the annotated denotation.

It should also be evaluated whether adding a context module to BASEJUKE and thus obtaining JUKE is a useful extension, at all.

The performance of both dialogue systems can best be compared if both systems accept the same natural language input.

As first step of the evaluation, one can evaluate BASEJUKE with the same data that has been used for evaluating JUKE. Here one can evaluate whether the extension of BASEJUKE to JUKE has lead to a better performance in theoretical cases.

A further step of the evaluation might evaluate both dialogue systems when they interact with real users. It is not clear whether users make use of anaphoric expressions, or whether they expect the system to remember what was said at an earlier stage of the dialogue. The evaluation of the dialogue manager should measure the *transaction success* and the *transaction efficiency* which are measures for the number of steps that subjects need to fulfil the task (Gibbon et al. 1997).

Thus, the usefulness of the extension carried out in this diploma thesis should be decided by means of the difference of the overall performance of JUKE and BASEJUKE when understanding the user's utterances and determining the current state of the task. Here, JUKE should perform significantly better than BASEJUKE, in particular when computing the meaning of pronouns, ordinal definite descriptions, definite descriptions, names and abbreviations of names, and also when determining the discourse intention.

## 8.7 Summary

The implementation carried out in the framework of this thesis provides at least the functionality to process the example dialogue (see table 3.1 and appendix A) and the other examples given in this thesis.

The context manager is the core component of the current implementation. It keeps track of the dialogue by processing dialogue acts without assuming any order of the dialogue turns. Furthermore, it provides the core functionality of anaphora resolution: it takes anaphoric structures and resolves them in relation with the focus stack which contains the entities that are in focus at a given point.

The implementation has not been completed yet. The most important open task is to integrate a dialogue manager. Furthermore, the semantic interpretation module needs a complete revision. Finally, some functionality which is currently located in the *context manager* should be moved to the anaphora resolution component.



## Chapter 9

# Future Work

This section ties up to section 8 which describes the state of the implementation in July 2002. Here, I propose some steps which aim at advancing the implementation with respect to three aspects: achieving the target architecture, extending the coverage of the system, and extending it to a spoken language dialogue system. At the end there is also a remark about integrating multi-modal user input.

**Moving the process of anaphora resolution** The implementation deviates from the target architecture (see figure 8.1) with respect to the location of the anaphora resolution process. It is situated in the context manager, but according to the target architecture it should be implemented in the anaphora resolution component. The reasons for this mismatch is that I changed the target architecture at a very late stage where the anaphora resolution process has been implemented already in the context manager. At this stage there was not much time left, so the implementation could not be finished. This change does not affect the functionality of the anaphora resolution component with respect to the whole system, but it (probably) speeds up the system.

The anaphora resolution component of the current state of the implementation identifies anaphoric expressions in the user input, transforms them to anaphoric structures and requests their resolution from the context manager. Although the context manager does not hold a concrete instance of the salience structure, it uses the focus stack to resolve anaphoric structures. This is possible, because all information included in the salience structure is also available in the focus stack. Therefore, all algorithms for resolving anaphoric structures operate on the focus stack. After having resolved the anaphoric structures, the context manager responds to the request of the anaphora resolution component by sending the corresponding resolved structures.

Separating the process of anaphora resolution from the context manager requires five steps: One has to (i) define the class representing the salience structure, (ii) provide methods which compute the salience structure from the focus stack, (iii) change the resolution algorithms, such that they operate on the salience

---

structure instead of the focus stack, (iv) change the communication between the context manager and the anaphora resolution component, and (v) add a field to the anaphora resolution component, which stores the most recent version of the salience structure.

The first four steps need further explanation:

- (i) The salience structure can be implemented as a list of triples. Each triple consists of the three different types of focus lists, i.e. visual focus, discourse focus, inferred focus.
- (ii) Extracting the salience structure from the focus stack can be simply implemented by copying the focus lists of each focus space into a new triple, thereby starting with the top focus space and proceeding to the other focus spaces in order.
- (iii) The algorithms that resolve anaphoric structures can be right applied to the salience structure. There is only the need of detaching them from the classes for focus space and focus stack and inserting them into the classes implementing the mentioned triple of focus lists and the list mentioned above, respectively.
- (iv) The communication between the context manager and the anaphora resolution component becomes much simpler than it is in the current state, because it is changed from bi-directional communication to one-way communication. The context manager should send an update of the salience structure to the anaphora resolution component every time the focus stack has changed.

Another important step to achieve the target architecture is to integrate a dialogue management component into the system.

**Integrating a Dialogue Manager** In order to integrate a dialogue manager into the current system one has to take care that the dialogue manager complies with the interface between the dialogue manager and the context manager.

**Input to the dialogue manager :** The context manager updates the dialogue manager with the state of the task and the state of the dialogue after every change of the discourse structure. Thus, the dialogue manager should always be ready to receive an update of the current discourse intention and the top focus space, or partial copies of the focus stack. Naturally speaking, the definition of what exactly is meant by the state of the task (and state of the dialogue), can be adjusted to the needs of the dialogue manager. For example, it is also possible to send the whole focus stack to the dialogue manager.

**Output of the dialogue manager :** The dialogue manager has to inform the context manager about each dialogue act addressed to the user, to ensure that the context is updated appropriately. The requirement is met

trivially, if the dialogue manager generate system responses in the format of dialogue acts and sends them both to the context manager and to a component generating the actual output.

BASEJUKE's dialogue manager expects input in the syntactic/semantic format as it is sent from the parser and the anaphora resolution component. This dialogue manager can be integrated into JUKE either by adjusting the output method of the context manager or by adjusting the input interface of the dialogue manager. Furthermore, the BASEJUKE's dialogue manager needs a new output method which sends dialogue acts to the context manager.

**Adding Communicative Capabilities** The communicative capabilities of the JUKE are restricted as mentioned in Chapters 2 and 8. In the following I will point out natural language understanding features which are desirable to have in JUKE. Among them there are (i) a representation of the start and the end phase of a dialogue, (ii) meta communication, (iii) the treatment of plural noun phrases and plural pronouns, (iv) unrestricted user input regarding the length, and (v) the treatment of ambiguous analyses occurring in the process of natural language understanding.

- (i) JUKE should be extended such that it can process a full session which begins with the user starting JUKE, choosing various titles and listening to them, and finally, ending the session.

In order to make this extension one needs to add at least one action intention expressing the user's intention to quit the system. If it is desired one may also add new dialogue acts which convey greeting in the starting phase of the dialogue.

Furthermore, the context manager needs additional rules which integrate these new dialogue acts and intentions into the dialogue context.

- (ii) Meta communication is communication about the discourse itself for the purpose of resolving ambiguities and clarifying the meaning of utterances. JUKE does not contain any meta communication, yet. At least, it is desirable that JUKE informs the user when it has not understood his utterance. For this, the dialogue manager needs the information that the utterance has not been understood.

In order to do so, I propose a simple change of the information which is passed from context manager to the dialogue manager. When the context manager informs the dialogue manager about the state of the task, it should additionally send the information about the state of the dialogue conveying whether the last utterance has been understood or not.

In particular, if parsing or semantic interpretation fails, the semantic interpretation module sends an empty dialogue act to the context manager. The context manager then pushes an empty focus space onto the focus stack. This does not affect the computation of the state of the task as far

---

as it is implemented already. Additionally the context manager can augment representation of the state of the task with a flag indicating whether the top focus space is empty.

A more sophisticated treatment of meta communication requires the introduction of new dialogue acts and new types of intentions. This is certainly out of scope for the near future.

- (iii) The current state of the implementation does not involve the treatment of plural noun phrases and plural pronouns. In order to change that, one first has to ensure, that the parser provides correct analyses. Second, one has to revise the procedures which translate between the syntactic/semantic format and anaphora representation structures. Third, it should be checked whether the anaphora resolution process produces the intended results. One should also consider the treatment of ambiguous antecedents - see below in item 5.
- (iv) A desirable property of the system is that the length of the user input is not restricted to one sentence. The user should rather be able to utter as many sentences in row as he likes to. In principle this can be done by the following steps:
  - (a) Adjust the grammar, such that it provides partial analyses of the input string. Thus, it also parses input containing more than one sentence.
  - (b) Adjust the anaphora resolution component and the semantic interpretation component such that they can correctly read partial analyses.
  - (c) In order to provide the resolution of inter-sentential anaphora which can occur within one multi sentence contribution of the user, one has to ensure that input is processed step wise. That means, a sentence has to be integrated into the dialogue context before the anaphora resolution of the successor sentence starts.

The context manager only requires minor changes. It already provides the functionality of processing a sequence of dialogue acts. One only needs to ensure that it does not update the dialogue manager before the complete utterance has been processed.

- (v) In order to obtain better results in natural language understanding one should add the treatment of ambiguities to all components. Ambiguous analyses can occur in the grammatical analysis of the parser, in the process of anaphora resolution and when producing a dialogue act in the semantic interpretation module. The context manager has to consider different ways of integrating one dialogue act into the discourse context. This results in non-unique states of the task which finally the dialogue manager has to cope with.

For a first approximation it would be sufficient to enable ambiguities for the interpretation of noun phrases and pronouns. They occur in the anaphora resolution component, and should be passed via the semantic interpretation module and the context manager to the dialogue manager. In such a case the dialogue manager should be able either to apply some reasoning which picks out one alternative or to start meta communication by asking the user for clarification.

In general, the context manager, the anaphora resolution component, the semantic interpretation component, and the dialogue manager have to be able to read graph structures which can contain alternative analyses for each possible substring of the input. A graph structure resembles a chart of a parser. In principle the parser of JUKE provides such a graph structure. The components have to identify the relevant nodes of the graph and apply their processing to them.

Concerning the output, the parser, the anaphora resolution component, the semantic interpretation component and the context manager also have to put out such graph structures after they have filled them with the result of their processing. The graph structure permits to express ambiguous results: nodes can cover distinct parts of the input, and they can contain more than one analysis.

**Extending the Domain** JUKE's domain is relatively small domain. Extending the domain includes (i) adding entries to the database, (ii) adding features to each data base item, and (iii) providing other operations which can be applied to data base items.

