Abstract

In this thesis the methods implemented to resolve anaphora in the speech recognition environment of the SPICE-EPG demonstration prototype, an electronic programming guide, of Philips Research are described. The SPICE-EPG uses shallow-parsing, which provides no information about sentence structure and only relevant phrases are returned. To resolve anaphora, syntactic information is very important, and without it anaphora resolution becomes very difficult. To overcome the lack of syntactic information a reference resolution model is used, which determines the preference for referents without needing syntactic information and a set of filters is applied to be able to determine some of the dependencies between different phrases, which are needed to successfully solve anaphora.
Three different ways to determine the dependencies between the phrases are employed: looking at the properties of the different phrases and determine the dependency based on the match with these properties, assigning a subphrase to a phrase which indicates the dependency, and assign a superphrase to a phrase which indicates the dependency. The first method is applied when two different phrases do not necessarily appear next to each other, but other unrelated phrases can occur between them. The second method is suitable when the two phrases always occur next to each other, and one of them provides extra information about the other. The third method is employed when a so called compound reference occurs: a phrase refers to a property of another phrase, which is a reference itself.
This group of methods is tested on a small corpus, which is based on examples of reference given by co-workers, based on their ideas about the type of references which the electronic programming guide should ideally be able to handle. Offline tests show that the chosen method is adequate in resolving references which fall within the scope of the project. Online tests however show that additional measures must be taken to solve certain problems with speech recognition errors.

Preface

The past twelve months I have been engaged in my diploma thesis for the faculty of Information Technology Systems at the Delft University of Technology in the specialization of Knowledge Based Systems. Starting from September 2000 until August 2001 I have done my diploma work at the Philips GmbH Forschungslaboratorien in Aachen, who were so graceful to provide me the opportunity to have my internship there, and support me continuously with advising employees, working space and equipment. This paper is written as the final report on my work on References in a Multi-modal User Interface, during my period in Aachen.

First I would like to thank Petra Philips, who was my supervisor at Philips GmbH Forschungslaboratorien in Aachen. Even though she was very busy with her own project, she always found time to help me.
Second I would like to thank my professor, drs. dr. L.J.M. Rothkrantz, for helping me to find a place to do my diploma thesis, supervising my project and supplying helpful hints and ideas for the project.
Further I would like to thank the other co-workers at Philips for being so welcome, and providing a helpful hand whenever I needed it. Especially Andreas Kellner and Thomas Portele for their hard work in helping me preparing the system for integration with my module, and their useful tips for my reports.

And last but not least, I would like to thank my parents and sister for their mental, physical, financial, material and overall support they have given me.

Summary of table of contents

Table of Contents

List of Tables

List of Figures

Chapter 1.

Introduction

Every serious company developing machines, user hardware, software, household appliances, or any other technical product used by humans should include usability considerations in their design process. A well designed man-machine interface can prevent much frustration when the user is trying to work with some apparatus, whether it is a simple thing like a payphone or a complicated thing like a new computer program: Which button should I press now? Why does it beep now? Why does it do something else than I want it to?
A badly designed user interface may cause users to have trouble when using the machine, and it will take a long time before the user understands the behavior of the apparatus, increasing the time before the machine can be adequately used. It may even be the cause of severe injury like Repetitive Strain Injury (RSI), which is believed to be the result of intensive use of the interface under stressful situations for a longer period of time.
If a user interface is easy to understand and ergonomically designed, personnel can easily and quickly adapt to a new user environment, so training time and costs can be reduced when a new system is introduced. It may even cut in medical costs since Repetitive Strain Injury is prevented.

Nowadays multi-modal man-machine interfaces are a very interesting research topic, because they allow a user to intuitively communicate with a machine in a for the user optimal and natural way [Coh98]. Different tasks can be performed using different modalities, allowing the user to pick the most suitable modality for specific tasks or subtasks.
Speech is an important aspect of multi-modality, because it is one of the primary ways how humans pass on information. It is especially suitable for complex tasks where otherwise many actions must be performed, before the task can be completed. With speech, this can usually be done in one or two sentences.
On the other hand, some simple tasks which can be done with one point of the finger are a tedious process when the speech modality is used, because extensive description of the required task is needed. The Man-machine Interface group of Philips Research in Aachen has built an Electronic Programming Guide (EPG) with a multi-modal interface for television program recording within the Speech Interface for Consumer Electronics project (SPICE). This system supports both speech as well as pointing input to accommodate user needs when operating the system.
References occur very often in natural language use, because of the so called general Conservation Principle, which states that hearers do not like to make new discourse entities when old ones will do and that speakers try to form their utterances so that the hearer can make maximal use of old entities [Pri81]. This report will describe the reference resolution module which has recently been added to the SPICE-EPG system, to further ease the user's tasks, when he or she is trying to record a program. The literature study on reference handling is mainly based on reference handling for texts, because most

of the literature available covers this topic. With speech recognition, the speech is transcribed into text, before it is further processed, so in principle text based reference handling can be applied in a speech understanding environment. However not all data which is available for text processing is available, because the methods used for text are not robust enough for speech processing.

Chapter 1 gives a general introduction about the environment of the project. In section 1.1 the problem is defined. Section 1.2 gives an introduction about the SPICE-EPG: why the SPICE-EPG is developed, what its features are, and what the structure of the dialogue model is, and where the reference resolution module is placed in this model. In section 1.3 is discussed in which form references occur and what they can refer to in general. Finally section 1.4 describes how the performance is measured for reference resolution models.
Chapter 2 gives a short overview of the state of the art in reference resolution. In section 2.1 different grammars suitable for reference resolution is described. This description is meant to give a general idea what the grammar is about and how it would help in anaphora resolution. It is not a description how each grammar work and how the different structures are constructed with the grammars and what operations can be performed on these structures. Section 2.2 gives an overview of anaphora resolution models. An introduction to ellipsis resolution is also given in this section, even though it is strictly not a form of anaphora. The introduction is given because ellipsis is an important and often occuring form of reference. It may be desirable to solve ellipsis for the SPICE-EPG, but chapter 3 will show that it falls outside the scope of the project. This is also the reason that no more than a short introduction is given on ellipsis resolution.
Chapter 3 describes the anaphora resolution module in the SPICE-EPG. In section 3.1 the requirements for the module are stated, which are narrrowed down in section 3.2 to fit within the time and environmental constraints of the project. Section 3.3 describes which reference resolution model was choosen in the first stage of the project and why (see also appendix I for the literature survey from the first stage of the project). Section 3.4 describes what information is needed from the grammar for reference resolution, and how the system should be adapted to provide this information. Section 3.5 gives a general outline of the algorithm, followed by the system design in section 3.6.
Chapter 4 describes the evaluation of the reference resolution module, what works and what went wrong and in Chapter 5 the conclusions of the project are stated. Finally in chapter 6 future recommendations are given about possible improvements.

1.1. The Problem Definition

In the first three months of this project a literature survey was done to find a suitable solution for reference resolution in a demonstration prototype of the SPICE-EPG (see appendix I). The most suitable model for reference resolution in this prototype was found to be the model proposed in [Str98], Never look back: An alternative to Centering. Close examination of the examples given for the algorithm gave rise to the suspicion that the algorithm was probably not implemented, but hand tested only, and the author assumed that the data for correct resolution are simply present. Requests for clarifactions on this by e-mail, were responded by vague answers and confirmation that the author assumes that the algorithms to provide the data are present. Also, the algorithm is tested only for written text, whereas reference resolution has to work in a speech recognition environment, where recognition is not 100% and grammar is much looser. In the second stage of this project, this algorithm has to be implemented and tested.
Another problem is that because of the grammar used for the SPICE-EPG environment, no binding constraints are present for references to items in the same sentence, also dependencies between words in the sentence are missing. For this also a solution must be found. So there is still a big gap between the theoretical model proposed in [Str98] to solve references in written text, and having the model work in a real speech processing environment, without the certainties of written text and the presence of all the data needed.
The goals of the second part of the project are as follows:

- Implement the proposed model for operation in a speech recognition environment.
- Test the proposed model in a speech recognition environment.
- Find a method to compensate for the lack of syntactic information in a shallow parsing environment.

1.2. The SPICE-EPG System

The SPICE-EPG system (Speech Interface for Consumer Electronics – Electronic Programming Guide) provides the application scenario in which the reference resolution module will be tested. It is a research prototype which was built to demonstrate the potential of conversational user interfaces for consumer electronic devices and is used to test and evaluate speech and language technology [Kel00].
In this section a short introduction to the SPICE-EPG will be given. The next subsection will explain why SPICE-EPG was developed, what the design goals were for the prototype, and what the features are. The last subsection will discuss the system architecture used for SPICE-EPG.

1.2.1. Motivation for the SPICE-EPG

An EPG is an application with which the user can browse through a database of TV programs, get additional information on specific TV programs, switch to a program that is currently running or schedule it for later viewing or recording.
The entries from the database are usually accessed through channel, date, time or category, by selecting the appropriate function on the TV's remote control. The matches retrieved for the current selection are displayed on the screen. The remote control is then used to select one of the programs.
A review of EPG systems in the magazine 'Sound&Vision' [SVM99] shows that it is quite awkward to operate them with just a remote control. In this paper it is pointed out that it takes between 8 and 48 button presses with today's EPG systems just to find out what is on a certain channel on a certain day at a certain time. Since the EPG is already a very complex application compared to, for example, controlling a TV-application, as far as the user-system interaction is concerned, it is deemed suitable as a carrier application within the SPICE-EPG project to test and evaluate speech and language technology.
Another consideration was the fact that the EPG is a very well understood application within Philips Research. It has been a testbed for many new technologies. The application is also relevant to Philips Consumer Electronics, since an electronic programming guide will probably become part of most TV sets in the future.

1.2.2. SPICE-EPG Design Goals

The SPICE-EPG is designed to be a Mixed Initiative System, which means that the user is able to decide what information is given at what time in which way, and that both the user and the system may control the dialogue flow. This in contrast to simple Interactive Voice Response, where the user is prompted for a sequence of specific information items in a purely system directed way and only to a limited set of command words. These command words are basically just a replacement of buttons on a remote control. The main features of conversational user interfaces are:

- Spontaneous speech input
- Direct access to content
- User driven interaction

- Cooperative dialogue

These features will be discussed in the following subsections.

1.2.2.1. Spontaneous speech input

With natural language input, the user is able to formulate his request or command in his own words and does not have to use a specific pre-defined keyword. In the simplest case, this means that the user can choose between a large number of alternative ways to express a command. Beyond that, he is able to use more complex formulations and give a number of information items in a single utterance.
For example the user can say: “record the six o’clock program on channel 4,” instead of: “channel,” wait for system to display choice of channels, “channel 4,” wait for system to display the list of programs on channel 4, “time,” wait for system to display choice of times, “six o’clock,” wait for system to show the six o’clock program on channel 4, “record.”

1.2.2.2. Direct access to content

In most traditional user interfaces, the user selects a specific content item (e.g. a movie title) by navigating through a selection displayed on the screen. For example: select time slot, browse through the list of times, select channel slot, browse through list of channels, select title slot, browse through list of programs, select the appropriate title, and then access the title.
Using its large-vocabulary speech recognition capabilities, a conversational interface even allows the user to directly access the complete content at any time in the interaction, even if this specific item is not displayed on the screen. Using natural language input and information retrieval, it is also possible to refer to a title or the description of a program directly even if the formulation used does not exactly match the database entry.
So it is possible to tell the system to “record the James Bond movie tonight”, without having the title displayed on the screen first. The system would recognize that the movie “James Bond 007: Golden Eye” has to be recorded.
Especially if the number of available choices is very large, this is a major advantage and allows for much faster and more intuitive access.

1.2.2.3. User-driven interaction

In most of the current interfaces to consumer electronics devices, the user has to follow a hierarchy of menus and sub-menus to accomplish a complex task. The correct top-level menu is often not very obvious and therefore the user either has to remember all the steps of a command or try out various menus and submenus to find the right one. In a conversational user interface, the user does not have to follow the structure of a pre-defined control menu in order to complete a task. He can directly access functions at any level in the control hierarchy and give the information items in an arbitrary order.
In cases where the user’s input is not sufficient to identify a specific command and its arguments, the system will ask additional questions in order to obtain the missing information.

1.2.2.4. Cooperative dialogue

In conversational user interfaces, the interaction between the user and the system becomes a two-way communication. While the user is in full control of the dialogue and tells the device what to do, the device can also take the initiative and 'talk' back to the user. This can be used to give the user feedback on the system's current state or to verify some of the user's commands.
Furthermore, the system can actively guide the user through a complex task or offer some suggestions for content-selection based on the user's preferences.

1.2.3. Features of SPICE-EPG

The SPICE-EPG allows input from two different modalities. It is possible to do all tasks hands-free with spoken input only, or control the device with touch screen input in addition to speech. The current prototype output consists mainly of visual feedback for displaying information (e.g. a list of program items matching the user's selection) and spoken output to guide the user through the dialogue. In addition, an anthropomorphic cartoon character gives visual feedback on the current system status. Figure 1 shows the graphical interface of the SPICE-EPG.