- (i) Extending the number of database entries does not pose any problems to the context manager and the anaphora resolution component. The lexicon and the dialogue manager will be affected most.
- (ii) Introducing new attributes which are associated with an object has some impact on the context manager and the anaphora resolution component. Currently, each recording is associated with three kinds of attributes, namely *genre*, *artist* and *title*. If one extends the number of attributes, as for example with *sub-genre*, *composer*, *producer*, *album*, *year of the recording*, and others ..., then one has to adjust (a) the syntactic/semantic representation which is put out by the parser, (b) the semantic representation format and the format of anaphora representation structures which are used by the anaphora resolution and the context manager, (c) the procedures which translate between these three representation formats.
  - (a) Extend the value range of the features TYPE and PRED which belong to feature structure types *Complex-NP-FS* and *Simple-NP-FS*.
  - (b) For each new attribute one has to add the according feature to the type *Semantic-Object-FS* which represents the semantic properties of an object. (see Section 4.3).

- 
- (c) Procedures which translate between both representation formats are implemented in the components ‘anaphora resolution’ and ‘semantic interpretation’. Furthermore, the classes implementing the types *Discourse-Object-FS* and *Semantic-Object-FS* provide methods which perform type checks and translation into other formats. All these classes have to be extended with the new attributes.

It is important to hint at a problem which arises when one adds attributes that denote human names. Within the current implementation the use of the attribute *artist* has a special use in anaphora representation structures: it indicates the requirement that the antecedent has to be human.

This treatment becomes incorrect if there are other attributes that denote human beings. It can be corrected by adding the feature HUMAN to *Semantic-Object-FS*. It takes the values ‘+’ or ‘-’ indicating whether the object is human or not human, respectively.

- (iii) Extending the functionality by adding operations to the system mainly requires adding new types of intentions such that the user can specify the operation. For example, one can add operations which modify the database: creating genres, assigning recordings a genre, deleting database items, creating play lists, deleting items from play list.

The extension is not trivial, and it requires major changes in all components of the dialogue system. The complexity of the task of the context manager would be much higher, than it is for the current implementation, because plenty of new *action-intentions* would have to be added to the discourse processing components. So far, there is only one *action-intention* (i.e. the intention to play a recording), but for every new *action-intention* it has to be precisely defined how it relates to any kind of discourse intention and thus, how it changes the focus stack.

**Extension to a Spoken Language Dialogue System** JUKE uses written natural language as input and output. The complexity of extending the context manager and the anaphora resolution component to process spoken natural language, mainly depends on the output of the speech recogniser. (a) If it only puts out the best hypothesis, this hypothesis can be right fed into the parser. No modifications have to be made. (b) If it puts out the  $n$  best hypotheses, it is necessary to include the processing of alternative hypotheses and the treatment of ambiguity. This has been briefly discussed in the previous paragraph. Furthermore, one has to integrate speech synthesis. This does not have a direct impact on implementations of the anaphora resolution component and the context manager.

**Integrating Multi-Modality** JUKE uses two output modalities. It puts out natural language and it displays parts of the content of the database. Adding multi-modal input from the user can be performed in two different ways as far as it concerns the context manager and the anaphora resolution component.

On the one hand, the input from various modalities can be merged before they enter the semantic interpretation component. In such a case, the context manager and the anaphora resolution component will not be affected by the change. On the other hand, the input of every modality can be sent to the context manager separately. In such a case both components would have to undergo changes. The context manager would have to understand new dialogue acts and it would need to know how to update the information state with them. The anaphora resolution component would need to be extended such that it also can resolve deictic expressions according to the context.

---

# Chapter 10

## Conclusions

This diploma thesis exemplifies the benefits of using discourse context in a dialogue system. The context is determined by the structure of a dialogue according to the theory of discourse structure presented by Grosz and Sidner (1986).

The dialogue system described in this thesis benefits from the context in two ways: (i) The context provides crucial information for computing the meaning of pronouns, definite noun phrases and proper names. And (ii), the context provides a structured view on the information specified by the user, even if the task completely lacks domain structure.

The computation of discourse structure is based on simple semantics, namely sets of attribute-value pairs as a formalisation of database queries. The computational effort of handling this kind of semantics is reasonably low.

**Discourse Structure** Following Grosz and Sidner (1986) a dialogue is represented by three distinct but interacting components: (i) The *linguistic structure* provides a notion of discourse structure for it divides the discourse into *discourse segments* which contain utterances and other discourse segments. (ii) The *intentional structure* contains the intentions that underly single utterances, discourse segments and the whole discourse. Furthermore, it holds the relations which hold among the intentions of a discourse. (iii) The *attentional state* represents the focus of attention. It serves as summary of the dialogue summarising the intentions and the objects which are of interest at a given point of the dialogue. The attentional state consists of a stack structure (*focus stack*) which contains the contributions (*focus spaces*) of dialogue segments. Each contribution contains two kinds of entities: the intentions that underly the corresponding dialogue segment and the objects that have been introduced into the context by that dialogue segment. The intentions are distributed of the focus stack in focus spaces. They provide the basis for computing the state of the task. The objects are also included in the focus spaces of the focus stack. They define the set of possible antecedents for anaphora resolution.

The objects contained in a focus space are divided into *discourse objects*, *visual*

---

*objects* and *inferred objects*. Discourse objects are the objects which have been introduced by the dialogue participants by means of natural language. Visual objects have been introduced into the dialogue by displaying them on the screen. Inferred objects are objects which abstract over the former two kinds of objects. Each focus space has exactly one intention. Any dialogue act that has an intention causes the start of a new dialogue segment, the contribution of which finally ends up in a new focus space pushed onto the top of the focus stack. But before the new focus space is pushed onto the top of the focus stack, the coherence with all intentions of the stack has to be guaranteed. If the intention of the new focus space is coherent with the rest of the focus stack, the new focus space is pushed onto the focus stack. Otherwise, if the intention is not coherent with the intentions of the focus stack, the focus stack is popped as many times as it is necessary to yield coherence. Only then, the new focus space is pushed onto the focus stack.

Pushing and popping the focus stack is controlled by the coherence of intentions. An intention is coherent with another intention if they do not contradict each other. Since we track most of the semantics down to database queries, we obtain a simple notion of contradiction: two database queries contradict each other if they have different values for the same attribute. Unspecified attributes are not taken into consideration. Therefore, an intention is coherent with the intentions of the focus stack if the intention does not specify an attribute which is specified in the focus stack with a different value.

**The Context Manager** The *context manager* maintains an information state which holds the state of the task and the state of the dialogue. The implementation of the information state contains an instance of the attentional state. Similar to the architectures from Larsson and Traum (2000), Allen, Ferguson and Stent (2001) and Luperfoy et al. (1998) the context manager belongs to the central dialogue processing components tightly interacting with the components for natural language understanding (NLU) and dialogue management (DM). NLU receives a salience structure from the context manager for the purpose of resolving anaphoric expressions. The salience structure contains all salient entities ordered according to their salience. The result of NLU is a dialogue act which is sent to the context manager.

DM receives the state of the task and the dialogue from the context manager. DM computes a reaction and informs the context manager about it via a dialogue act.

The context manager stores the information of a dialogue which it has identified as being relevant for interpreting new utterances. This information is useful for understanding the input and for planning further actions.

**Anaphora Resolution** Understanding anaphoric expressions that occur in natural language utterances requires the interpreter of the utterance to determine the unique antecedent of the expression among all entities that have entered the context of the dialogue at an earlier stage. If there are more than one

object that qualify as antecedent on grounds of syntactic and semantic considerations, the decision is made according to the salience of the objects: the object that is most salient is identified as the antecedent of the anaphoric expressions. The context manager provides the anaphora resolution component with a salience structure which is a copy of the focus stack yet without copying the intentions. The focus stack determines salience as follows: the more recent (higher) the focus space, the more salient are the objects of the focus space. Within a focus space discourse objects are more salient than inferred objects which are more salient than visual objects.

The implementation of anaphora resolution and salience structure supports multi-modal anaphora resolution, because it operates on visual objects and discourse objects. In particular the following kinds of anaphora can be handled:

- Short forms of proper names are expanded to full proper names. If a name is only realised as a substring, the anaphora resolution component tries to find an appropriate antecedent indicating the full name. If it succeeds, the substring is expanded to the full string.
- Definite noun phrases can receive their denotation from objects in the context. Possible antecedents have to match the semantic content of the noun phrase, the most salient object is taken to be the actual antecedent.
- A special treatment is applied to definite noun phrases containing an ordinal specifier. They are resolved to objects which are displayed in lists on the screen. The semantic content is matched against the properties of the items. If more than one item match the semantic content, the specifier determines which items are taken to serve as antecedents of the definite noun phrase.
- Names can receive their denotation from objects in the context that bear the same name. Apart from the reading that full names denote the value of one database attribute, there is another reading of (full) names, such that they denote objects and sets of properties. The system can introduce names into the dialogue which denote a set of attributes. For example, it displays the artist name “Miles Davis”, but it actually means “Miles Davis restricted to jazz”. If the user uses the same name (“Miles Davis”), it is interpreted anaphorically, such that it takes up the denotation of the former use (“Miles Davis restricted to jazz”).

**Task and Dialogue Manager** The dialogue system provides flexible access to a database of music recordings. The state of the task is represented as a database query which the user revises from step to step in order to navigate through the database.

The state of the task can be derived from discourse structure by unifying the intentions of the focus stack. Therefore, discourse structure compensates for the lack of domain structure. Since the domain of the database does not have

---

a naturally given hierarchy that specifies how the pieces of information dependent on each other, such a hierarchy is built dynamically in the course of the dialogue. The hierarchy is defined by the focus stack which reflects the part of the discourse structure which is relevant at any given point.

The dialogue manager plans the actions of the system on the basis of the state of the task and the state of the entire dialogue. As said earlier, the state of the task is represented by a database query. Utterances of the user convey intentions to manipulate the last database query. The dialogue manager is free to accept these intentions, or to reject them in cases where it cannot satisfy them.