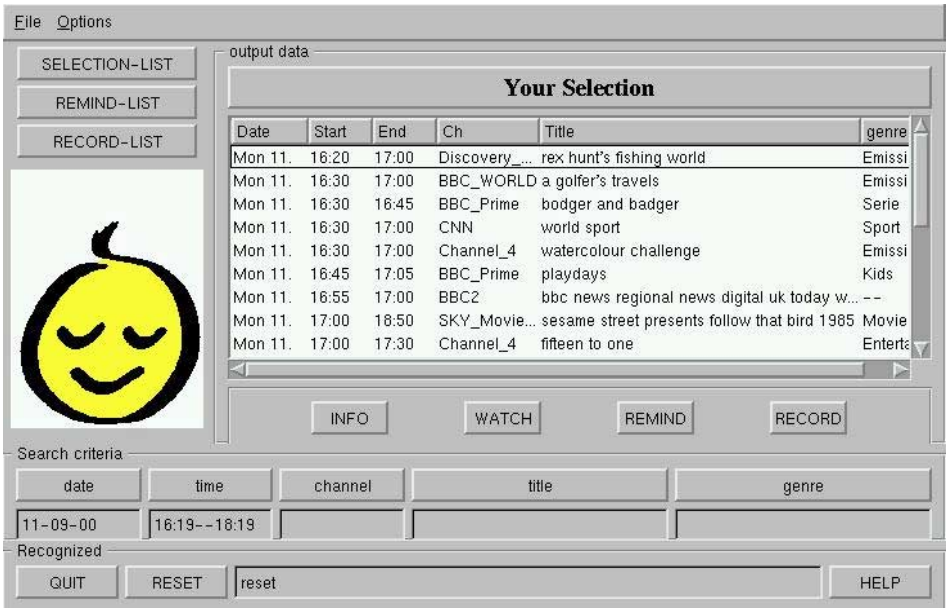


Figure 1: The SPICE-EPG Graphical Interface.

In the SPICE-EPG prototype, an offline copy of three weeks of program data with 7110 entries downloaded EuroTV (http://www.eurotv.com) is used. The user can select a set of programs from the database by specifying one or more of the following items in one utterance:

- Date
- Time
- Genre
- Channel
- Title
- Description

1.2.4. The SPICE-EPG Architecture

The dialogue model used for the SPICE-EPG used by Philips Research in a man-machine speech interface is shown in figure 2 [Kel00]. The different components of the model are discussed in the following subsections in order of the flow through the system.

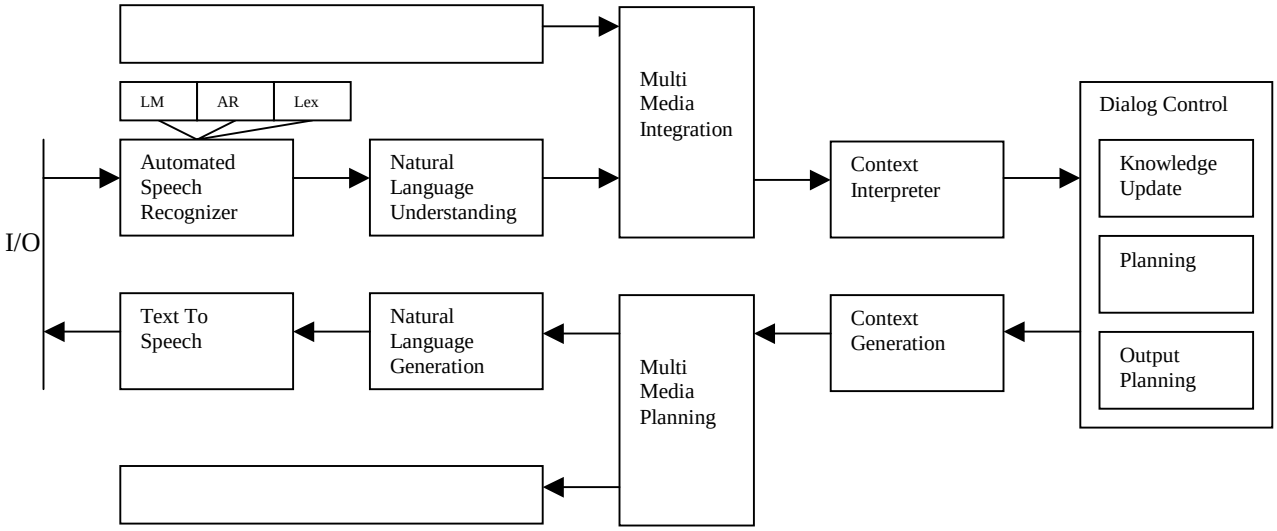


Figure 2. Dialogue system model used by Philips Research.
LM = Language Model, AR = Acoustic Reference, Lex = Lexicon.

1.2.4.1. The Automated Speech Recognizer.

The Automated Speech Recognizer (ASR) analyses the acoustic waveforms, and recognizes the word sequence spoken by the user. In order to do this, the ASR makes use of a Lexicon, an Acoustic Reference model, and a Language model. In the Lexicon the phonetic transcription of every word in the systems vocabulary is defined (similar to the pronunciation of a word in a dictionary). These phonetic transcriptions can be matched to (parts of) the acoustic waveform with the Acoustic Reference model, which calculates the likelihood that a signal refers to a particular phonetic unit. The Language Model is used to determines the a-priori likelihood of a word sequence, based on a text corpus that reflects the statistics of the application data.

1.2.4.2. The Natural Language Understanding Module

The Natural Language Understanding module interprets the input sentences. This means, it derives all the semantic information that is relevant in the given application. In some cases this analysis is done together with parsing (which is finding the syntactic structure of a sentence). While not strictly correct, both functionalities will be grouped in this text under the term parsing, for ease of explanation of the different grammars, this is also because in many papers no distinction is made and parsing is also used for derivation of semantic information.

Depending on the parser the depth of the derived information ranges from a dependency tree, where all the relations between the words are identified (see figure 3.) [Str95], to an overview of only grammatical relations like subject, object, etc. [Ken99] (see figure 4.), to syntactic structures like verbs, noun phrases, pronouns, etc. [All95] (see figure 5.) to only the meaningful parts without grammar base [Kel00] (see figure 6.). Most of these approaches are based on complex linguistic grammars, and achieve great performance at parsing large bodies of text, but are not robust enough to be used in spoken language dialogue systems. The main reasons for that are:

- Most of the input sentences are not structured correctly according to textual grammar rules.
- The speech recognizer introduces additional errors which lead to parsing problems.

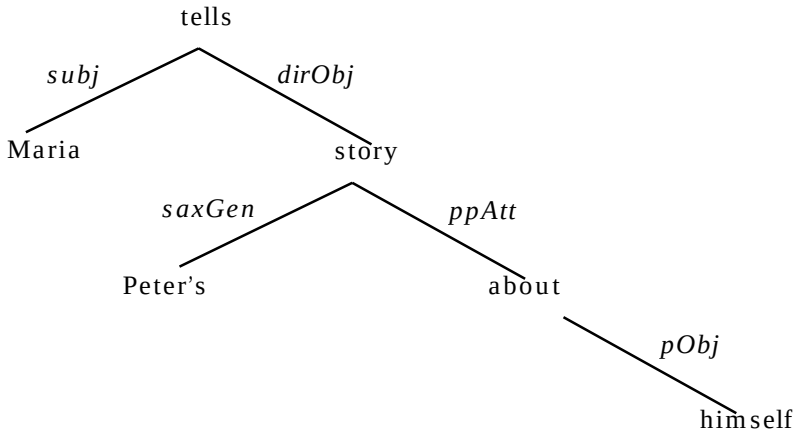


Figure 3. Dependency tree for: Maria tells Peter's story about himself.

0			
1 Maria	Maria	subj:>2	**@SUBJ** N NOM SG
2 tells	tell	main:>0	**@+FMAINV** V PRES SG3
3 Peter's	Peter	dat:>2	**@I-OBJ** N GEN SG
4 story	story	obj:>2	**@OBJ** N NOM SG
5 about	about	ha:>2	**@ADVL** PREP
6 himself	he	pcomp:>5	**@<P** <Refl> PRON PERS MASC SG3

Figure 4. Overview of grammatical relationships for: 'Maria tells Peter's story about himself.'

It is enough for the system though to only extract those words or phrases that are meaningful with respect to the current task. The rules specified in the grammar do not have to cover the complete user utterance, but only the meaningful phrases in the utterance must be represented. These are called concepts. The concepts are extracted from the user utterance by a top-down chart parser that allows for island parsing. This means that the parser attempts to assign concept types to the largest group of the words first and progressively decreases the size of the group. These groups can be isolated parts of the utterance, so that meaningless phrases are ignored. The so-called filler model allows the handling of meaningless phrases, which cannot be assigned to a concept [Aus95] [MaS00] [Sou00].

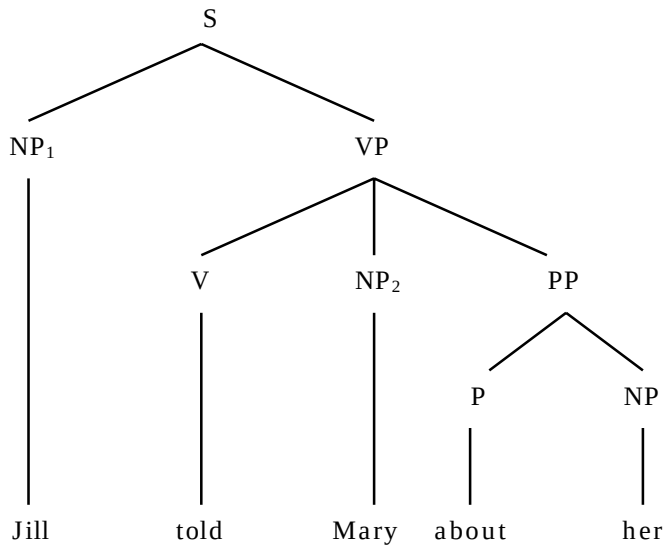


Figure 5. Syntactic tree for: Jill told Mary about her.

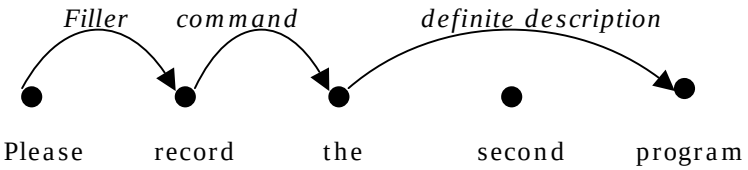


Figure 6. Meaningful concepts of 'Please record the second program'.

1.2.4.3. The Multimodal Integration Module

Here the parsed sentence is matched with the dialogue history and information from other modalities. The semantics represented in the speech input and in the pointing input are combined into a coherent semantic representation of the user input, so deixis is already taken care of. It allows for synchronous coordinated use of speech and pointing [Phi00].

1.2.4.4. The Context Interpretation module

References to previously mentioned topics or objects are resolved here.

1.2.4.5. The Dialogue Manager

The Dialogue Manager is the central module of the system. It maintains the system's internal knowledge stack (system belief), interacts with the actual application (e.g. TV, VCR, or EPG-database), and decides about the next action of the system.

1.2.4.6. The Media Planner

The Media Planning module decides how the information should by presented to the user and which media are suitable to do this. The abstract representation of the system's response is split into information to be presented in spoken and visual form.

1.2.4.7. The Language Generation Module

The Language Generation module translates the abstract representation of the system output which has to be spoken into a sequence of words.

1.2.4.8. The Text-to-Speech Module

The Text-to-Speech Module transforms the textual representation of this sequence of words into acoustic waveforms that are played to the user. In the SPICE-EPG prototype, the acoustic waveforms consist of pre-recorded spoken text.

1.3. An Introduction to References

Currently the Context Interpretation module of Philips lacks a method to solve references to previously mentioned topics or objects. Only references to objects which have been pointed to using the touch screen are resolved here. Since references very often abound in naturally occurring discourse, they are a critical part of natural language understanding. It is therefore important to be able to solve references in a user-friendly speech environment.
For example the user of the SPICE-EPG may want to use references like:

- *Record the first movie.*
- *Remind me of it.*
- *Record the one at ten p.m.*

1.3.1. References in Natural Language

In general the following types of references can be distinguished [All95]:

- **Anaphoric reference**: A reference to a previously mentioned entity. For example: *Mary bought a dress. It is very beautiful.*
  An anaphoric reference can be intrasentential (e.g. the referent is mentioned in the same sentence) or intersentential (e.g. the referent is mentioned in a different sentence).
- **Cataphoric reference**: A reference to a yet to be mentioned entity. For example: *These are our demands: We want three million helicopters and a dollar!... uhm I mean three million dollars and a helicopter....*
- **Deictic reference**: A reference to an entity from another modality. For example: *I want that* (with the speaker pointing to an apple).
- **Ellipsis**: A grammatically incomplete sentence, where part of a previous sentence grammatically completes this sentence. For example: *Sam forgot his wallet. Jack did too.*

Anaphoric, cataphoric and deictic references can be in the form of [All95]:

- **Pronouns**: *I, me, my, mine, you, your, yours, he, him, his, she, her, hers, it, its, we, our, ours, they, their, theirs, myself, yourself, himself, herself, itself, ourselves,* and *themselves.*

- **A zero pronoun** (not for cataphoric references): The referring pronoun is left out of the sentence, for example: *A judge ordered that Mr. Curtis be released, but ∈ agreed with the request from the prosecutors.* Here ∈ marks the spot where a pronoun, referring to the judge, should have been. Zero pronouns are not very common in English, but may occur often in languages like Spanish, Italian, Japanese, and Chinese [Fer00]. It can be argued that zero pronouns are some form of ellipsis (at least in English).
- **Demonstratives**: *this, that, these* and *those.*
- **Noun phrases modified with a definite article, a quantifying determiner, or a demonstrative** (definite descriptions): The word which forms the basis of the phrase is called the head, the words that provide extra information about the head, are called modifiers: *the dog, the mangy dog, the mangy dog at the pound, the four books, all books, some of the books, those books* etc.
- **The word *one*** (also called one anaphora. One anaphora is also not used in cataphoric cases), for example: *John had a blue shirt, Mary had a red one.*

Ellipsis takes the form of a grammatically incomplete sentence, where a subject, object, verb, or other grammatical function is missing: *My friend came by, and gave me a present.*

References may refer to [Bea99, Byr99, Eck99, Mcc96, Mur96, Pin00]:
- an entity that was introduced into the discourse via a noun phrase.
- a subset of a group that was introduced into the discourse via a noun phrase.
- a superset of individual entities that were introduced into the discourse via noun phrases.
- a general class of entities introduced into the discourse as a specific entity via a noun phrase.
- a property of an entity that was introduced into the discourse via a noun phrase.
- an event type.
- an action type.
- a property of an action.
- a fact or proposition.
- the general topic of the conversation.
- world/common knowledge not mentioned in the discourse.
- nothing at all.
- an entity from another modality.

In the next few subsections, the different types of references will be described in more detail.

1.3.1.1. Reference to an entity that was introduced into the discourse via a noun phrase.

The referent is introduced previously in the discourse via a noun phrase. Example:
*First we are going to take [both engines] from Elmira to Corning and then to Dansville. In Dansville **they** should pick up the three boxcars.*

In this example ***they*** refers to the previously introduced noun phrase *both engines*.

1.3.1.2. Reference to a subset of a group that was introduced into the discourse via a noun phrase

A group of entities as a whole may be introduced during the discourse and then later references can be made to a more specific (set of) individual entities of the previously introduced group. Example: *[A group of girls] went to Yorkshire by car. **The girl behind the wheel** was not paying attention to the road.*
In this example ***The girl behind the wheel*** is a reference to an individual from the previously introduced group *A group of girls*.

1.3.1.3. Reference to a superset of individual entities that were introduced into the discourse via noun phrases.

Sometimes individual entities are mentioned first in the discourse and later referred to as a group. Example:
*[Bill] paid [Bob] a visit. **The men** talked for a long time.*
In this example *Bill* and *Bob* were introduced first in the discourse and are referred to as a group in the next sentence by ***The men*.**

1.3.1.4. Reference to a general class of entities introduced into the discourse as a specific entity via a noun phrase.

Sometimes, a specific entity will be introduced into the discourse and then a subsequent reference will be to a more general class, of which the specific entity is a member. Strictly this is not considered an anaphoric reference, but in some applications (e.g. information extraction) it may have to be linked with the entity, since it may add important information about that entity. Example:
*[Familymart Co. of Seibu Saison group] will open a convenience store in Taipei Friday in a jointventure with Taiwan's largest car dealer. This will be the first overseas store to be run by **a Japanese convenience chain store operator**.*
The identifier **a Japanese convenience chain store operator** is a rather general reference to a class of entities. However, a system, which for instance is interested in nationality information of organizations, may need to be able to link this noun phrase to Familymart Co. of Seibu Saison group.

1.3.1.5. Reference to a property of an entity that was introduced into the discourse via a noun phrase.

The referent of a definite noun phrase is a property of a previously mentioned entity. Example:
*I went to [an old house] yesterday. **The roof** was leaking badly and…*
In this example ***The roof*** refers to the roof of *an old house*. Another example:
*…in [the Soviet Union], **they** spent more money on military power than anything.*
In this example ***they*** refers to the government of *the Soviet Union*.

1.3.1.6. Reference to an event type.

The referent may be an event mentioned in the discourse. Example:
*Oh, let me just check that we do not have [two trains trying to cross each other on the same track], but I do not think **that** is happening.*
In this example ***that*** refers to the event of *two trains trying to cross each other on the same track.*

1.3.1.7. Reference to an action type.

The referent is an action mentioned in the discourse. Example:
*How long does it take to [convert the oranges into orange juice]? **It** takes one hour.*
It refers to the action type *convert the oranges into orange juice.*

1.3.1.8. Reference to a property of an action.

The referent may be a property of a mentioned action in the discourse. Example:
*So that will take two hours to [get to Corning] an hour to [load the oranges] and two hours to [get to Bath]. So **that** will be another five hours.*
In this example ***that*** refers to the time required to perform the action *get to Corning, load the oranges and get to Bath.*

1.3.1.9. Reference to a fact or proposition.

The referent may be a fact or proposition. This is usually a whole sentence. The difference between fact and proposition is that a fact is true, and a proposition may be true. Example:
We need to pick up the boxcar of bananas in Avon.
*Okay, um [there are boxcars that are closer to Avon], if **that** helps any.*
***It** does not really matter, but…*
In this example both ***It*** and ***that*** refer to the whole sentence *there are boxcars that are closer to Avon.*

1.3.1.10. Reference to the general topic of the conversation.

Also called 'Vague Anaphors', the referent is not a clearly defined linguistic antecedent, but the general discourse topic. Example:
I mean, the baby is like seventeen months and she just screams. Well even if she knows that they are fixing to get ready to go over there. They are not even there yet – you know…
*Yeah. **It** is hard.*

1.3.1.11. Reference to world/common knowledge not mentioned in the discourse.

Usually a definite noun phrase refers to an entity mentioned previously in the discourse. Sometimes, though, a definite noun phrase is unique in the context and refers to some world knowledge instead of something mentioned previously. Example:
*Yesterday a man was busted by **the FBI**.*

In this example ***the FBI*** refers to a commonly known institute, namely the Federal Bureau of Investigation. It does not have to be introduced prior usage, because it is known what is meant with it.

1.3.1.12. Reference to nothing at all.

Occasionally pronouns do not refer to anything at all. These are called expletives. Example:
It *is hard to realize, that there are places that are just so, bare on the shelves as there.*

1.3.1.13. Reference to an entity from another modality.

In an multi-modal user interface a user may point to an object and refer verbally to that object. Example:
I want ***that*** *(*while pointing to an orange*)*!
Naturally ***that*** refers to the orange.

1.4. The Evaluation of Performance

Once recognized, a reference has to be resolved. Ideally a reference resolution algorithm would correctly find every reference and resolve it to its referent. Unfortunately, this is not an ideal world, and any reference resolution algorithm that is designed for any large corpus of text is likely to make mistakes. Even human reference resolution is not flawless.
In general there are two approaches to evaluate the performance of an algorithm [Mcc96]. The simplest approach is *Accuracy*, a more elaborate approach is *Recall & Precision*.
Recall is defined as the fraction of reference relationships between entities in a text that are correctly found by a system.
Precision is defined as the fraction of reference relationships found by a system that are correct.
For example consider figure 7: In the text there are six entities with referential properties: *A, B, C, D, E* and *F*. *B* refers to *A*, *D* refers to *C* and *F* refers to *E*. Suppose an algorithm finds the following matches: *B* refers to *A*, *D* refers to *A* and *F* refers to *E*. Then this algorithm has a recall of 67% (2 out of 3 reference relationships between entities in the text are correctly resolved) and a precision of 67% (2 out of 3 reference relationships found are correct).
Accuracy is simply defined as the percentage of correctly resolved references, and does not distinguish between not finding an existing reference relationship and finding a non-existing reference relationship. So 12 out of 15 combinations are correctly classified as referring or not-referring, so the algorithm has an accuracy of 87%.

<A-B>	<A-C>	<A-D>	<A-E>	<A-F>
	<B-C>	<B-D>	<B-E>	<B-F>
		<**C-D**>	<C-E>	<C-F>
			<D-E>	<D-F>
				<**E-F**>

Figure 7. Instances representing six relevant references to entities

Given a particular level of accuracy, the results of different reference resolution algorithms can vary widely: Suppose there are two different algorithms. One is very conservative, rarely matching a reference to a referent. The other is more liberal, matching references to wrong referents more easily. They both can have the same accuracy, but the first algorithm will have a low recall and high precision, whereas the second a high recall and low precision. The relative importance between recall and precision is still an open question though, but it does give an indication of where the algorithm fails.
Because different authors use different corpuses and evaluation criteria, comparison of the different algorithms based on these values only is difficult.

Chapter 2.

State of the Art in Anaphora Resolution

Currently the most popular research topics on references are pronominal anaphora and ellipsis. Definite descriptions is a less frequent topic of research. Very few is written on demonstrative anaphora and deixis, and even less is written on cataphoric references. Three different disciplines are interested in researching reference resolution methods:

- Computer linguist: in need of deeper understanding of language structure in dialogues and/or written text.
- Information Extraction: in need to resolve references to retrieve interesting information for their database.
- Man-Machine Interface: in need to understand the user using natural language.

Even though different disciplines may take more interest in different types of references, their theories may be interesting for the different areas with some modifications. So theories presented in papers by computer linguists may be used for man-machine interfaces. It should be noted though, that computer linguist and information extraction application do not require real-time reference resolution, whereas with man-machine interfaces real-time resolution is crucial. Also each disciplines usually focus on different types of applications, so the references encountered may differ.
In this chapter a short overview will be given of the grammars and algorithms used for anaphora resolution. These methods are mainly for reference handling in texts and not speech, but because speech is transcribed into text before it is processed further, the models would be in principle usable for reference handling in speech.
Also a short introduction is given on the general method of how ellipsis is resolved. According to linguists ellipsis is not really a form of anaphora (although it is also known as sentence anaphora) and in section 3.2.1 will be discussed that ellipsis falls outside the scope of the project, but because in appendix A where examples are given of references which ideally should be solved, according to the co-workers at Philips, contain ellipsis, a short introduction is given on the general method used to solve ellipsis.

2.1. Suitable Grammars for Anaphora Resolution

In order to be able to resolve references, information is needed about the role the reference has in the sentence and how other words in the sentence relate to the reference. A grammar model should provide the possibility to do so. There are several grammar models suitable for reference resolution. These are Government and Binding, Discourse Representation theory, and ParseTalk. In addition to these grammars, a method using a tagger is also discussed, because it allows to analyze a sentence without the use of complex parsers. The description of the grammars is meant to give a general idea what the grammar is about and how it would help in anaphora resolution. It is by no means meant to describe how the grammars work and how to construct the different structures with the grammars.

2.1.1. Government and Binding

One of the most sophisticated approaches for treating anaphora at the sentence level of description is Government and Binding Theory, which was mainly developed by Chomsky [Cho81]. Government and Binding Theory (GB) assumes that a large portion of the grammar of any particular language is common to all languages, and is therefore part of Universal Grammar. The GB view is that Universal Grammar can be broken down into two main components: levels of representation and a system of constraints.
In the level of representation it is defined which words can be grouped into meaningful groups (phrases), what the syntax of these groups are, and how these groups are structured into a sentence.
For example, for the noun *interpretation* the following syntax can be defined to group it into a meaningful phrase:

- *interpretation, N, [_(PP$_{[of]}$)]*

Which means, that *interpretation* is a noun (*N*), which can be complemented with a proposition phrase using the proposition *of* (*PP$_{[of]}$*). The underscore denotes the position of the noun *interpretation* in the group.
Figure 8. shows an example of how the different phrases are structured in the sentence: *The musician's interpretation of that sonata*.