The following tasks of a dialogue system which are not dealt with in this thesis can also benefit from the information hold by the context manager:

**input prediction** : The salience structure contains the objects which the dialogue system presents to the user as possible continuations that restrict the database search. Thus, if the dialogue system uses speech recognisers, they can use the information about possible continuations in order to form expectations about the user input.

**output generation** : If the system puts out natural language containing anaphoric expressions, the salience structure provides the information which objects are available for anaphoric uptake.

This diploma thesis has shown the feasibility and the benefits of furnishing a dialogue system with a discourse context component that exploits discourse structure. In particular we have seen that we only need relatively simple semantic processing to build discourse structure, and gain important information for natural language understanding and dialogue management.

# Appendix A

## Example Dialogue

This appendix shows a transcription of the internal structures of the context manager for the processing of the example dialogue given in 3.1. The transcription lists the input (*viz* the dialogue acts) and the representation of the context (*viz* the information state).

For purposes of readability, empty feature structures have been replaced by a minus and feature structure of the type *Semantic-Object-FS* have been replaced by an abbreviated representation.

The following parametrised representation

$$\text{Obj}_{\langle \text{TYPE}, \text{RECORDING} \rangle}(\text{GENRE}, \text{ARTIST}, \text{TITLE}).$$

actually means the following parametrised feature structure:

$$\left[ \begin{array}{l} \text{TYPE} \quad \text{NAME} \\ \text{DESC} \quad \left[ \begin{array}{l} \text{ARTIST} \quad \text{ARTIST} \\ \text{TITLE} \quad \text{TITLE} \\ \text{GENRE} \quad \text{GENRE} \end{array} \right] \\ \text{REF} \quad \text{RECORDING} \end{array} \right]$$

### A.1 Utterance (i)

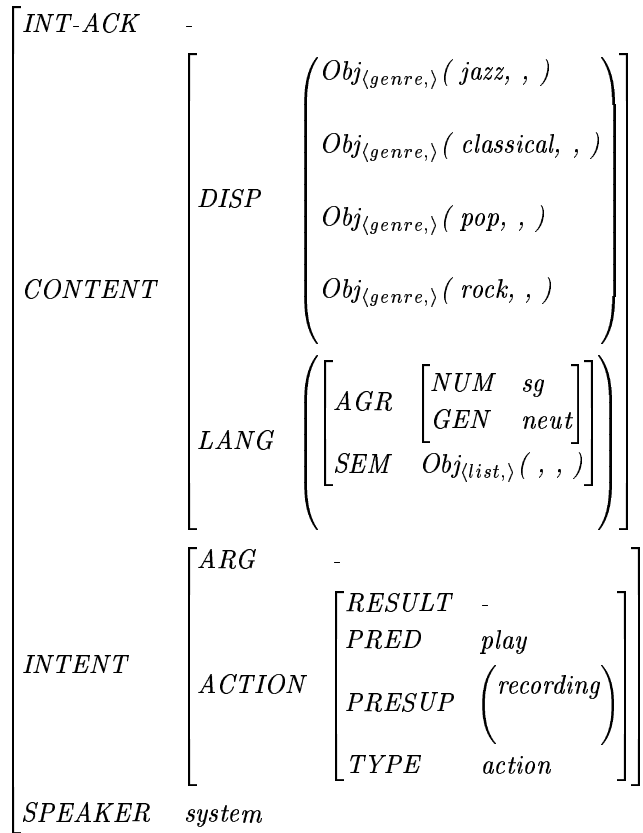
**System:**

*Select an artist an I will play it for you.  
Here is a list of genres.*

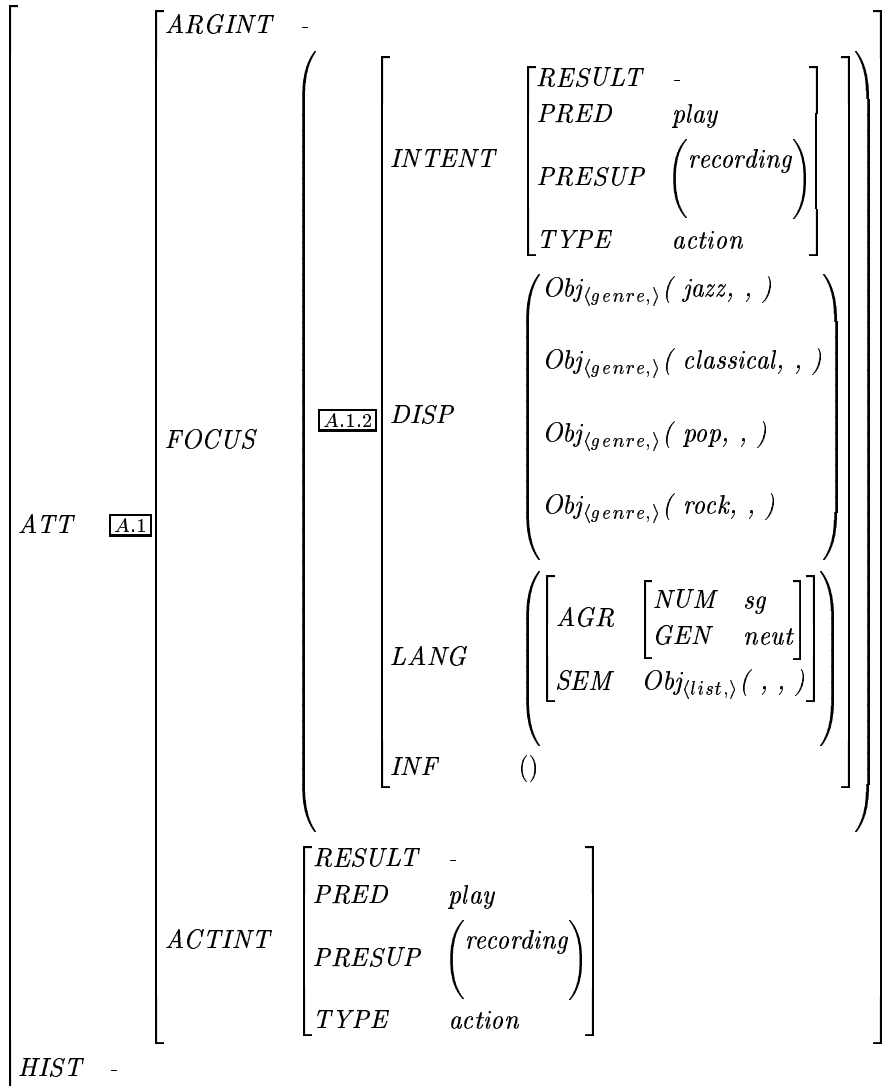
- |   |
|---|
| <ol style="list-style-type: none"><li>1. jazz</li><li>2. classical</li><li>3. pop</li><li>4. rock</li></ol> |
|---|

---

A.1.1 Dialogue Act



**A.1.2 Information State**



**A.2 Utterance (ii)**

**User:**

*Is there anything from Maceo*

**A.2.1 Dialogue Act**

<i>INT-ACK</i>	-
<i>CONTENT</i>	$\left[ \begin{array}{l} \text{DISP} \quad () \\ \text{LANG} \quad \left( \left[ \begin{array}{l} \text{AGR} \quad \left[ \begin{array}{l} \text{NUM} \quad \text{sg} \\ \text{GEN} \quad \text{masc} \end{array} \right] \\ \text{SEM} \quad \text{Obj}_{\langle \text{artist}, \rangle} (, \text{maceo}, ) \end{array} \right] \right) \end{array} \right]$
<i>INTENT</i>	$\left[ \begin{array}{l} \text{ARG} \quad \left[ \begin{array}{l} \text{RESULT} \quad \text{Obj}_{\langle \text{artist}, \rangle} (, \text{maceo}, ) \\ \text{PRED} \quad \text{artist} \\ \text{PRESUP} \quad () \\ \text{TYPE} \quad \text{argument} \end{array} \right] \\ \text{ACTION} \quad - \end{array} \right]$
<i>SPEAKER</i>	<i>user</i>

**A.2.2 Information State**

<i>ATT</i>	<i>ARGINT</i>	$\left[ \begin{array}{l} \text{RESULT} \quad \text{Obj}_{\langle \text{artist}, \rangle} (, \text{maceo}, ) \\ \text{PRESUP} \quad () \\ \text{TYPE} \quad \text{argument} \\ \text{PRED} \quad \text{artist} \end{array} \right]$
	<i>FOCUS</i>	$\left( \begin{array}{l} \boxed{\text{A.1.2}} \\ \left[ \begin{array}{l} \text{INTENT} \quad \left[ \begin{array}{l} \text{RESULT} \quad \text{Obj}_{\langle \text{artist}, \rangle} (, \text{maceo}, ) \\ \text{PRESUP} \quad () \\ \text{TYPE} \quad \text{argument} \\ \text{PRED} \quad \text{artist} \end{array} \right] \\ \boxed{\text{A.2.2}} \text{DISP} \quad () \\ \text{LANG} \quad \left( \left[ \begin{array}{l} \text{AGR} \quad \left[ \begin{array}{l} \text{NUM} \quad \text{sg} \\ \text{GEN} \quad \text{masc} \end{array} \right] \\ \text{SEM} \quad \text{Obj}_{\langle \text{artist}, \rangle} (, \text{maceo}, ) \end{array} \right] \right) \\ \text{INF} \quad () \end{array} \right)$
	<i>ACTINT</i>	$\left[ \begin{array}{l} \text{RESULT} \quad - \\ \text{PRED} \quad \text{play} \\ \text{PRESUP} \quad (\text{recording}) \\ \text{TYPE} \quad \text{action} \end{array} \right]$
<i>HIST</i>	$\boxed{\text{A.1}}$	

**A.3 Utterance (iii)****System:***I don't have an artist called Maceo.*

1. jazz
2. classical
3. pop
4. rock

**A.3.1 Dialogue Act**

INT-ACK	reject
CONTENT	-
INTENT	-
SPEAKER	system

**A.3.2 Information State**

ATT	A.1
HIST	A.1

**A.4 Utterance (iv)****User:***I want a jazz song.***A.4.1 Dialogue Act**

INT-ACK	-												
CONTENT	<table border="1"> <tr><td>DISP</td><td>()</td></tr> <tr><td>LANG</td><td> <table border="1"> <tr><td>AGR</td><td> <table border="1"> <tr><td>NUM</td><td>sg</td></tr> <tr><td>GEN</td><td>neut</td></tr> </table> </td></tr> <tr><td>SEM</td><td>Obj<sub>{genre,}</sub>(,jazz,)</td></tr> </table> </td></tr> </table>	DISP	()	LANG	<table border="1"> <tr><td>AGR</td><td> <table border="1"> <tr><td>NUM</td><td>sg</td></tr> <tr><td>GEN</td><td>neut</td></tr> </table> </td></tr> <tr><td>SEM</td><td>Obj<sub>{genre,}</sub>(,jazz,)</td></tr> </table>	AGR	<table border="1"> <tr><td>NUM</td><td>sg</td></tr> <tr><td>GEN</td><td>neut</td></tr> </table>	NUM	sg	GEN	neut	SEM	Obj <sub>{genre,}</sub> (,jazz,)
DISP	()												
LANG	<table border="1"> <tr><td>AGR</td><td> <table border="1"> <tr><td>NUM</td><td>sg</td></tr> <tr><td>GEN</td><td>neut</td></tr> </table> </td></tr> <tr><td>SEM</td><td>Obj<sub>{genre,}</sub>(,jazz,)</td></tr> </table>	AGR	<table border="1"> <tr><td>NUM</td><td>sg</td></tr> <tr><td>GEN</td><td>neut</td></tr> </table>	NUM	sg	GEN	neut	SEM	Obj <sub>{genre,}</sub> (,jazz,)				
AGR	<table border="1"> <tr><td>NUM</td><td>sg</td></tr> <tr><td>GEN</td><td>neut</td></tr> </table>	NUM	sg	GEN	neut								
NUM	sg												
GEN	neut												
SEM	Obj <sub>{genre,}</sub> (,jazz,)												
INTENT	<table border="1"> <tr><td>ARG</td><td> <table border="1"> <tr><td>RESULT</td><td>Obj<sub>{genre,}</sub>(jazz, ,)</td></tr> <tr><td>PRED</td><td>genre</td></tr> <tr><td>PRESUP</td><td>()</td></tr> <tr><td>TYPE</td><td>argument</td></tr> </table> </td></tr> <tr><td>ACTION</td><td>-</td></tr> </table>	ARG	<table border="1"> <tr><td>RESULT</td><td>Obj<sub>{genre,}</sub>(jazz, ,)</td></tr> <tr><td>PRED</td><td>genre</td></tr> <tr><td>PRESUP</td><td>()</td></tr> <tr><td>TYPE</td><td>argument</td></tr> </table>	RESULT	Obj <sub>{genre,}</sub> (jazz, ,)	PRED	genre	PRESUP	()	TYPE	argument	ACTION	-
ARG	<table border="1"> <tr><td>RESULT</td><td>Obj<sub>{genre,}</sub>(jazz, ,)</td></tr> <tr><td>PRED</td><td>genre</td></tr> <tr><td>PRESUP</td><td>()</td></tr> <tr><td>TYPE</td><td>argument</td></tr> </table>	RESULT	Obj <sub>{genre,}</sub> (jazz, ,)	PRED	genre	PRESUP	()	TYPE	argument				
RESULT	Obj <sub>{genre,}</sub> (jazz, ,)												
PRED	genre												
PRESUP	()												
TYPE	argument												
ACTION	-												
SPEAKER	user												

A.4.2 Information State

ATT	ARGINT	$\left[ \begin{array}{l} \text{RESULT } Obj_{\langle genre, \rangle}(jazz, ,) \text{ PRESUP } () \\ \text{TYPE } argument \\ \text{PRED } artist \end{array} \right]$
	FOCUS	$\left( \begin{array}{l} \boxed{\text{A.1.2}} \\ \left[ \begin{array}{l} \text{RESULT } Obj_{\langle genre, \rangle}(jazz, ,) \\ \text{PRESUP } () \\ \text{TYPE } argument \\ \text{PRED } genre \end{array} \right] \\ \text{DISP } () \\ \text{LANG } \left( \begin{array}{l} \text{AGR } \left[ \begin{array}{l} \text{NUM } sg \\ \text{GEN } neut \end{array} \right] \\ \text{SEM } Obj_{\langle genre, \rangle}(jazz, ,) \end{array} \right) \\ \text{INF } () \end{array} \right)$
	ACTINT	$\left[ \begin{array}{l} \text{RESULT } - \\ \text{PRESUP } (recording) \\ \text{TYPE } action \\ \text{PRED } play \end{array} \right]$
HIST	$\boxed{\text{A.1}}$	

## A.5 Utterance (v)

**System:****Please, choose a jazz artist.**

1. John Coltrane
2. Miles Davis
3. Klaus Graf
4. Nils Lundgren
5. Wynton Marsalis
6. Maceo Parker

## A.5.1 Dialogue Act

<i>INT-ACK</i>	-	
<i>CONTENT</i>	<i>DISP</i>	$\left( \begin{array}{l} \text{Obj}_{\langle \text{artist}, \rangle}(\text{jazz}, \text{john coltrane}, ) \\ \text{Obj}_{\langle \text{artist}, \rangle}(\text{jazz}, \text{miles davis}, ) \\ \text{Obj}_{\langle \text{artist}, \rangle}(\text{jazz}, \text{klaus graf}, ) \\ \text{Obj}_{\langle \text{artist}, \rangle}(\text{jazz}, \text{nils lundgren}, ) \\ \text{Obj}_{\langle \text{artist}, \rangle}(\text{jazz}, \text{wynton marsalis}, ) \\ \text{Obj}_{\langle \text{artist}, \rangle}(\text{jazz}, \text{maceo parker}, ) \end{array} \right)$
	<i>LANG</i>	$\left( \begin{array}{l} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ neut} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{genre}, \rangle}(\text{jazz}, , ) \end{array} \right] \end{array} \right)$
<i>INTENT</i>	-	
<i>SPEAKER</i>		<i>system</i>



## A.6 Utterance (vi)

*User:**Maceo.*

## A.6.1 Dialogue Act

<i>INT-ACK</i>	-	
<i>CONTENT</i>	$\left[ \begin{array}{l} \textit{DISP} \quad () \\ \textit{LANG} \quad \left( \left[ \begin{array}{l} \textit{AGR} \quad \left[ \begin{array}{l} \textit{NUM} \quad \textit{sg} \\ \textit{GEN} \quad \textit{masc} \end{array} \right] \\ \textit{SEM} \quad \textit{Obj}_{\langle \textit{artist}, \rangle} (\textit{jazz}, \textit{maceo parker}, ) \end{array} \right] \right) \end{array} \right]$	
<i>INTENT</i>	<i>ARG</i>	$\left[ \begin{array}{l} \textit{RESULT} \quad \textit{Obj}_{\langle \textit{artist}, \rangle} (\textit{jazz}, \textit{maceo parker}, ) \\ \textit{PRED} \quad \textit{artist} \\ \textit{PRESUP} \quad () \\ \textit{TYPE} \quad \textit{argument} \end{array} \right]$
<i>SPEAKER</i>	<i>ACTION</i> -	
	<i>user</i>	

**A.6.2 Information State**

<i>ATT</i>	<i>ARGINT</i>	$\left[ \begin{array}{l} \textit{RESULT} \quad \textit{Obj}_{\langle \textit{artist}, \rangle}(\textit{jazz}, \textit{maceo parker},) \\ \textit{PRESUP} \quad () \\ \textit{TYPE} \quad \textit{argument} \\ \textit{PRED} \quad \textit{artist} \end{array} \right]$
<i>ATT</i>	<i>FOCUS</i>	$\left( \begin{array}{l} \boxed{\text{A.1.2}} \\ \boxed{\text{A.5.2}} \\ \left[ \begin{array}{l} \textit{INTENT} \quad \left[ \begin{array}{l} \textit{RESULT} \quad \textit{Obj}_{\langle \textit{artist}, \rangle}(\textit{jazz}, \textit{maceo parker},) \\ \textit{PRESUP} \quad () \\ \textit{TYPE} \quad \textit{argument} \\ \textit{PRED} \quad \textit{artist} \end{array} \right] \\ \textit{DISP} \quad () \\ \textit{LANG} \quad \left( \left[ \begin{array}{l} \textit{AGR} \quad \left[ \begin{array}{l} \textit{NUM} \quad \textit{sg} \\ \textit{GEN} \quad \textit{masc} \end{array} \right] \\ \textit{SEM} \quad \textit{Obj}_{\langle \textit{artist}, \rangle}(\textit{jazz}, \textit{maceo parker},) \end{array} \right] \right) \\ \textit{INF} \quad () \end{array} \right)$
<i>ATT</i>	<i>ACTINT</i>	$\left[ \begin{array}{l} \textit{RESULT} \quad - \\ \textit{PRESUP} \quad (\textit{recording}) \\ \textit{TYPE} \quad \textit{action} \\ \textit{PRED} \quad \textit{play} \end{array} \right]$
<i>HIST</i>	$\boxed{\text{A.5}}$	