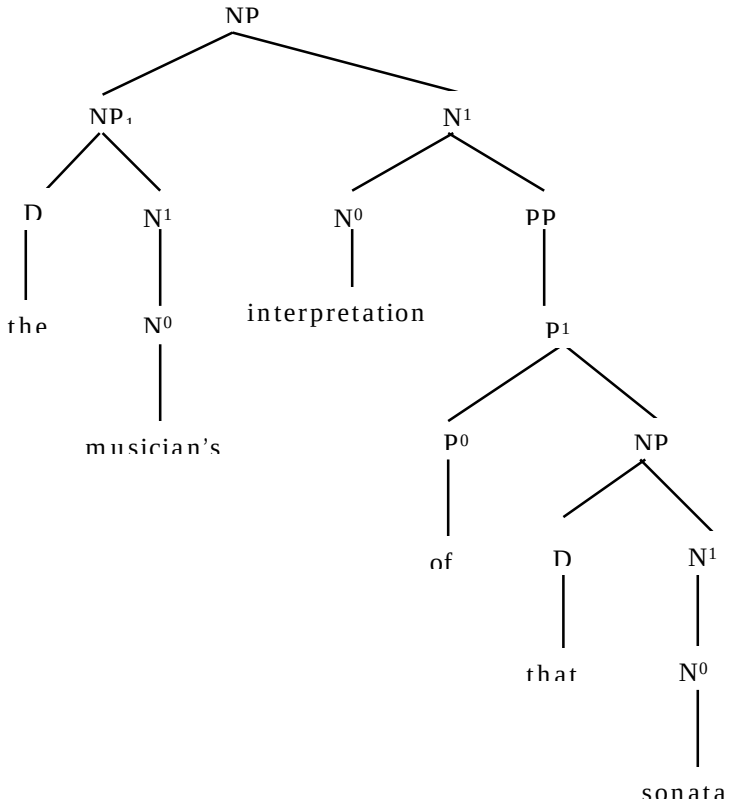


Figure 8. Syntactic tree of: 'the musician's interpretation of that sonata'

The system of constraints consists of binding constraints which define the scope of noun phrases in an intrasentential context. For example, consider figure x. In this figure a syntactic tree is shown for the sentence: *Jill read Mary's book about her*. In order to

resolve the reference *her* to the correct referent, the system of constraints defined in GB can be used. The scope of the referents are based on their relative position in the syntactic tree.

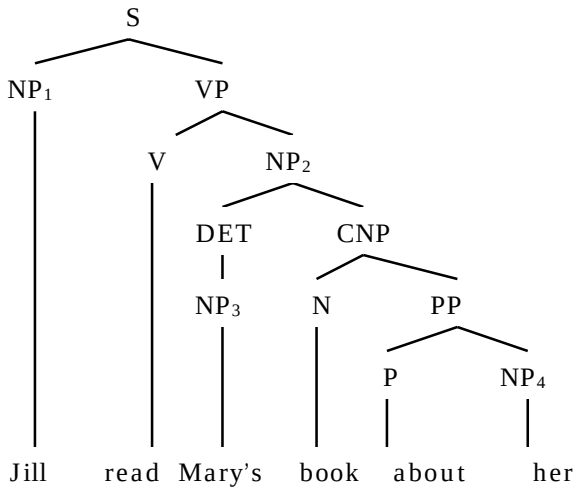


Figure 9. Syntactic tree of the sentence: Jill read Mary's book about her.

2.1.1.1. Co-reference constraints in Government and Binding.

1. A reflexive pronoun must refer to a noun phrase (NP) in the same domain with the following properties: The NP does not dominate the pronoun and the first branching node that dominates the pronoun must also dominate the NP. This property is called the C-commanding relationship.
   A domain of a node consists of the set of nodes in the tree, which are grouped under the closest S or NP. So NP_3 and NP_4 are in the same domain, but NP_1 and NP_3 not. The S or NP which defines the domain is said to dominate the nodes in that domain. (In figure 9, NP_1 does not dominate NP_2, NP_3, and NP_4, and the first branching node that dominates NP_1 is the S; thus NP_1 C-commands NP_2, NP_3, and NP_4. NP_2, on the other hand dominates NP_3, and NP_4, so it does not C-command them. The first branching node of NP_6 (NP_2) dominates NP_7, and as a result C-commands it). The domain of an item is defined as the set of items closest S or NP that contains it. So NP_3 and NP_4 are in the same domain.
   For NP_4 to co-refer with *Mary*, the pronoun would have to be reflexive, because according to this constraint a pronoun can only refer to a NP in the same domain which it C-commands if it is reflexive.
2. A non-reflexive pronoun cannot refer to a C-commanding NP within the same local domain. For example, the pronoun *her* in figure 9 can refer to *Jill* according to this constraint because, although NP_1 C-commands NP_4, it is not in NP_4's local domain.
3. A non-pronominal NP cannot co-refer with an NP that C-commands it. This constraint acounts for sentences like *He said Jack wants to leave.* Because *he* C-commands *Jack*, they cannot co-refer.
4. Two co-referential noun phrases must agree in number, gender and person.

2.1.2. Discourse Representation Theory

Another strong alternative for considering anaphora constitutes the framework of Discourse Representation Theory (DRT) [Kap81]. DRT was originally designed as a principled method to cope with two related problems: The fact that intersentential and intrasentential pronouns seem to call for two entirely different types of explanation, and a problem in connection with the interpretation of full noun phrases in certain types of sentences, like : *John does not own a donkey. It is gray.* and *Every boy invited a girl. Her name is Joan.* In these sentences *It* and *Her* cannot refer to *a donkey* and *a girl* respectively, because they contradict with the information given in the sentence. To solve this problem DRT defines some accessibility restrictions to so called 'Discourse Representation Structures' (DRS), which are objects which represent the information provided in a sentence. The scope of DRS's define the scope of possible referents. In figure 10. an example is given of how this can be done. In general it consists of a 'discourse referent' (u1 – u6), which is basically a marker representing an object which has been introduced in the discourse, and a 'condition' over this discourse referent. *It* (u6) cannot refer to 'book', because discourse referents cannot access other discourse referents from outside, unless if both are in the parts of the same conditional DRSs. So u5 can refer to u4. Another restriction is that a negated discourse referent cannot be accessed from outside, so that the pronoun *It* in the sentence '*John does not own a donkey. It is gray,*' cannot refer to *a donkey.*

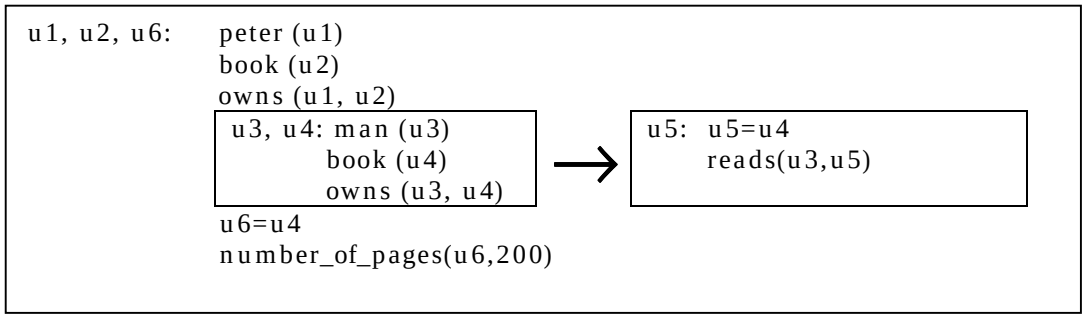


Figure 10. A DRS of the sentences '*Peter owns a book.', ' If a man owns a book he reads it.' & ' It has 200 pages.'*

2.1.3. ParseTalk

Another model to describe the role of words in a sentence is ParseTalk [Str95], which is a dependency-oriented grammar model. In [Str95] the authors claim that ParseTalk overcomes the problems that Government and Binding (GB) and Discourse Representation Theory (DRT) have. According to them, GB cannot handle some crucial linguistic phenomena, such as topicalization very well (*This picture, I never liked it*, is the topicalized form of *I never liked this picture*), without assuming very complex forms, and it is not very suitable for free word order languages, such as German.
DRT lacks a thorough treatment of complex syntactic constructions, and fails when various non-anaphoric text phenomena need to be interpreted. This is due to the fact that DRT is basically a semantic theory, not a comprehensive model for text understanding. It lacks any systematic connection to comprehensive reasoning systems concerning the

conceptual knowledge and specific problem solving models underlying the chosen domain.
The authors claim that the dependency-based grammar model underlying ParseTalk

1. covers intrasentential anaphora at the same level of descriptive adequacy as current GB, although it provides less complex representation structures than GB analysis,
2. does not exhibit an increasing level of structural complexity when faced which cause considerable problems for current GB theory,
3. goes beyond GB in that it allows the treatment of anaphora at the intersentential level of description within the same grammar formalism as is used for intrasentential anaphora, and,
4. goes beyond the anaphora-center treatment of text structure characteristic of the DRT approach in that it already accounts for the resolution of intrasentential ellipsis.

Like most grammars, the ParseTalk model of Dependency Grammar consists of a lexicon, a set of rules, which specify how words are grouped into meaningful phrases, and a set of constraints, which define the scope of possible referents.
The lexicon is ordered as a hierarchical tree, which defines the relationships between different words. Figure 11 shows an example of a part of the hierarchical tree of a lexicon. The hierarchical tree in the lexicon allows for resolution of references like: *My computer crashes quite often. I think the motherboard has the wrong chipset* and *Yesterday I bought a LPS 105, this harddisk has a really good perfomance!* With the lexicon *the motherboard* can be resolved to *the motherboard* of *my computer*, and *this harddisk* can be resolved to *LPS 105*.

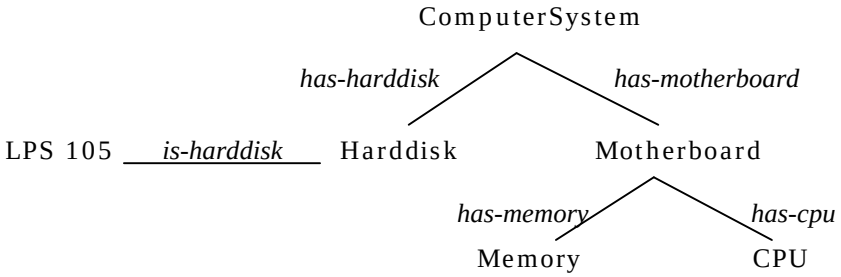


Figure 11. Part of a hierarchical tree of a lexicon in ParseTalk.

For each lexical item, rules are defined what dependencies it has with other items that can modify it in a sentence. For example: *tell* has a *subject* and a *direct object*. Using these rules, a sentence can be parsed into a dependency tree. Figure 12 shows a dependency tree for the sentence *Maria tells Peter's story about himself.*
Finally a set of binding constraints define the scope of possible referents based on the relative position in the tree.

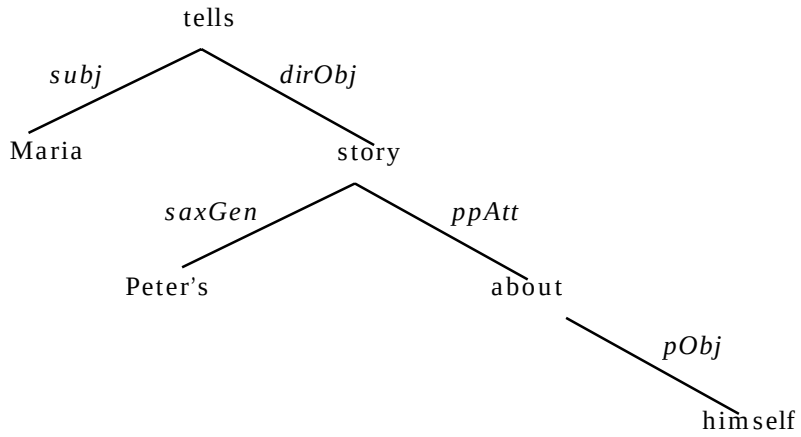


Figure 12. Dependency tree for: Maria tells Peter's story about himself.

2.1.3.1. Binding constraints in ParseTalk

Before the constraints can be discussed, the term *d-binding* must be introduced. A head is the the word which forms the basis of a phrase, and the set of phrases which complete the meaning of the head are colled modifiers, so in figure 12, *tells* is the head, and *Maria* and *Peter's story about himself* are the modifiers. It is said that the head governs its modifiers. A modifier M in the tree is d-bound by some head H, if no node N intervenes between M and H for which one of the following conditions holds:

1. node N represents a finite verb, or
2. node N represents a noun with a possessive modifier, i.e. possessive determiners, saxon genitive, genitival and prepositional attributes.

Based on the definition of d-binding, it is possible to specify several constraints on reflexive pronouns and anaphors in Dependency Grammar terms:

1. A reflexive pronoun and the antecedent to which the reflexive pronoun refers are d-bound by the same head. So according this constraint, in the example of figure 12 *himself* can refer to *Peter* because both are d-bound by the same head, and *himself* is reflexive.
2. The antecedent to which a pronominal or nominal anaphor refers may only be governed by the same head H_1 which d-binds the anaphor, if the antecedent is a modifier of head H_2, which is governed by H_1, and the antecedent precedes the anaphor in the linear sequence of text items. In the example *Whether Peter should go to Dublin, he could not decide, Peter* is governed by *decide*, which d-binds *he* and *Peter* is a modifier of *go* which in turn is governed by *decide*. But *Peter* precedes *he* in the linear sequence of text items, so *Peter* and *he* can co-refer. In the example *He could not decide whether Peter should go to Dublin, he* precedes *Peter* in the linear sequence, so this constraint is violated, meaning *he* and *Peter* cannot co-refer.