## A.7 Utterance (vii)

**System:***I have these jazz recordings from Maceo Parker.*

1. *Black Widow*
2. *Coin Toss*
3. *Dial Maceo*
4. *My Baby Loves You*

## A.7.1 Dialogue Act

<i>INT-ACK</i>	-				
<i>CONTENT</i>	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border-right: 1px solid black; padding-right: 10px; vertical-align: middle;"><i>DISP</i></td> <td style="padding-left: 10px;"> <math display="block">\left( \begin{array}{l} \text{Obj}_{\langle \text{recording}, mp0 \rangle}(\text{jazz, maceo parker, black widow}) \\ \text{Obj}_{\langle \text{recording}, mp1 \rangle}(\text{jazz, maceo parker, coin toss}) \\ \text{Obj}_{\langle \text{recording}, mp2 \rangle}(\text{jazz, maceo parker, dial maceo}) \\ \text{Obj}_{\langle \text{recording}, mp3 \rangle}(\text{jazz, maceo parker, my baby loves you}) \end{array} \right)</math> </td> </tr> <tr> <td style="border-right: 1px solid black; padding-right: 10px; vertical-align: middle;"><i>LANG</i></td> <td style="padding-left: 10px;"> <math display="block">\left( \begin{array}{l} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ neut} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{genre}, \rangle}(\text{jazz, ,}) \end{array} \right] \\ \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ masc} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{artist}, \rangle}(\text{jazz, maceo parker ,}) \end{array} \right] \end{array} \right)</math> </td> </tr> </table>	<i>DISP</i>	$\left( \begin{array}{l} \text{Obj}_{\langle \text{recording}, mp0 \rangle}(\text{jazz, maceo parker, black widow}) \\ \text{Obj}_{\langle \text{recording}, mp1 \rangle}(\text{jazz, maceo parker, coin toss}) \\ \text{Obj}_{\langle \text{recording}, mp2 \rangle}(\text{jazz, maceo parker, dial maceo}) \\ \text{Obj}_{\langle \text{recording}, mp3 \rangle}(\text{jazz, maceo parker, my baby loves you}) \end{array} \right)$	<i>LANG</i>	$\left( \begin{array}{l} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ neut} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{genre}, \rangle}(\text{jazz, ,}) \end{array} \right] \\ \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ masc} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{artist}, \rangle}(\text{jazz, maceo parker ,}) \end{array} \right] \end{array} \right)$
<i>DISP</i>	$\left( \begin{array}{l} \text{Obj}_{\langle \text{recording}, mp0 \rangle}(\text{jazz, maceo parker, black widow}) \\ \text{Obj}_{\langle \text{recording}, mp1 \rangle}(\text{jazz, maceo parker, coin toss}) \\ \text{Obj}_{\langle \text{recording}, mp2 \rangle}(\text{jazz, maceo parker, dial maceo}) \\ \text{Obj}_{\langle \text{recording}, mp3 \rangle}(\text{jazz, maceo parker, my baby loves you}) \end{array} \right)$				
<i>LANG</i>	$\left( \begin{array}{l} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ neut} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{genre}, \rangle}(\text{jazz, ,}) \end{array} \right] \\ \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ masc} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{artist}, \rangle}(\text{jazz, maceo parker ,}) \end{array} \right] \end{array} \right)$				
<i>INTENT</i>	-				
<i>SPEAKER</i>	<i>system</i>				



## A.8 Utterance (viii)

*User:*

*Show me the second artist.*

## A.8.1 Dialogue Act

<i>INT-ACK</i>	-	
<i>CONTENT</i>	<i>LANG</i>	$\left( \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ masc} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{artist}, \rangle} (\text{jazz}, \text{miles davis},) \end{array} \right] \right)$
<i>INTENT</i>	<i>ARG</i>	$\left[ \begin{array}{l} \text{RESULT} \text{ Obj}_{\langle \text{artist}, \rangle} (\text{jazz}, \text{miles davis},) \\ \text{PRED} \text{ artist} \\ \text{PRESUP} () \\ \text{TYPE} \text{ argument} \end{array} \right]$
<i>SPEAKER</i>	<i>ACTION</i>	-
		<i>user</i>

A.8.2 Information State

ATT	A.8	ARGINT	$\left[ \begin{array}{l} RESULT \quad Obj_{(artist,)}(jazz, miles davis,) \\ PRESUP \quad () \\ TYPE \quad argument \\ PRED \quad artist \end{array} \right]$
		FOCUS	$\left( \begin{array}{l} \left[ \begin{array}{l} A.1.2 \\ A.5.2 \end{array} \right] \\ \left[ \begin{array}{l} INTENT \\ \left[ \begin{array}{l} RESULT \quad Obj_{(artist,)}(jazz, miles davis,) \\ PRESUP \quad () \\ TYPE \quad argument \\ PRED \quad artist \end{array} \right] \end{array} \right] \\ DISP \quad () \\ LANG \quad \left( \begin{array}{l} AGR \quad \left[ \begin{array}{l} NUM \quad sg \\ GEN \quad masc \end{array} \right] \\ SEM \quad Obj_{(artist,)}(jazz, miles davis,) \end{array} \right) \\ INF \quad () \end{array} \right)$
		ACTINT	$\left[ \begin{array}{l} RESULT \quad - \\ PRESUP \quad (recording) \\ TYPE \quad action \\ PRED \quad play \end{array} \right]$
HIST	A.7		

## A.9 Utterance (ix)

**System:**

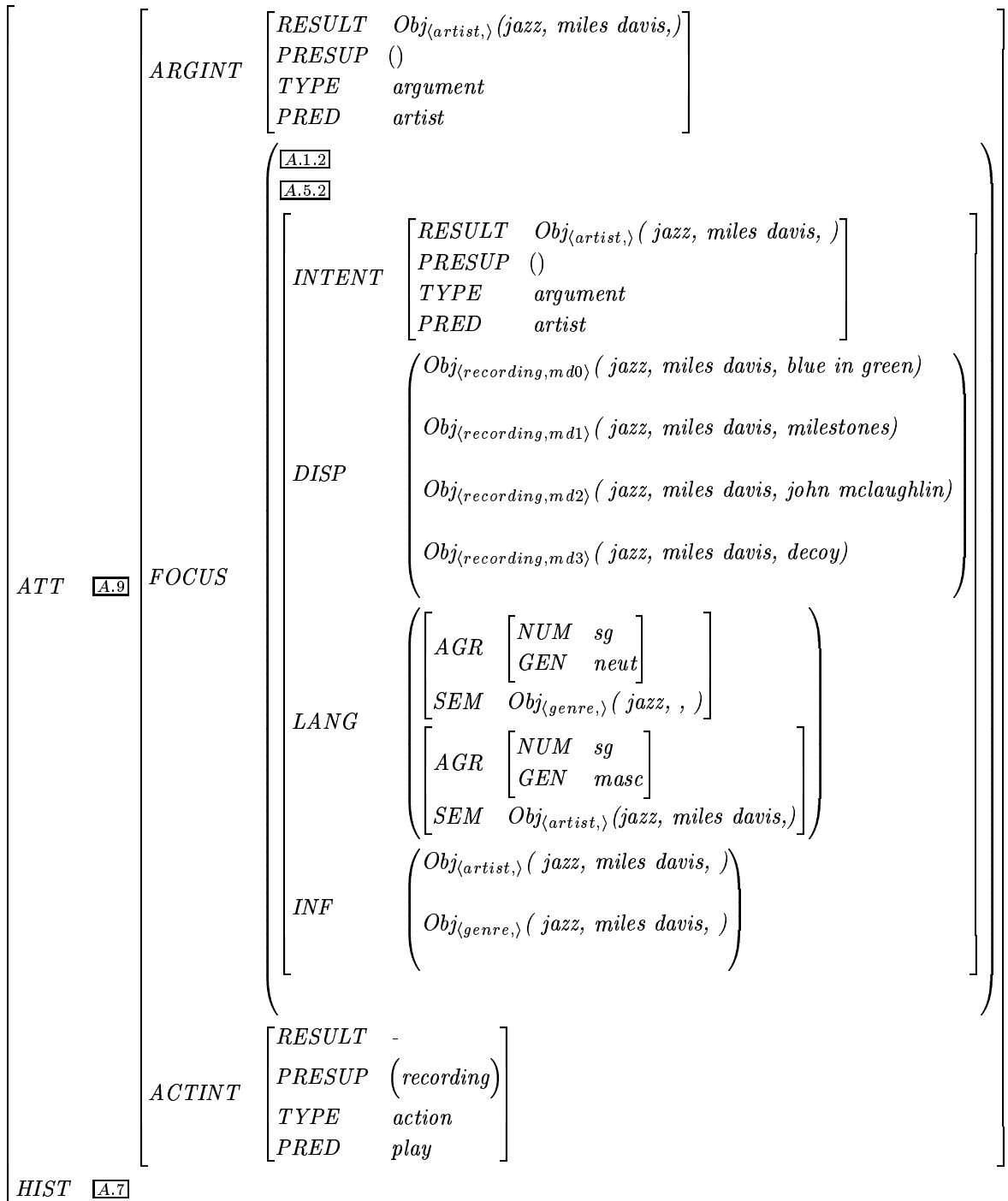
*Choose a jazz recording from Miles Davis.*

1. *Blue In Green*
2. *Milestones*
3. *John McLaughlin*
4. *Decoy*

## A.9.1 Dialogue Act

<i>INT-ACK</i>	-				
<i>CONTENT</i>	<table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border-right: 1px solid black; padding: 10px; vertical-align: middle;"><i>DISP</i></td> <td style="padding: 10px;"> <math display="block">\left( \begin{array}{l} \text{Obj}_{\langle \text{recording}, \text{md}0 \rangle}(\text{jazz}, \text{miles davis}, \text{blue in green}) \\ \text{Obj}_{\langle \text{recording}, \text{md}1 \rangle}(\text{jazz}, \text{miles davis}, \text{milestones}) \\ \text{Obj}_{\langle \text{recording}, \text{md}2 \rangle}(\text{jazz}, \text{miles davis}, \text{john mclaughlin}) \\ \text{Obj}_{\langle \text{recording}, \text{md}3 \rangle}(\text{jazz}, \text{miles davis}, \text{decoy}) \end{array} \right)</math> </td> </tr> <tr> <td style="border-right: 1px solid black; padding: 10px; vertical-align: middle;"><i>LANG</i></td> <td style="padding: 10px;"> <math display="block">\left( \begin{array}{l} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ neut} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{genre}, \rangle}(\text{jazz}, , ) \end{array} \right] \\ \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ masc} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{artist}, \rangle}(\text{jazz}, \text{miles davis}, ) \end{array} \right] \end{array} \right)</math> </td> </tr> </table>	<i>DISP</i>	$\left( \begin{array}{l} \text{Obj}_{\langle \text{recording}, \text{md}0 \rangle}(\text{jazz}, \text{miles davis}, \text{blue in green}) \\ \text{Obj}_{\langle \text{recording}, \text{md}1 \rangle}(\text{jazz}, \text{miles davis}, \text{milestones}) \\ \text{Obj}_{\langle \text{recording}, \text{md}2 \rangle}(\text{jazz}, \text{miles davis}, \text{john mclaughlin}) \\ \text{Obj}_{\langle \text{recording}, \text{md}3 \rangle}(\text{jazz}, \text{miles davis}, \text{decoy}) \end{array} \right)$	<i>LANG</i>	$\left( \begin{array}{l} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ neut} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{genre}, \rangle}(\text{jazz}, , ) \end{array} \right] \\ \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ masc} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{artist}, \rangle}(\text{jazz}, \text{miles davis}, ) \end{array} \right] \end{array} \right)$
<i>DISP</i>	$\left( \begin{array}{l} \text{Obj}_{\langle \text{recording}, \text{md}0 \rangle}(\text{jazz}, \text{miles davis}, \text{blue in green}) \\ \text{Obj}_{\langle \text{recording}, \text{md}1 \rangle}(\text{jazz}, \text{miles davis}, \text{milestones}) \\ \text{Obj}_{\langle \text{recording}, \text{md}2 \rangle}(\text{jazz}, \text{miles davis}, \text{john mclaughlin}) \\ \text{Obj}_{\langle \text{recording}, \text{md}3 \rangle}(\text{jazz}, \text{miles davis}, \text{decoy}) \end{array} \right)$				
<i>LANG</i>	$\left( \begin{array}{l} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ neut} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{genre}, \rangle}(\text{jazz}, , ) \end{array} \right] \\ \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ masc} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{artist}, \rangle}(\text{jazz}, \text{miles davis}, ) \end{array} \right] \end{array} \right)$				
<i>INTENT</i>	-				
<i>SPEAKER</i>	<i>system</i>				

A.9.2 Information State



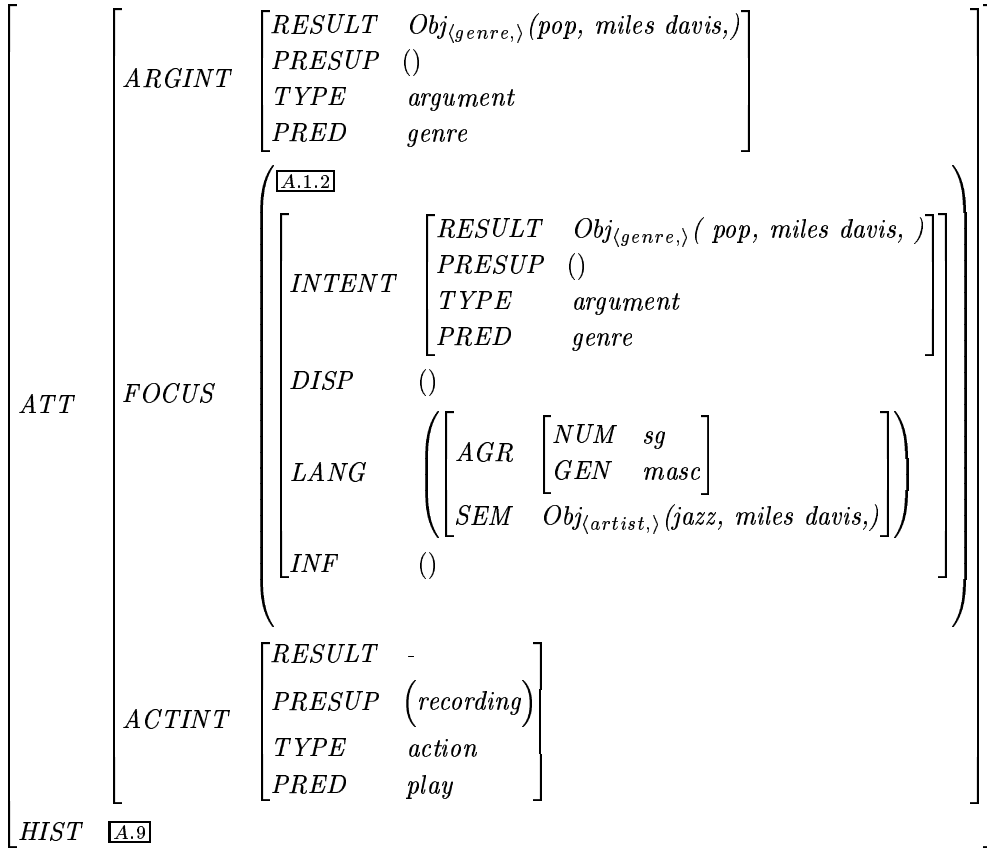
## A.10 Utterance (x)

*User:*

*Is there any pop from him?*

A.10.1 Dialogue Act	<div style="display: flex; flex-direction: column; align-items: flex-start;"> <div style="margin-bottom: 10px;"> <p><i>INT-ACK</i> -</p> </div> <div style="margin-bottom: 10px;"> <p><i>CONTENT</i></p> <div style="display: flex; align-items: center; margin-left: 20px;"> <div style="margin-right: 10px;"><i>LANG</i></div> <div style="border-left: 1px solid black; padding-left: 10px;"> <div style="display: flex; flex-direction: column; align-items: center;"> <div style="margin-bottom: 10px;"> <p><i>DISP</i> ()</p> </div> <div style="margin-bottom: 10px;"> <p><i>AGR</i> <math>\left[ \begin{array}{l} \text{NUM } sg \\ \text{GEN } neut \end{array} \right]</math></p> </div> <div style="margin-bottom: 10px;"> <p><i>SEM</i> <math>Obj_{(genre,)}(pop, , )</math></p> </div> <div style="margin-bottom: 10px;"> <p><i>AGR</i> <math>\left[ \begin{array}{l} \text{NUM } sg \\ \text{GEN } masc \end{array} \right]</math></p> </div> <div> <p><i>SEM</i> <math>Obj_{(artist,)}(jazz, miles davis, )</math></p> </div> </div> </div> </div> </div> <div style="margin-bottom: 10px;"> <p><i>INTENT</i></p> <div style="display: flex; align-items: center; margin-left: 20px;"> <div style="margin-right: 10px;"><i>ARG</i></div> <div style="border-left: 1px solid black; padding-left: 10px;"> <div style="display: flex; flex-direction: column; align-items: center;"> <div style="margin-bottom: 5px;"><i>RESULT</i> <math>Obj_{(genre,)}(pop, miles davis, )</math></div> <div style="margin-bottom: 5px;"><i>PRED</i> genre</div> <div style="margin-bottom: 5px;"><i>PRESUP</i> ()</div> <div><i>TYPE</i> argument</div> </div> </div> </div> </div> <div> <p><i>ACTION</i> -</p> </div> <div style="margin-top: 10px;"> <p><i>SPEAKER</i> user</p> </div> </div>
---------------------	--

**A.10.2 Information State**



## A.11 Utterance (xi)

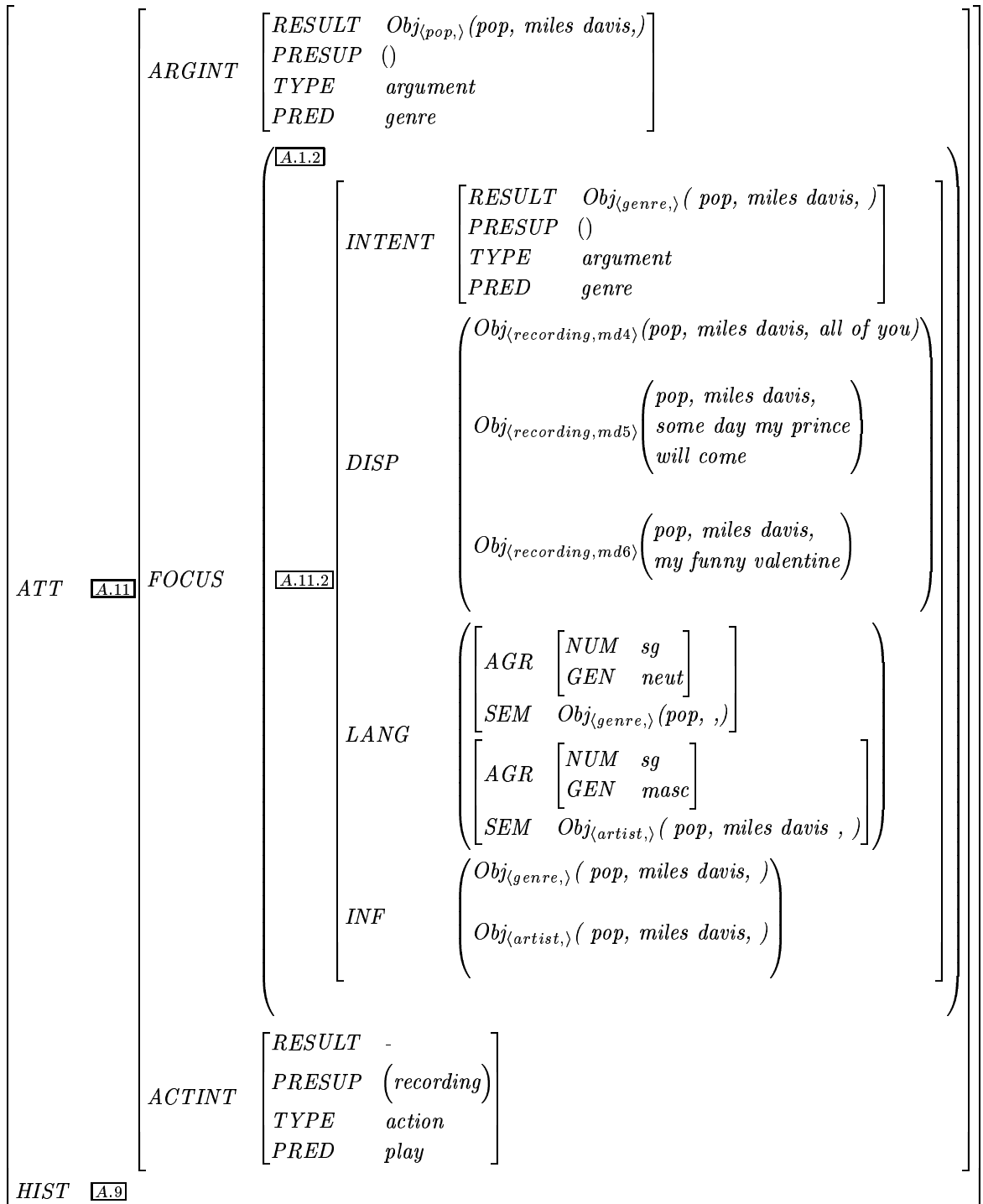
**System:***These are the pop recordings from Miles Davis.*