2.1.4. Tagger as substitute for parser.

According to [Ken96] current state-of-the-art parsing technology still falls short of robust and reliable delivery of syntactic analysis of real texts to the level of detail needed for most anaphora resolution algorithms. Because of this, the authors have developed a text processing framework which builds its capabilities entirely on the basis of a considerably shallower linguistic analysis of the input stream.
The base level linguistic analysis of the text processing framework is the output of a part of speech tagger, augmented with syntactic function annotations for each item. This kind of analysis is generated by the morphosyntactic tagging system described in [Kar95], and can be tested at http://www.conexor.fi/testing.html. The tagger provides a very simple analysis of the structure of the text: for each lexical item in each sentence, it provides a set of values which indicate the morphological, lexical, grammatical and syntactic features of the item in the context in which it appears. Figure 13 shows the output from the tagger of: *Maria tells Peter's story about himself*. In the first column the offset is listed, in the second the actual words are listed, the third column lists the basic form of those words, the fourth column lists the linguistic representation (subj= subject, main = main element, dat = indirect object, obj = object, ha = heuristic high attachment, pcomp= prepositional complement), and the last column lists the functional tags (@SUBJ = subject, @+FMAINV = finite main predictor, @I-OBJ = indirect object, @OBJ = object, @ADVL = adverbial, @<P = other prepositional complement) with information about the type and form of the words (i.e. noun, verb, nominal, genitive, reflective, etc.).
The tagger used achieves 99.77% overall recall and 95.54% overall precision, over a variety of text genres, meeting the requirement to develop a robust text processor.

```
0
1 Maria     Maria    subj:>2     @SUBJ N NOM SG
2 tells     tell     main:>0     @+FMAINV V PRES SG3
3 Peter's   Peter    dat:>2      @I-OBJ N GEN SG
4 story     story    obj:>2      @OBJ N NOM SG
5 about     about    ha:>2       @ADVL PREP
6 himself   he       pcomp:>5    @<P <Refl> PRON PERS MASC SG3
```

Figure 13. Output from the tagger of: 'Maria tells Peter's story about himself.'

After the text is tagged, the text is run through a set of filters to acquire information about sentence structure and phrasal units.
The first filter identifies noun phrases, using a grammar which contains pattern characteristics about noun phrase composition. A second filter is used to detect nominal sequences in two subordinate syntactic environments: containment in an adverbial adjunct and containment in an NP. Containment means that there is a phrase or object within a phrase. For example see figure 9, where NP_2 is contained in VP. Finally a third filter identifies and tags occurrences of expletive *it*. These are occurrences of *it* where no specific referents are present.
Because the tagger does not generate any configurational information, the binding constraints are based on inferences from grammatical function and precedence. The authors show that in practice these constraints are extremely accurate. Their reference resolution algorithm achieves a 75% accuracy rate using this text processing framework,

whereas the original algorithm using a parser achieves a ratio of 85%. Of the 75 misinterpreted pronouns, only a few could be traced to a failure to correctly identify the syntactic context in which the referent appeared.

2.1.4.1. Binding constraints using the tagger

Three conditions which are of particular relevance to anaphora resolution are defined, using the functional information provided by the tagger:

1. A pronoun which has the function of subject or direct object, cannot co-refer with a direct object, indirect object, or accusative item, which follow the pronoun, without an intervening subject (The hypothesis being that a pronoun cannot corefer with a coargument, and that a subject indicates the beginning of the next clause). For example, in *he gave him a hug*, the subject *he* cannot corefer with the direct object *him*.
2. A pronoun which is contained cannot refer to an object which precedes it, if there is no object in between with a containment value of nil. For example in *Jill read Mary's book about her*, the pronoun *her* cannot refer to *Mary*, because *Mary* precedes the pronoun *her* which is contained in *Mary's book about her*, and no object with a containment value of nil (*book* is also contained) is present between them.
3. Two co-referential noun phrases must agree in number, gender and person.

2.2. Anaphora Resolution Algorithms

In this section four different approaches to anaphora resolution are presented. These are: resolution based on the recency constraint (section 2.2.1), resolution based on the centering model (section 2.2.2), resolution based on given-new (section 2.2.3), resolution based on heuristics (section 2.2.4).

2.2.1. A simple model of anaphora resolution based on history lists

The most simple technique to resolve anaphora is with the use of simple history lists [All95]. This algorithm implements what is often called the recency constraint, which states that the antecedent should be the most recently mentioned object that satisfies all the constraints. This algorithm can often be used for definite descriptions as well as pronouns.

The history is a list of discourse entities generated by the preceding sentences, with the most recent listed first. The entities from the current local context are listed first, then the entities in local context generated by the sentence before that, and so on.

The possible antecedents for pronouns are not restricted to appearing in the local context, but the local context is very important for resolving pronominal reference. A large majority of antecedents for pronouns are found in the same sentence or in the local context. The further back in the discourse an antecedent was last mentioned, the less likely it is to be referred to again by a pronoun.

The history list consists of all the discourse entities that have been evoked in the reasonably recent past. Some systems allow just the last one or two local contexts, while

others let the history list grow unboundedly. Given the history list, the algorithm for finding an antecedent proceeds as follows: Check the most recent local context for an antecedent that matches all the constraints related to the pronoun. Constraints may come from any source. For example, reflexivity constraints will prohibit some objects from being the antecedent, gender and number will eliminate others. If no antecedent is found in the current local context, then move down the history list to the next most recent local context and search there.

2.2.2. The Centering Model

A more advanced and currently very popular algorithm is based on the notion of a 'discourse focus' or 'center'. The centering model is a refinement of Sidner's local focusing model [Sid83]. The intuition behind these theories is that most discourse is organized around an object that the discourse is about [All95]. This object, called the center, tends to remain the same for a few sentences and then shift to a new object. The second key intuition is that the center of a sentence is typically pronominalized. This affects the interpretation of pronouns because once a center is established, there will be a strong preference for subsequent pronouns to continue to refer to the center. For example:

a. Jack left for the party late.
b. When he arrived, Sam met him at the door.
c. He decided to leave early.

Semantically, sentence c of the example makes sense with either Jack or Sam as the antecedent, and the structural preferences favor Sam because he plays a central role in the major clause in sentence b of the example. Centering theory, however, would predict that Jack is the antecedent because Jack was referred to pronominally in sentence b and thus is the center of sentence b, and nothing in sentence c indicates that the center has changed.

2.2.2.1. Technical Details of the Centering Model

In centering theory two interacting structures are used [All95]:

- The discourse entities in the local context, which are called the 'potential next centers' (or forward-looking centers, C_f). These are listed in an order reflecting structural preferences: subject first, direct object next, indirect object, and then the other discourse entities in the sentence. The first one on the list is called the 'preferred next center' (C_p).
- The center, also called 'backward-looking center' (C_b), is what the current sentence is about. The backward-looking center is one of the potential next centers, and typically it is pronominalized.

The constraints between the center and pronominalization can be stated as follows:

- Centering Constraint 1: If any object in the local context is referred to by a pronoun in the current sentence, then the center of that sentence must also be pronominalized.
- Centering Constraint 2: The center must be the most preferred discourse entity in the local context that is referred to by a pronoun.

- Centering Constraint 3: Continuing with the same center from one sentence to the next is preferred over changing the center.

The last constraint can be specified more precisely in the following way:

- Centering Constraint 3': Continuing with the same center from one sentence to the next, which is the preferred next center (Continue), is preferred over continuing with the same center from one sentence to the next, which is not the preferred next center (Retain), Retaining over shifting to the preferred next center, and shifting to the preferred next center over shifting to the nonpreferred next center. Table 1 shows the types of movement for centers.

Table 1. The types of movement for centers.

	$Cb_2 = Cp_2$	$Cb_2 \neq Cp_2$
$Cb_1 = Cb_2$	Continuing	Retaining
$Cb_1 \neq Cb_2$	Shifting to preferred	Shifting to nonpreferred

Continue < Retain < Shift to Preferred < Shift to nonpreferred.
With < being the preference relationship.

In [Kam93] a variant on the centering model is presented, in which the transitions differ. This model, called the temporal centering model, was originally presented as a means to resolve anaphoric properties of past and present, using centering theory, but is also used for pronominal reference resolution in other papers [Pas89][Pas96].
The following four transition relation types for centering are described: Cb-retention, Cb-establishment, Cb-resumption, and the NULL transition.

- Cb-retention means that the same center is kept from one sentence to another. In this model no distinction is made between Continuing (continuing with the same center from one to the next, which is the preferred center) and Retaining (continuing with the same center from one to the next, which is not the preferred center).
- Cb-establishment means that another member of the forward-looking center becomes the current focus of attention. Again no distinction is made between shifting to preferred and shifting to nonpreferred.
- Cb-resumtion means that an old center (Cb) not in the list of forward-looking centers becomes the current focus of attention. This is one of the real differences between the model described in [All95] and the temporal centering model. In the 'normal' centering model, only centers in the list of forward-looking centers are candidates for the next focus of attention, centers outside the list are ignored.
- Cb-NULL means that in the new state, there is no center.

Temporal centering posits a default preference for retention over establishment. Establishment is preferred over resumption or NULL-transition.

Example:

a.	John went to the store.	Cf_1=[John', store1],	Cb_1=NULL	
b.	He saw Bill.	Cf_2=[John', Bill'],	Cb_2=John',	*Cb-establishment*
c.	He walked towards him.	Cf_3=[John', Bill'],	Cb_3=John',	*Cb-retention*
c'.	He appeared pale to him.	$Cf_{3'}$=[John', Bill'],	$Cb_{3'}$=Bill',	*Cb-establishment*

In the example the centering model is illustrated, with sentence c and sentence c' as alternative continuations of sentence b. After sentence a., the list of forward-looking centers contain two entities, John' and store1. In b., John' is referred to with a subject pronoun, and is established as the center. In c., because John' is the current Cb, and because retention is preferred over establishment, centering predicts that a subject pronoun will refer to John' rather than to Bill'. The default is overridden in c' and instead, the subject pronoun is inferred to refer to Bill' because it is likely that the perceiver in the first perceptual state, 'see', remains the perceiver in the subsequent perceptual state, 'appear'.
This model can be extended with additional constraints which define the behavior on centering of the pronoun *it* and the demonstrative *that* [Pas89]. These constraints are based on the notion that the grammatical role and form of the pronoun and demonstrative may indicate a preference to certain antecedents in certain grammatical roles and forms. Grammatical roles refer to subject and non-subject roles, and grammatical forms refer to canonical and non-canonical forms, meaning a single word or noun phrase and a clause like phrase respectively. The following constraints are defined.

- *It* indicates canonical or non-canonical center retention.
- *It* in subject role conflicts with non-subject antecedents, but is compatible with an NP-subject antecedent.
- *That* blocks canonical center retention.
- *That* may be more compatible with non-canonical center retention.
- *That* in subject role is most likely when the antecedent is not a noun phrase.
- *That* is enhanced when the antecedent is not a noun phrase.
- *That* is enhanced when the antecedent NP is a non-subject.

Noun phrase subjects have a relatively unspecified attentional status.
The algorithm to solve pronominal references will be as follows:

1. Generate a ranked list of possible antecedents for each pronoun
2. Use general reasoning to select the appropriate antecedents based on local discourse context, the co-reference restrictions, and the centering constraints.
3. Use the results of step 2 to define the C_b for the sentence to be used as part of the local context of the next sentence.

2.2.2.2. Interaction of Centering Preferences with Intrasentential Interpretations

It is still not entirely clear how centering preferences interact with the possibility of intrasentential interpretations, which are provided by certain grammars, like Government and Binding (see section 2.2.1). Determining what technique is best must await further development and evaluation of the possible algorithms. Currently, some algorithms always prefer intrasentential referents, while others favor the reverse. Another

combination is to prefer any interpretation that assigns a pronoun to the center, but failing that, to prefer intrasentential readings over intersentential readings [All95]. [Keh93] describe another possibility , where the following observations are made:

- Intersententially-referring pronouns have a strong bias towards their *preferred referent*, that is, the most highest-ranked entity in the forward-looking center, for which reference is not blocked by syntactic co-reference or agreement constraints.
- All pronouns have reflexive and non-reflexive forms (e.g., accusative = him, nominative = he, genitive = his).
- Non-reflexive pronouns *cannot* refer to a C-commanding NP.
- Reflexive forms *must* refer to a C-commanding NP.

In [Str96b] is stated that not only the grammatical roles must be considered when finding the preferred referent, but the functional information structure is crucial in finding it. The functional information structure has impact not only on the resolution of intersentential anaphora, but also on the resolution of intrasentential anaphora. Hence, the most preferred antecedent of an intrasentential anaphor is a phrase which is also anaphoric. To illustrate this, consider the following example:

If the resume mode is active, the T3100SX switches itself automatically of.
When the computer is turned on later, it resumes at exactly the same place.

In the second sentence *the computer* is resolved to *the T3100SX* from the previous sentence, and the pronoun *it* is resolved to the already resolved anaphor *the computer*.

2.2.2.3. Solutions for Centering Ambiguity

There is a situations where the centering model will come into trouble and will not be able to solve the situation correctly. This is the case where a reference is ambiguous to what it refers to, and choice of the wrong referent will cause strange behaviour of the algorithm. This can be illustrate by the following example:

As far as performance is concerned, the LPS 105 harddisk *also produced rather compelling results.*
Regarding the mean access time (16,5 ms) this hard disk *compares to* the Seagate ST-3144, *by which* it *scores second-best in this category.*
Also, considering data throughput it *turns out to be a high-caliber product.*

The first sentence has a unique structural analysis, the forward-looking centers consist of two semantic/conceptual elements, *the LPS 105 hard disk* and *performance*. In the second sentence, a nominal anaphor occurs, *this hard disk*, which is resolved to LPS 105 from the previous sentence. Unfortunately, the noun phrase *this hard disk* is nominative as well as accusative and may be alternatively attached to the verb *compares to* both in its subject and object role. In this state, one cannot determine which of the grammatical functions is the correct one, thus a structural ambiguity has been identified. Since the second NP in this sentence (*the Seagate ST-3144*) is ambiguous with respect to both of these cases, too, the parser produces two structurally and conceptually ambiguous readings. As a

consequence, two different forward-looking centers (Cfs) have been created, namely *LPS-105* and *Seagate ST-3144*, which indicate two different center transitions, eligible at the end of the analysis of the second sentence. This choice option becomes crucial for the resolution of the pronoun *it* in the third sentence, as it depends on the appropriate selection of one of the two different Cfs. Depending on how the text actually proceeds either one is equally possible. So, for the actual anaphora resolution the transition type preferences are of no help at all to decide among any of these variants. It is therefore concluded that additional representation devices have to be supplied to keep track of these structurally induced ambiguities at the center level [Hah96].
Therefore a two-level representation of structural ambiguities for the centering model is proposed, one at which local and global structural ambiguities are made explicit. Global ambiguities are represented as sets of forward-looking centers, while local ambiguities are represented as a set of such centering sets. When an ambiguity is encountered, a set is created for each possibility. For each set, the center is determined, and kept for the next sentence. If the new sentence contains information which indicates that the center of a set is incorrect, the set is discarded. Otherwise for each possibility the new center will be determined.

2.2.3. Never look back: An alternative to Centering

In [Str98] an alternative to centering is proposed, in which the functions of the *backward-looking center* and the *centering transitions* are replaced by the order among the elements of the list of salient discourse entities (S-list). This S-list ranking criteria is based on the observation from [Pri81] that there is a preference for *hearer-old* over *hearer-new* discourse entities. Hearer-old means that the entity is already in the knowledge model or the hearer, whereas hearer-new means that it is not. Because of these ranking criteria, the difference in salience between definite NPs (mostly hearer-old) and indefinite NPs (mostly hearer-new) can be accounted for. Table 2 shows how discourse entities can be categorized according to how new they are in the discourse and to the hearer.

Table 2. Discourse and hearer newness of discourse entities

	Hearer-old	Hearer-new
Discourse-old	Evoked (E) Situationally Evoked (E^S)	Inferrable (I) Containing Inferrable (I^C)
Discourse-new	Unused (U)	Brand-New Anchored (BN^A) Brand-New (BN)

Discourse-new entities can be of two types. In one case, the hearer creates a new entity, either of the form BRAND-NEW (BN) or BRAND-NEW ANCHORED (BN^A). A discourse entity is ANCHORED if the noun phrase representing it is linked by means of another noun phrase, or "anchor," to some other discourse entity. Thus *a bus* is UNANCHORED, and simply BRAND-NEW, whereas *a guy I work with*, containing the noun phrase *I*, is BRAND-NEW ANCHORED, since the discourse entity the hearer creates for this particular guy will be immediately linked to his discourse entity for the speaker. In the data, all anchored entities contain at least one anchor that is not itself BRAND-NEW. In the other case, the

hearer has a corresponding entity in his own model and simply has to place it in the discourse-model, these discourse entities are usual proper names and titles. This type is called UNUSED (U).
Discourse-old entities can also be of two types. Either the discourse entity is already in the discourse-model, in which case it is an EVOKED (E) or a SITUATIONALLY EVOKED (E^S) entity, or the discourse entity is not already in the discourse model, but can be inferred, via logical or plausible reasoning, from discourse entities already present in the model, in which case it is an INFERABLE (I) or a CONTAINING INFERABLE (I^C). A discourse entitiy is EVOKED if the entity is previously introduced into the discourse model via a noun phrase. It is SITUATIONALLY EVOKED it the entity entered the model through another modality. A discourse entity is INFERABLE if the speaker assumes the hearer can infer it, via logical or plausible reasoning, from discourse entities already in the discourse model, for example *The driver* may be inferable from *a bus*. A discourse entity is a CONTAINING INFERABLE if it can be inferred from a discourse entity which is a superset containing this entity, for example *one of these eggs* is a CONTAINING INFERABLE, as it is inferable from *these eggs*. With this definition of the hearer's attentional state, the following familiarity scale can be defined, where x > y indicates that an entity from x is preferred over an entity from y:

$$\{E, E^s\} > U > I > I^C > BN^A > BN$$

So the hearer is more likely to assign a referent to an evoked entity than a brand new entity. Based on this familiarity scale, three different sets of expressions are distinguished by [Str98]: *hearer-old discourse entities* (OLD), *mediated discourse entities* (MED) and *hearer-new discourse entities* (NEW). OLD consists of *evoked* and *unused* discourse entities, while NEW consists of *brand-new* discourse entities. MED consists of *inferables*, *containing inferables* and *anchored brand new* discourse entities. These discourse entities are *discourse-new* but *mediated* by some *hearer-old* discourse entity.

Table 3. Grouping of the different types of discourse entities.

OLD	MED	NEW
E, E^S, U	I, I^C, BN^A	BN

Anaphora resolution is performed with a simple look-up in the salience list, which is ranked as follows:

- An entity that is OLD precedes a MED entity.
- An entity that is OLD precedes a NEW entity.
- An entity that is MED precedes a NEW entity.
- If both entities are from the same attentional state, than the entity from the later utterance precedes the other entity [Ram93], [Val90], [Val96].
- If both entities are from the same attentional state, and the same utterance, than the entity which comes first precedes the other entity [Ram93], [Val90], [Val96].

Table 4. Precedence of entities in the salience list.

if $(x \in \text{OLD} \wedge y \in \text{MED}) \vee (x \in \text{OLD} \wedge y \in \text{NEW}) \vee (x \in \text{MED} \wedge y \in \text{NEW})$ **then** $x < y$
if $(x, y \in \text{OLD} \vee x, y \in \text{MED} \vee x, y \in \text{NEW})$ **then** **if** $(utt_x < utt_y)$ **then** $y < x$ **if** $(utt_x = utt_y \wedge pos_x < pos_y)$ **then** $x < y$

The reference resolution algorithm with Never look back is as follows:

- Process the utterance from left to right.
- If a reference is encountered, test the elements of the S-list in the given order until one test succeeds.
- Update the S-list just after an anaphoric expression is resolved.
- Update the S-list if a non-referential noun phrase is encountered.
- If the analysis of the utterance is finished, remove all discourse entities from the S-list, which are not used in the utterance.

2.2.3.1. Resolution of abstract entities

In [Eck99], [Eck99b] this algorithm is extended for resolution of abstract entities. In order to do this, a filter is used so that references to abstract entities and individual entities can be distinguished. This is done by looking for verbs like *is true, assume* [Gar97], which is summarized in table 5, where *I-incombatibility* means preferentially associated with abstract objects and *A-incompatibility* means preferentially associated with individual entities. References to individual entities are solved using an S-list, references to abstract entities are solved using an A-list, which contain abstract objects previously referred to anaphorically. These objects remain only for one turn. Checking for compatibility of candidate abstract referents is done in the following order:

- abstract entities in the A-List
- abstract entities within the same turn: Clause to the left of the clause containing the anaphor.

- abstract entities within the previous turn: Rightmost main clause (and subordinated clauses to its right).
- abstract entities within the previous turn: Rightmost complete sentence.

The first compatible entity is accepted as the referent.

table 5. Recognizing individual and abstract entities

I-incompatible *(Preferentially associated with abstract objects)*	**A-incompatible** *(Preferentially associated with individual objects)*
• Equating constructions where a pronominal referent is equated with an abstract object, e.g., *x is making it easy, x is a suggestion*. • Copula constructions whose adjectives can only be applied to abstract entities, e.g., *x is true, x is false, x is correct, x is right, x isn't right.* • Arguments of verbs describing propositional attitude which *only* take S'-complements, e.g., *assume*. • Object of *do*. • Predicate or anaphoric referent is a "reason", e.g., *x is because I like her, x is why he's late.*	• Equating constructions where a pronominal referent is equated with a concrete individual referent, e.g., *x is a car.* • Copula constructions whose adjectives can only be applied to concrete entities, e.g., *x is expensive, x is tasty, x is loud*. • Arguments of verbs describing physical contact/stimulation, which cannot be used metaphorically, e.g., *break x, smash x, eat x, drink x, smell x* but NOT **see x*.

2.2.4. Heuristic Algorithms

Another popular method to resolve references is the use of heuristics. Heuristics is a method where experimental rules are used to solve problems. Use of these rules is determined by trial and error experiments. There are several methods for this, which are discussed in the following subsections.

2.2.4.1. Training a decision tree

In [Mcc96] a training model is proposed to link those entities with each other, that refer to the same object. This model is designed to extract only interesting pieces of information from large bodies of newspaper articles about joint ventures and terrorist bombing. Each new reference is paired with each previous reference in a text and categorized as coreferring (e.g referring to the same object) or non-coreferring. In order to form these pairs, the entities go through a decision tree, which contain domain-independent and domain-dependent features. These features are tests which can be answered with TRUE, FALSE, UNKNOWN. For example:

- Do the phrases come from the same trigger family?
- Do the phrase share a common, simple noun phrase?
- Is phrase 2 an alias of phrase 1?
- Does each phrase contain a different name?
- Does phrase *i* start with a definite article?

- Does phrase *i* start with an indefinite article?
- Are both phrases subjects in their respective clauses?
- Do the two phrases occur in the same constituent?
- Do the phrases share a common head noun?
- Do the phrases share a common modifier?
- Do the phrases share a common head noun or modifier?
- Do the phrases share a common, simple noun phrase?
- Do the phrases agree in gender?
- Is phrase 1 the most recent phrase that is compatible with phrase 2?

These features are built into the decision tree using a machine learning algorithm based on a corpus.

2.2.4.2. Stochastic model for heuristics

In [Mur96] a stochastic model is proposed where probability weights of the referential properties of an entity and the candidate referents are calculated. Rules to determine the referential property of noun phrases include:

- When a noun is modified by a referential pronoun, *this, its*, etc.
  Then { indefinite (0,0) definite (1,2) generic (0,0) }
- When a noun phrase is accompanied by a particle *to, up to* or *from*
  Then { indefinite (1,0) definite (1,2) generic (1,0) }
- When a noun phrase is accompanied by *of*, and it modifies a noun phrase
  Then { indefinite (1,0) definite (1,2) generic (1,3) }

The two numbers between parenthesis are the possibility and the probability weight (ranging from 0 to 10) of having the referential property. The entities are tested for each rule and the probability weights are added. The property with the highest number will be assigned to the entity.

To determine referents of noun phrases, the following three constraints are made:

1. Referential property constraint: When a noun phrase is estimated to be a definite noun phrase, the system judges that the noun phrase refers to a previous noun phrase which has the same head noun. Else the system gets a possible referent of the noun phrase from topic and focus, and determines the referent of the noun phrase using the plausibility of the estimated referential property that is a definite noun phrase, the weight of a possible referent in the case of topic or focus and the distance between the estimated noun phrase and a possible referent.
2. Modifier constraint: When two noun phrase's have different modifiers they commonly do not refer to each other.
3. Possessor constraint: For example a part of a body can only refer to a human or animal.

Referents of noun phrases are determined by rules, which state the probability of a referent.

- When a noun phrase is like *the following*: {(Next sentences, 50)}
- When a noun phrase is the word *oneself* {(The subject in the sentence, 25)}

- When a noun phrase is estimated to be a definite noun phrase, and satisfies modifier constraint and possessor constraint, and the same noun phrase X has already appeared {(Then NP X, 30)}
- When a NP is estimated to be a generic NP {(Generic, 10)}
- When a NP is like *together* and *true*, which is used as an adverb or an adjective {(no referent, 30)}
- When a NP X is not estimated to be a definite NP {(A NP X which satisfies modifier constraint and possessor constraint, W – D + P + 4)}
  W = weight of topic and focus, D = distance between estimated NP and the possible referent, P= plausibility.

Similar type of rules are defined for pronoun resolution and references to properties of entities (indirect anaphora).
In [Byr99] a combination is used of fixed rules and heuristic rules. The fixed rules are used to filter out the entities which are incompatible. The heuristic rules are used to calculate the salience of the entities. Determination of the probabilities of the heuristic rules are done with genetic algorithms or data mining.
In [Ken96] a combination of 10 contextual, grammatical, and syntactic constraints are used to calculate the salience. The algorithm presented here is unique in that it does not need a parser, but uses a tagger instead (see section 2.1.4).