1. *All Of You*
2. *Some Day My Prince Will Come*
3. *My Funny Valentine*

## A.11.1 Dialogue Act

<i>INT-ACK</i>	-	
<i>CONTENT</i>	<i>DISP</i>	$\left( \begin{array}{l} \text{Obj}_{\langle \text{recording}, \text{md4} \rangle}(\text{pop}, \text{miles davis}, \text{all of you}) \\ \text{Obj}_{\langle \text{recording}, \text{md5} \rangle}(\text{pop}, \text{miles davis}, \\ \text{some day my prince will come}) \\ \text{Obj}_{\langle \text{recording}, \text{md6} \rangle}(\text{pop}, \text{miles davis}, \text{my funny valentine}) \end{array} \right)$
<i>LANG</i>	<i>LANG</i>	$\left( \begin{array}{l} \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ neut} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{genre}, \rangle}(\text{pop}, , ) \end{array} \right] \\ \left[ \begin{array}{l} \text{AGR} \left[ \begin{array}{l} \text{NUM} \text{ sg} \\ \text{GEN} \text{ masc} \end{array} \right] \\ \text{SEM} \text{ Obj}_{\langle \text{artist}, \rangle}(\text{pop}, \text{miles davis}, , ) \end{array} \right] \end{array} \right)$
<i>INTENT</i>	-	
<i>SPEAKER</i>	<i>system</i>	

A.11.2 Information State



## A.12 Utterance (xii)

*User:*

*Play the last one.*

## A.12.1 Dialogue Act

<i>INT-ACK</i>	-
<i>CONTENT</i>	$\left[ \begin{array}{l} \textit{DISP} \quad () \\ \textit{LANG} \quad \left( \left[ \begin{array}{l} \textit{AGR} \quad \left[ \begin{array}{l} \textit{NUM} \quad \textit{sg} \\ \textit{GEN} \quad \textit{neut} \end{array} \right] \\ \textit{SEM} \quad \textit{Obj}_{(\textit{recording}, \textit{md6})} \left( \begin{array}{l} \textit{pop}, \textit{miles davis}, \\ \textit{my funny valentine} \end{array} \right) \end{array} \right] \right) \end{array} \right]$
<i>INTENT</i>	$\left[ \begin{array}{l} \textit{ARG} \quad \left[ \begin{array}{l} \textit{RESULT} \quad \textit{Obj}_{(\textit{recording}, \textit{md6})} \left( \begin{array}{l} \textit{pop}, \textit{miles davis}, \\ \textit{my funny valentine} \end{array} \right) \\ \textit{PRED} \quad \textit{recording} \\ \textit{PRESUP} \quad () \\ \textit{TYPE} \quad \textit{argument} \end{array} \right] \end{array} \right]$
<i>SPEAKER</i>	- <i>user</i>

**A.12.2 Information State**

<i>ATT</i>	<i>ARGINT</i>	$\left[ \begin{array}{l} \textit{RESULT} \textit{ Obj}_{\langle \textit{recording}, \textit{md6} \rangle} \left( \begin{array}{l} \textit{pop}, \textit{miles davis}, \\ \textit{my funny valentine} \end{array} \right) \\ \textit{PRESUP} \textit{ } () \\ \textit{TYPE} \textit{ argument} \\ \textit{PRED} \textit{ recording} \end{array} \right]$
		$\left( \begin{array}{l} \boxed{\text{A.1.2}} \\ \boxed{\text{A.11.2}} \\ \left[ \begin{array}{l} \textit{INTENT} \left[ \begin{array}{l} \textit{RESULT} \textit{ Obj}_{\langle \textit{recording}, \textit{md6} \rangle} \left( \begin{array}{l} \textit{pop}, \textit{miles davis}, \\ \textit{my funny valentine} \end{array} \right) \\ \textit{PRESUP} \textit{ } () \\ \textit{TYPE} \textit{ argument} \\ \textit{PRED} \textit{ recording} \end{array} \right] \\ \textit{DISP} \textit{ } () \\ \textit{LANG} \left( \begin{array}{l} \textit{AGR} \left[ \begin{array}{l} \textit{NUM} \textit{ sg} \\ \textit{GEN} \textit{ neut} \end{array} \right] \\ \textit{SEM} \textit{ Obj}_{\langle \textit{recording}, \textit{md6} \rangle} \left( \begin{array}{l} \textit{pop}, \textit{miles davis}, \\ \textit{my funny valentine} \end{array} \right) \end{array} \right) \\ \textit{INF} \textit{ } () \end{array} \right) \end{array} \right)$
	<i>ACTINT</i>	$\left[ \begin{array}{l} \textit{RESULT} \textit{ Obj}_{\langle \textit{recording}, \textit{md6} \rangle} \left( \begin{array}{l} \textit{pop}, \textit{miles davis}, \\ \textit{my funny valentine} \end{array} \right) \\ \textit{PRESUP} \textit{ } () \\ \textit{TYPE} \textit{ action} \\ \textit{PRED} \textit{ play} \end{array} \right]$
<i>HIST</i>	$\boxed{\text{A.11}}$	

**A.13 Utterance (xiii)**

**System:**

**OK.**

**A.13.1 Dialogue Act**

<i>INT-ACK</i>	<i>accept</i>
<i>CONTENT</i>	-
<i>INTENT</i>	-
<i>SPEAKER</i>	<i>system</i>

## A.13.2 Information State

ATT	ARGINT $\left[ \begin{array}{l} \text{RESULT } \text{Obj}_{\langle \text{recording}, \text{md6} \rangle} \left( \begin{array}{l} \text{pop, miles davis,} \\ \text{my funny valentine} \end{array} \right) \\ \text{PRESUP } () \\ \text{TYPE } \text{argument} \\ \text{PRED } \text{artist} \end{array} \right]$	
ATT	FOCUS $\left( \begin{array}{l} \boxed{\text{A.1.2}} \\ \boxed{\text{A.11.2}} \\ \left[ \begin{array}{l} \text{INTENT} \left[ \begin{array}{l} \text{RESULT } \text{Obj}_{\langle \text{recording}, \text{md6} \rangle} \left( \begin{array}{l} \text{pop, miles davis,} \\ \text{my funny valentine} \end{array} \right) \\ \text{PRESUP } () \\ \text{TYPE } \text{argument} \\ \text{PRED } \text{recording} \end{array} \right] \\ \text{DISP } () \\ \text{LANG } () \\ \text{INF } () \end{array} \right] \end{array} \right)$	
ATT	ACTINT $\left[ \begin{array}{l} \text{RESULT } \text{Obj}_{\langle \text{recording}, \text{md6} \rangle} \left( \begin{array}{l} \text{pop, miles davis,} \\ \text{my funny valentine} \end{array} \right) \\ \text{PRESUP } () \\ \text{TYPE } \text{action} \\ \text{PRED } \text{play} \end{array} \right]$	
HIST	$\boxed{\text{A.11}}$	

---

# Bibliography

- Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L. and Stent, A.: 2001, Towards conversational human-computer interaction, *AI Magazine* .
- Allen, J., Ferguson, G. and Stent, A.: 2001, An architecture for more realistic conversational systems, *Proceedings of Intelligent User Interfaces*, Santa Fe.
- Asher, N.: 1993, *Reference to Abstract Objects in Discourse*, Kluwer Academic Publishers, Dordrecht.
- Asher, N. and Lascarides, A.: 1997, Bridging, *Journal of Semantics* **15**, 83–113.  
\*citeseer.nj.nec.com/asher97bridging.html
- Aust, H., Oerder, M., Seide, F. and Steinbiss, V.: 1995, The philips automatic train timetable information system, *Speech Communication* **17**, 249–262.
- Beaver, D.: 1997, Presuppositions, in J. van Benthem and A. T. Meulen (eds), *Handbook of logic and language*, Elsevier, Amsterdam.
- Bernsen, N. O., Dybkjær, H. and Dybkjær, L.: 1996, Principles for the design of cooperative spoken human-machine dialogue, *Proc. ICSLP '96*, Vol. 2, Philadelphia, PA, pp. 729–732.
- Bernsen, N. O. and Dybkjær, L.: 2000, From single word to natural dialogue, in M. V. Zelkowitz (ed.), *Advances in Computers*, Vol. 52, Academic Press, London, pp. 267–327.
- Bernsen, N. O., Dybkjær, L. and Dybkjær, H.: 1994, A dedicated task-oriented dialog theory in support of spoken language dialog systems design, *Proc ICSLP International Conference on Spoken Language Processing*, Yokohama, pp. 875–878.
- Bos, J. and Heine, J.: 2000, Discourse and dialog semantics for translation, in Wahlster (2000).
- Bray, T., Paoli, J. and Sperberg-McQueen, C. M.: 2000, Extensible markup language (xml) 1.0 (second edition), *Technical report*, W3C.