2.2.4.3. Experimenting with different configurations of rules

In [Mar00] an experiment is done with different configurations of rules. These rules are based, intuitively, on the following three steps:

a) anaphoric accessibility space definition,
b) application of constraint system, and
c) application of preference system.

The experiments were conducted using 40 spoken dialogues that have been obtained by means of the transcription of conversations between a telephone operator of a railway company and users of the company. The adjacency pair (a pair of turns in a conversation, each by different speakers, the first requiring an answer) [Fox87] [Sac74] and the topic of the dialogue were used in order to define the anaphoric accessibility space. Concretely, an anaphoric accessibility space is defined by means of the adjacency pair of the anaphor, the previous adjacency pair of the anaphor, adjacency pairs containing the adjacency pair of the anaphor, and finally, the main topic of the dialogue.
Morphological agreement constraints and C-command constraints (see [All95] for more on C-Command constraints) and the following preferences were used in the experiments:

- Preferences in the case of pronominal anaphora:
  1. Candidates that are in the same adjacency pair as the anaphor
  2. Candidates that are in the previous adjacency pair to the anaphor
  3. Candidates that are in some adjacency pair containing the adjacency pair of the anaphor
  4. Candidates that are in the topic
  5. Candidates that are proper nouns or indefinite NPs
  6. If the anaphor is a personal pronoun, then preference for proper nouns
  7. Candidates that have been repeated more than once
  8. Candidates that have appeared with the verb of the anaphor more than once

9. Candidates that are in the same position as the anaphor with reference to the verb (before or after)
10. Candidates that are in the same syntactic constituent (they have the same number of parsed constituent as the anaphor)
11. Candidates that are not in CC
12. Candidates most repeated in the text
13. Candidates most appeared with the verb of the anaphor
14. The closest candidate to the anaphor

- Preferences in the case of adjectival anaphora:
  1. Candidates that are in the same adjacency pair as the anaphor
  2. Candidates that are in the previous adjacency pair to the anaphor
  3. Candidates that are in some adjacency pair containing the adjacency pair of the anaphor
  4. Candidates that are in the topic
  5. Candidates that share the same kind of modifier (e.g. a prepositional phrase)
  6. Candidates that share the same modifier (e.g. the same adjective 'red')
  7. Candidates that agree in number
  8. Candidates more repeated in the text
  9. Candidates appearing more with the verb of the anaphor
  10. The closest candidate of the anaphor

Different preference configurations were tested on the corpus. Depending on the result of the test, some preferences were disabled and other were enabled. These experiments, where is attempted to find the configuration of preferences which has the highest performance, demonstrate that:

- the definition of an anaphoric accessibility space based on dialogue structure, and the set of preference according to this structure, helps anaphora resolution.
- traditional anaphora resolution systems are not easily transferable to other kinds of texts.
- anaphora resolution in dialogues requires an hybrid system able to combine linguistic information plus main topic information. In this case, the task that requires a greater effort is to find a method that combines both approaches.

2.2.5. Summary of resolution methods

In the previous sections four different principles to determine the preferred referent are discussed: the recency constraint, the centering model, the given-new principle, and heuristics.
The recency constraint is a very simplistic model. While the model is quite intuitive, the performance is not very high (except for the simpler type of references), because the focus of attention is not taken in account. What the recency constraint basically does is look up the most recent compatible object, and returns it as the referent.
The centering model goes a step further, and is based on the theory that most discourse is organized around an object that the discourse is about. It assumes that references are most likely to refer to this object. This model is very popular, but in [Pas96] it was found that

centering transitions (from both the model discribed in [All95] and the variant described in [Kam93]) does not directly reflect the segmental structure of a discourse, meaning that shifts in the center of attention does not correspond well with shifts in topic in the discourse.
The model described in [Str98] is based on the given-new principle [Pri81]. The model first started as an extension of the centering model [Hah96] [Str95] [Str96] [Str96b], but slowly developed into a model which does not look at what the discourse is about anymore, but assumes the focus of attention is determined by using discourse old objects.
Use of heuristics to determine the referent is very popular and is often actually implemented, instead staying stuck on the theoretical basis, where the model is tested with the assumption that the necessary data is actually available. The heuristic rules which are implemented are usually application specific though, and cannot be used for other applications.
The model described by Strube is very easy to understand, and no complex data is needed to find the focus of attention. This in contrast to the centering model, which is more complex and needs more information, which is more difficult to retrieve. Besides, [Kam93] showed that the behavior of the centering model does not correlate well with shifts in topic. In addition performance tests from [Str98] show that the performance of both centering models is lower than his. Heuristics can obtain good performance, and the fact that they have often been implemented shows that they indeed work in a natural language understanding application. The model described in [Mur96] can be used without complex data, because probabilities are assigned by looking at the phrases used. It is therefore suitable for use in a speech understanding application. Unfortunately the rules specified are meant for the Japanese language, and cannot easily be ported for application in English. For heuristic models to achieve very high results, it is necessary to use rules which are very application specific [Mcc96], so when a different application is used, new heuristics must be implemented.

2.3. Introduction to Ellipsis Resolution

Ellipsis is identified when a syntactic tree is built for a sentence, and some nodes of this tree are found to be empty. These empty nodes refer to an entity in a previous sentence. In [Keh93b] an algorithm to solve ellipsis is described. First the phrase which contains the referent of the ellipsis must be identified (the source), and the structure of this phrase with unfilled roles must be used. These roles are filled with entities from the phrase containing the ellipsis (the target).Then the remaining empty roles must be copied from the source:

a) Identify *parallel elements*, i.e. the objects in the source representation corresponding to the empty roles in the target.
b) All role fillers may be (i) *referred to*, where the appropriate function is used to link the role filler to the corresponding object in the source representation. In the case that the role filler is a function *with a link to the source event*, it may also be (ii) *copied*, where a new instantiation of the function is created and the source event variable is replaced with its corresponding parallel target event variable.

For example: *John likes his mother and Bill does too* has the possible readings: *John likes John's mother and Bill likes John's mother* or *John likes John's mother and Bill likes Bill's mother*.
The representation for the source clause (*John likes his mother*) is:
e_1: [predicate: like
agent: John
theme: [obj: mother
poss: agent(e_1)]]
The parallel event for the target is constructed (Step 1), and *Bill* is added as the agent (Step 2):
e_2: [predicate:
agent: Bill
theme:]
Step 3b can only *refer to* the value of the *predicate* role. Since the theme of the source event contains a referential link to the source event itself, Step 3b allows the theme to be *referred to* with a function *copied* by creating a new instantiation of the function occupying the theme and replacing the event variable e_1 with its parallel event variable e_2.
e_2: [predicate: like
agent: Bill
theme: [theme(e_1)]
or
e_2: [predicate: like
agent: Bill
theme: [obj: mother
poss: agent(e_e)]]
The same applies to anaphora like: *John got shot by his father. That happened to Bob too.* and *John kissed his wife, and Bill followed his example.* and *Although John bought a picture of his son, Bill snapped one himself.*

Chapter 3.

The Anaphora Resolution Module in the SPICE-EPG

In this chapter the steps which are taken before the implementation of the reference resolution module are described. First an analysis is made of the requirements for the module. Because it is not possible to meet all requirements, the scope must be narrowed down. For this an analysis is made of what is feasible during the period of this project, and within the constraints of the environment. Priorities will be set for certain tasks, so that the most important parts of the anaphora resultion module can be implemented. Once the scope of the project is defined, a model is chosen which will be used for anaphora resolution. Based on this model the information needs are determined, and methods to provide for these information needs are discussed.

3.1. Requirements for the module: Must-haves and Should-Haves

Since the SPICE-EPG prototype is the environment where the reference resolution module will be running, the majority of the requirements are derived from the present situation of the system and the view on how the system ideally should be. Other requirements are based on future use of the module, possibly in different contexts. The requirements are divided into two different types: Requirements the reference resolution module must have, and requirements the reference resolution module should have. These will be explained in the following subsections.

3.1.1. Must-haves

In this section an overview will be given of the requirements that are strictly necessary for the reference resolution module. In short they are:

- Reference resolution.
- Operational within SPICE.
- Operational in real-time.
- Not dependent on an extensive lexicon.

3.1.1.1. Reference resolution

Naturally, the system should be able to resolve the references which are used in the application. But in man-machine interaction, similar to the one encountered in a dialogue system like the SPICE-EPG, not all types of references will be used. In fact many types of references which are found in written texts will not be used in this type of man-

machine interaction. On the other hand, references to entities in another modality can be expected.
The types of utterances encountered are user requests for information on television programming schedules and programs, and commands regarding searching, recording, reminding, and switching to programs. No new information will be introduced by the user, which is not already present in the database, and the entities used are limited to the context of operating an electronic programming guide. In appendix A an overview is given of what co-workers view as what should be possible in an ideal natural language understanding EPG. Use of the following references can be extracted from this data:

- Definite descriptions. For example: *Show me information about **the first program***.
- Pronouns. For example: *Are there any other movies with **her**?*
- Demonstratives. For example: *Record **that.***
- Ellipsis. For example: *Are there any movies with Robert Redford today? **How about tomorrow**?*
- One anaphora. For example: *Remind me of **the one on Channel 5.***
- Deixis. Where the user for example points to a program in the list.

These references can refer to the following type of entities:

- an entity that was introduced into the discourse via a noun phrase. For example: *What is on CNN right now? Switch to **that channel (CNN)***.
- a property of an entity that was introduced into the discourse via a noun phrase. For example: *Show me information on CNN world news. Are there any other programs at **the same time (the time of CNN world news)**?*
- a superset of individual entities from another modality. For example: *Please record **the Mad Max movies (Mad Max 1, Mad Max 2, ...).***
- world knowledge. For example: *Is there any news on **the latest earthquake**?*
- fact. For example: *Is she not beautiful? Oh, I forgot you are a computer, you do not know anything about **this (she being beautiful)***.
- an entity from another modality. For example: *Can you show me more information about **this movie (movie user just pointed to)**?* or *Record **the second program (program displayed on the screen).***

3.1.1.2. Operational within SPICE

The reference resolution module must be able to operate within the SPICE-EPG context. This means that it must be able to perform its tasks with the data provided by the system, and not be dependent on technology which is not available within the SPICE-EPG. Because of this the reference resolution module must be able to operate with the data generated by the shallow parser, which means that no deep syntactic and semantic information is available. A method must be found to compensate for this lack of information.
The reference module will be part of the context interpretation module, and will only perform actions locally. Ideally reference handling should also be part of the dialogue management, but this will not be the case here, since it is not possible to access this part of the system. It will therefore not be possible to handle references to things out of context.

Within SPICE-EPG communication between the different modules is done in XML, and as such the reference resolution module should be able to read data in XML format.

3.1.1.3. Operational in real-time

The reference resolution module will be used as part of a man-machine speech understanding interface, and as such all data must be processed in real-time. The user cannot wait very long for a response, so the chosen reference resolution algorithm must be fast, and not too complex.

3.1.1.4. Not dependent on extensive lexicon

The module must not be dependent on an extensive lexicon of words, which specify their syntactic, semantic, functional properties and all dependencies between the different words. It must be able to operate with as little information as possible.

3.1.2. Should-haves

In this section an overview will be given of the requirements that are not strictly necessary, but are still important for the reference resolution module to have. In short they are:

- Robustness
- Adaptable for other applications
- Parameterized settings
- No increase in system requirements
- Little increase in processing time
- Written in C++

3.1.2.1. Robustness

Nowadays state-of-the-art speech recognizers are still far from perfect. As a result, speech recognition errors are still very common. The reference resolution module will run in such an environment, and it can be expected that misrecognitions of the user by the system will be processed by the reference resolution module. The module should be robust for these cases, otherwise recognition errors may upset the entire system, and the user may have to start all over again, before he can be understood correctly. To avoid this kind of frustration, it would be best if the system would not be dependent on correct information only, but is able to find the information needed, or recognize wrong information. The parser uses a N-best list to find the set of phrases which matches the user's utterance as best as possible. The reference resolution module should be able to determine the references in the N-best list, aiding the dialogue manager in selecting the best hypothesis.

3.1.2.2. Adaptable for other applications

Even though the SPICE-Electronic Programming Guide was designed as a prototype to show the possibilities of current state-of-the-art technology within Philips Research, it should not be the case that the reference resolution module will only work in the SPICE-

EPG environment. It should be possible to do some simple modifications in order to adapt the reference resolution module for other types of applications.

3.1.2.3. Parameterized settings

Because many things are still uncertain, about how entities from other modalities come into focus, how this may differ depending on the combinations of modalities used as input and output, and whether the user is in fact performing a monologue or a dialogue with the system, it would be convenient if the settings which are expected to affect this focus can easily be changed without the need to alter the code.

3.1.2.4. No increase in system requirements

In the end the EPG application should be able to run on a device with limited memory and processing power. In addition the SPICE-EPG prototype is currently quite heavy, so it is desirable that system requirements will be kept to a minimum. Even though the processing power double yearly, even for handhelds, and eventually it the system requirements will not be any problem, it has been specifically stated that increase in system requirements are undesirable, and no heavy third party software should be used.