- Bücher, K., Forkl, Y., Görz, G., Klarner, M. and Ludwig, B.: 2001, Discourse and application modeling for dialogue systems, *KI-2001 Workshop on Applications of Description Logics*, Vol. 44, TU Wien CEUR Proceedings.
- Bunt, H.: 1994, Context and dialogue control, *THINK Quarterly* **3**(1), 19–31.
- Bunt, H.: 1997, Dynamic interpretation and dialogue theory, in M. Taylor, D. Bouwhuis and F. Neel (eds), *The Structure of Multi-Modal Dialogue*, Vol. 2, John Benjamins, Amsterdam.
- Carletta, J., Isard, A., Isard, S., Kowtko, J., Doherty-Sneddon, G. and Anderson, A.: 1997, The reliability of a dialogue structure coding scheme, *Computational Linguistics* **23**(1), 13–31.
- Carlson, R.: 1996, The dialog component in the waxholm system, *Proceedings of the Twente Workshop on Language Technology (TWLT11) Dialogue Management in Natural Language Systems*.
- Cooper, R.: 1997, Information states, attitudes and dialogue, *Proceedings of the Second Tbilisi Symposium on Language, Logic and Computation*, Tbilisi.
- Core, M. G. and Allen, J. F.: 1997, Coding dialogues with the DAMSL annotation scheme, in D. Traum (ed.), *Working Notes: AAAI Fall Symposium on Communicative Action in Humans and Machines*, American Association for Artificial Intelligence, Menlo Park, California, pp. 28–35.  
\*citeseer.nj.nec.com/core97coding.html
- Dyade: 1999, Koala object markup language, *Technical report*, ILOG, Paris, France.
- EMBASSI Project*: 2002, <http://www.embassi.de>.
- Evans, G.: 1980, Pronouns, quantifiers and relative clauses, in M. Platts (ed.), *Reference, Truth and Reality*, Routledge, Kegan Paul, New York.
- Flycht-Eriksson, A.: 1999, A survey of knowledge sources in dialogue systems, in J. Alexandersson (ed.), *Proceedings of the IJCAI-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, International Joint Conference on Artificial Intelligence, Murray Hill, New Jersey, pp. 41–48.  
\*citeseer.nj.nec.com/328028.html
- Gallwitz, F., Aretoulaki, M., Boros, M., Haas, J., Harbeck, S., Huber, R., Niemann, H. and Nöth, E.: 1998, The Erlangen Spoken Dialogue System EVAR: A State-of-the-Art Information Retrieval System, *Proceedings of 1998 International Symposium on Spoken Dialogue (ISSD 98)*, Sydney, Australia, pp. 19–26.  
\*citeseer.nj.nec.com/gallwitz98erlangen.html
- Ge, N., Hale, J. and Charniak, E.: 1998, A statistical approach to anaphora resolution, *Proceedings of the Sixth Workshop on Very Large Corpora*, pp. 161–171.

- Gibbon, D., Moore, R. and Winski, R. (eds): 1997, *Handbook of Standards and Resources for Spoken Language System*, Mouton de Gruyter.
- Ginzburg, J.: 1996, Dynamics and the semantics of dialogue, in J. Seligman (ed.), *Language, Logic and Computation*, Vol. 1 of *CSLI Lecture Notes*, CSLI, Stanford.
- Groenendijk, J. and Stokhof, M.: 1991, Dynamic predicate logic, *Linguistics and Philosophy* **14**(1), 39–100.  
\*citeseer.nj.nec.com/groenendijk91dynamic.html
- Grosz, B.: 1995, Discourse and dialogue, *Survey of the State of the Art in Human Language Technology*.
- Grosz, B. and Sidner, C.: 1986, Attention, intentions and the structure of discourse, *Computational Linguistics* **12**(3), 175–204.
- Gundel, J. K., Hedberg, N. and Zacharski, R.: 1993, Cognitive status and the form of referring expressions in discourse, *Language* **69**(2), 274–307.
- Heim, I.: 1982, *The Semantics of Definite and Indefinite Noun Phrases*, PhD thesis, University of Massachusetts, Amherst.
- Hulstijn, J.: 2000, *Dialogue Models for Inquiry and Transaction*, PhD thesis, University of Twente.
- Kamp, H.: 1981, A theory of truth and semantic representation, in Groenendijk et al. (eds), *Formal Methods in the Study of Language*, Mathematisch Centrum, Amsterdam, pp. 277–322.
- Kamp, H.: 2001, The importance of presuppositions, in C. Rohrer and A. Rosdeutscher (eds), *Linguistic Form and its Computation*, CSLI. Selected papers from the SFB 340.
- Kamp, H. and Reyle, U.: 1993, *From Discourse to Logic*, Kluwer, Dordrecht.
- Kaplan, R. M. and Bresnan, J.: 1982, Lexical-functional grammar: A formal system for grammatical representation, in J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, MIT Press, chapter 4, pp. 173–281.
- Kehler, A.: 2000, Cognitive status and form of reference in multimodal human-computer interaction, *Proceedings of the 17th National Conference on Artificial Intelligence*, AAAI, Austin, Texas.
- Kievit, L. A., Piwek, P., Beun, R. and Bunt, H.: 2001, Multimodal cooperative resolution of referential expressions in the DenK system, in H. C. Bunt and R. J. Beun (eds), *Cooperative Multimodal Communication*, number 2155 in *Lecture Notes in Artificial Intelligence*, Springer Verlag, Berlin.

- Krahmer, E. and Piwek, P.: 2000, Varieties of anaphora: Introduction, in E. Krahmer and P. Piwek (eds), *Varieties of Anaphora*, ESSLLI, Birmingham, pp. 1–15.
- Labrou, Y. and Finin, T.: 1997, A proposal for a new KQML specification, *Technical Report TR CS-97-03*, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250. Also available online at <http://www.cs.umbc.edu/kqml/>.  
\*citeseer.nj.nec.com/labrou97proposal.html
- Larsson, S. and Traum, D. R.: 2000, Information state and dialogue management in the TRINDI dialogue move engine toolkit, *Natural Language Engineering* **6**(3-4), 323–340.
- Lascarides, A. and Asher, N.: 1993, Temporal interpretation, discourse relations and commonsense entailment, *Linguistics and Philosophy* **16**(5), 437–493.
- Lascarides, A. and Asher, N.: 1999, Cognitive states, discourse structure and the content of dialogue, *Proceedings to Amsteloque 1999*.
- Lewis, D.: 1979, Scorekeeping in a language game, *Journal of Philosophical Logic* **8**, 339–359.
- Lucke, H.: 1998, SUTK: An extensible and freely configurable speech dialogue research tool, *Internal report*, Sony International (Europe), ATCS.
- Ludwig, B.: 2001, Dialogue understanding in dynamic domains, *the 5th Workshop on Formal Semantics and Pragmatics of Dialogue (BI-DIALOG 2001)*, Bielefeld.
- Ludwig, B., Görz, G. and Niemann, H.: 2000, An inference-based approach to the interpretation of discourse, *Special Issue of the Journal of Language and Computation* **1**(2), 241–258.
- Luperfoy, S., Duff, D., Loehr, D., Harper, L., Miller, K. and Reeder, F.: 1998, An architecture for dialogue management, context tracking, and pragmatic adaptation in spoken dialogue systems, *Proceedings of ACL Annual Meeting*.
- Mann, W. C. and Thompson, S. A.: 1987, Rhetorical structure theory: a theory of text organization, *Technical Report ISI/RS-87-190*, Information Sciences Institute, University of Southern California.
- Matheson, C., Poesio, M. and Traum, D.: 2000, Modelling grounding and discourse obligations using update rules, *Proceedings of NAACL 2000*, Seattle.
- McCarthy, J. F. and Lehnert, W. G.: 1995, Using decision trees for coreference resolution, *IJCAI*, pp. 1050–1055.
- McGlashan, S.: 1996, Towards multimodal dialogue management, *Proceedings of Twente Workshop on Language Technology 11*, Enschede.

- Meng, H., Busayapongchai, S., Glass, J., Goddeau, D., Hetherington, L., Hurley, E., Pao, C., Polifroni, J., Seneff, S. and Zue, V.: 1996, WHEELS: A conversational system in the automobile classifieds domain, *Proc. ICSLP '96*, Philadelphia, PA.
- Müller, C., Rapp, S. and Strube, M.: 2002, Applying co-training to reference resolution, *ACL Proceedings*.
- Neale, S.: 1990, *Descriptions*, MIT Press.
- Pollard, C. and Sag, I.: 1994, *Head-driven Phrase Structure Grammar*, University of Chicago Press, CSLI Publications, Chicago.
- Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P. and Zue, V.: 1998, Galaxy-ii: A reference architecture for conversational system development, *Proc. ICSLP '98*, Sydney.
- Souvignier, B., Kellner, A., Rueber, B., Schramm, H. and Seide, F.: 2000, The thoughtful elephant: Strategies for spoken dialog systems, *IEEE Transactions on Speech and Audio Processing* 8(1), p51.
- Stalnaker, R.: 1974, *Pragmatic Presuppositions*, New York University Press, pp. 129–213.
- Traum, D. R. and Allen, J. F.: 1994, Discourse obligations in dialogue processing, in J. Pustejovsky (ed.), *Proceedings of the Thirty-Second Meeting of the Association for Computational Linguistics*, Morgan Kaufmann, San Francisco, pp. 1–8.  
\*citeseer.nj.nec.com/traum94discourse.html
- TRINDI: 2001, The Trindi Book, <http://www.ling.gu.se/projekt/trindi/>, Gothenburg, Sweden. notes from the ESSL-2001.
- van der Sandt, R.: 1992, Presupposition projection as anaphora resolution, *Journal of Semantics* 9, 223–267.
- van Noord, G., Bouma, G., Koeling, R. and Nederhof, M.-J.: 1998, Robust grammatical analysis for spoken dialogue systems, *Natural Language Engineering* 1(1), 1–48.
- van Rooy, R.: 2001, Exhaustivity in dynamic semantics; referential and descriptive pronouns, *Linguistics and Philosophy* 24(5), 621–657.
- Wahlster, W. (ed.): 2000, *Verbmobil: Foundations of Speech-to-Speech Translation*, Springer Verlag, Berlin-Heidelberg.
- Wahlster, W., Reithinger, N. and Blocher, A.: 2001, Smartkom: Multimodal communication with a life-like character, *Proc. of Eurospeech 2001*, Aalborg.
- Webber, B.: 1991, Structure and ostension in the interpretation of discourse deixis, *Natural Language and Cognitive Processes* 6(2), 107–135.