3.1.2.5. Little increase in processing time

Since the user is in constant interaction with the system, it is important that information is returned in as little time as soon as possible. Eventually machines will be fast enough to perform the speech recognition in a fraction of a second, but currently the user already has to wait a few seconds before receiving information, and it has been specifically stated that any increase in processing time is undesirable. Large increases in processing time will also lead to problems during testing, because the processing time will allow only few tests in a certain period of time.

3.1.2.6. Written in C++

Within Philips it has been decided that the standard programming language is C++. Since the program is written for Philips and other people must also be able to understand and expand the program, the programming language should be C++.

3.2. Narrowing the scope

In order to meet all these requirements and build a really good system, many years of work and research will be needed. Also, some of the requirements interfere with each other. The fact that the module must be operational within SPICE-EPG and may not depend on technology which is not available within SPICE-EPG, puts a severe limitation on the types of references which can be solved. Most state-of-the-art anaphora resolution algorithms depend heavily on deep parsing, which is not available within SPICE-EPG. Also, the fact that it is not possible to use an extensive lexicon, which specify syntactic, semantic, functional properties and all dependencies between all the words that are expected to be encountered, is an additional limiting factor. On the other hand, considering the time available for this project, it would not have been feasible to build such a lexicon anyway. Because of this the scope of the project must be narrowed, so the

most important and the most feasible requirements will be met within the predesignated time for this project. In the next subsection an analysis is made of the references which are expected to be encountered in this application. The final subsection will give an overview of what can be included in the scope of this project. Once the reference resolution model and the means to provide for its information needs are established, the general outline of the algorithm is determined. Based on this outline, the design for the module is made, and this chapter ends with an overview of what each class in the module does.

3.2.1. Solving references within the constraints

Determining which references will or will not be solved by the module is usually done by examaning a corpus, and determine which references are used the most and which are used the least, so that the references used most will at least be implemented.
Unfortunately no such corpus was available, because in the past references were carefully avoided, when corpera were built. Therefore the co-workers at Philips were asked to phrase several examples of the references which ideally should be solved (see appendix A). This provides no information however, on how often these references will be used, and how important they will be. Therefore the set of references which will be solved are based on the information available and the authors thoughts on how important each reference will be.
In this section a short overview will be given of each of the reference types which are encountered in appendix A in order of appearance. For each type will be stated what kind of information is needed, and how much of this information will be available in the environment where the reference resolution module will be running.

3.2.1.1. Ellipsis

Ellipsis is commonly used in natural language and much research has been done in this area. In general the resolution method for ellipsis is as follows: When a parse tree is built, either a full syntactic tree as is shown in figure 9. or a dependency tree as is shown in figure 12, ellipsis will be detected if an empty node is encountered. In the example: *SPICE, are there any movies starring Mel Gibson today? How about this week?* The second sentence is missing both a subject as well as a predicate, or in the case of a syntactic tree: both a noun phrase and a verb phrase are missing. To solve the ellipsis, the following steps must be taken: First the empty structure of the syntactic tree of the phrase containing the referent of the ellipsis is created. Then the nodes of this tree are filled with the phrases of the referring sentence. Finally the remaining empty nodes are filled with the phrases of the referential sentence.
In the SPICE-EPG environment, it is not possible to find any empty nodes, since the shallow parser only returns meaningful concepts, which contain no information about its function or place in the sentence. Detection of ellipsis will become very problematic and is impossible using the conventional method. An option would be to look for clues in the sentence like *'How about,' 'and so did,' etc.* But this is no insurance that there is indeed a case of ellipsis, and lack of these indictors certainly does not mean there is no case of ellipsis. Additionally, many so called indicators are of the form *'<indicator-part1> …*

<indicator- part2> …,' which is not possible to extract as such, using the shallow parser. Besides that, it will be hard to determine what is missing in the sentence, if it has been decided that ellipsis occurs. It would be possible to assume that if there is an indicator for ellipsis, and a slot is empty which contained data in the previous turn, that ellipsis is encountered (the system uses a slot filling strategy, which means that attempts are made to get all required data in order to fulfill a request). But this is hardly a very robust way to handle ellipsis, since speech recognition errors may add or remove a concept which indicates ellipsis or a slot value, which may result in a very strange behavior of the EPG. On the other hand, the dialogue manager of the SPICE-EPG system already keeps slot values, unless certain condition occur (e.g. the reset command), and as such it will probably be not worth the time, effort and extra overhead to create a module which fulfills a function already performed by the dialogue manager. Therefore this kind of reference has low priority.

3.2.1.2. References to an entity from another modality

References to an entity from another modality are expected to be very common, since most of the information presented to the user are displayed on a screen. The user can refer to this information with speech only or use pointing input additionally. This can be done with pronouns, demonstratives or definite descriptions. Pronouns and demonstratives will usually be accompanied with pointing input if there are multiple possibilities of what the pronoun or demonstrative can refer to. For example when there are many programs on the selection list and the user wants to pick a single program for recording, the user might say: *Record this,* while pointing to the appropriate program. If there is only a single item on the display list, the user can say: *Record it*, without pointing to any program. The user may use definite descriptions to specify objects which are in or out of focus, or pick a single entity when there are many entities on a selection list, for example: *Record the program,* or *Show me the previous list,* or *Record the fourth program.*

To solve these kinds of references, information is needed about what the system output was, what the user has pointed to (if applicable), and what is in focus. To determine what is in focus, information like the subject, or object of the sentence are needed for most algorithms [All95] [Kam93]. This is not possible with a shallow parser alone, but there are algorithms that do not need this information directly [Str98]. Otherwise, if this information is needed, a tagger may provide a solution to this problem [Ken99].

In the SPICE-EPG system it is already possible to determine what the user has pointed to, and match it to the reference in the multimodal integration module. It is still necessary though to find a method to solve references to an entity from another modality if no pointing is done.

These kind of references are expected to be one of the most occurring types, and should be the least what can be resolved.

3.2.1.3. References to a superset of individual entities from another modality

It is possible that the user wants to refer to a group of entities which have been displayed on the screen. This is usualy done with definite descriptions. For instance when the user is a Star Trek fan and wants to record all the Star Trek episodes displayed on the screen,

the user might say *Record all the Star Trek episodes*. To resolve these kind of references, the system must know that the programs are part of the Star Trek series. The information that a program is part of the Star Trek series might be part of the content description, or the title, but is not provided as a seperate attribute by the system for each program. It will therefore be very difficult to determine which programs belong to the Star Trek series locally in during context interpretation. This would be more a task which should be handled in the dialogue manager, but since no access is granted to modify this part, it will be very hard to handle these kind of references. In the case that the user wants to select multiple programs on the same channel or of the same genre, an attribute which is provided by the system for each program, for instance in *Record the programs on CNN*, or *Record the sport programs,* it will be easier to solve, since the information necessary is readily available.
To refer to a superset of individual entities from another modality it is not always necessary to use definite descriptions. If the user wants to refer to every item, pronouns and demonstratives can be used: *Record them*, or *Record these*.
To be able to refer to a superset of individual entities. information on the grouping criteria is needed. In certain cases these are available, in other cases it will be very difficult using the present database structure.
Even though it is possible to resolve references to a group of entities in certain cases, the dialogue manager will not be able to handle it. This should not be a reason not to implement it though, because future upgrades or applications might be able to handle multiple selections.

3.2.1.4. References to a property of an entity from another modality

Sometimes the user may want to refer to a property of an entity. For example: *Give me information on the director of this movie*. Again information is needed about the properties of the entity, which in this case is again part of the description. In this case it is even harder to find the referent, because in the description there is no fixed tag to specify who is the director of the movie. If the property is a date, time, channel or genre, it can be recognized more easily.
Another possibility is that the user wants to refer to an item of a list. For example: *Record the fourth program.* Here information is needed about what items are in the list and in what order. In the current system, there are still some troubles because the display server decides on its own how things are displayed. This is a minor problem, and will only cause errors when there are more items on the list than can be displayed and the user is counting from bottom upwards. For instance: *Switch to the second program from below* refers to the program before the last displayed program, but will be resolved to the last program on the list.
Being able to resolve references like *the fourth program* and *the second program from below* is very important, because the system output consists most of the time of lists of programs, and these references are very easy for the user to use.

3.2.1.5. References to an entity that was introduced into the discourse via a noun phrase

These types of references are perhaps one of the most basic types of references encountered. It is well researched topic and there are algorithms which might work in an

environment with shallow parsing. With these types of references the user can refer back to an entity which he said before. For example *What is on CNN right now? Switch to that channel*. Because this type is one of the most basic types, it should be one of the least to be implemented.

3.2.1.6. References to world knowledge not mentioned in the discourse

World knowledge includes the general knowledge about the structure of the world. In the case of an electronic programming guide, one may think about events in the world which may relate to certain programs, program schedules, and user properties. For example, when an earthquake struck Turkey, the user may be interested in *the latest earthquake*, or the user may want to schedule *the next match* after watching a baseball game, or requests information about sports featuring *his favorite baseball team*.
These kind of references should be handled by the dialogue manager, because this part of the system contains both world and local knowledge. Also the dialogue manager may decide to ask for clarrification when it is not sure about the user's intents. It is very difficult to handle these kind of referece as part of context interpretation, since these references are out of context. Since it is no access is granted to modify the dialogue manager, these references will not be part of the scope of the project.

3.2.1.7. References to a fact

In one of the examples in appendix A, there is a reference to a fact: *Is that Sandra Bullock? Isn't she beautiful? Oh, I forgot you are a computer, you do not know anything about this.* Here *this* refers to the fact that *Sandra Bullock is beautiful*. Even though there is an example of a user conversation with the EPG using this kind of reference, it is not expected that it will be used in this type of application. The current system uses a slot filling strategy, and is only interested in finding the required information. Therefore it will not engage in a conversation with the user. Instead it will try to direct the user to provide the necessary information to fulfill a certain task, which in this case does not include facts like *Sandra Bullock being beautiful*.

3.2.1.8. References to nothing at all

Although this type of references is not encountered in the examples, it is important that pronouns and demonstratives not referring to anything are recognized. Attempts to resolve these kind of references are a waste of processing power, and may even result in upsetting the focus of attention, or frustrating the user.
To recognize these kind of references it is necessary to find the general form in which they occur. For example: *It seems … , it appears …, etc.…*

3.2.2. The narrowed down scope

Considering the information presented in the previous section, the scope can be narrowed down to the following requirements:

Must haves
- Resolution of references to an entity from another modality.
- Resolution of references to an entity introduced previously via a noun phrase.
- Resolution of references to a property of an entity from another modality.
- Operational within SPICE-EPG.
- Operational in real-time.
- Not dependent on an extensive lexicon.

Should haves
- Robustness.
- Adaptable for other applications.
- Parameterized settings.
- Resolution of references to a superset of individual entities from another modality.
- Filter out references to nothing at all.
- No increase in system requirements.
- No increase in processing time.
- Written in C++.

Note that resolution of references to a property of an entity as a must have is limited to the properties which are predefined, and do not include properties which have to be extracted from the title or the description.
The same applies for references to a superset of individual entities. These entities can only be grouped by predefined properties, and not by information which have to be extracted from the title or the description. This type of reference is put under the should haves, because the SPICE-EPG system is not capable of handling multiple selections.

3.3. Choosing the reference resolution method

In this section a comparison is made for the different reference resolution methods described in section 2.2. Using this comparison a method is selected which is used for the reference resolution module in the SPICE-EPG. Table 6. shows an overview of the properties of the different reference resolution methods and some results obtained from literature.

Table 6. Overview of the properties and results of the different reference resolution methods.

Section	Method	Parse	Imp	Perf	Corpus
2.2.1	Simple History List	phrase	easy	47% [1]	Train93
2.2.2	Centering	struct/	med	72,9% [2]	a)
	Temporal Centering	seman		76,0% [2]	a)
2.2.3	Never Look Back	phrase	med	85,4% [2]	a)
2.2.4	Heuristics [Byr99]	synt/	hard	69,1% [1]	Train93
	[Mcc96]	seman		92,4% [3]	MUC-5 + 6
	[Mur96]			78% [4]	b)
	[Mar00]			73,8% [5]	Basurde
	[Ken96]			75%	27 random texts

Parse = parsing information needed, phrases, phrase structure, semantic information, syntactic information
Imp = difficulty of implementation
Perf = performance. Tested in [1] [Byr99] [2] [Str98] [3] [Mcc96] [4] [Mur96] [5] [Mar00]

a. texts from the information technology domain, text from the German news magazine 'Der Spiegel,' a short story by 'Heiner Müller'.
b. Grammar book "Usage of English articles", "The Old Man with a Lump" "Tensei Jingo".

Considering the sentences and types of references which are to be solved in the SPICE-EPG (section 3.2), using the simple model for anaphora resolution (section 2.2.1) will probably be quite adequate. It is easy to implement and no additional requirements on the parser are needed, since the only information needed to resolve references are recency, gender, person, and number. However, in other applications where the utterances go beyond simple commands and more complex, but still simple constructions are used, the performance of the performance of the algorithm will drop dramatically, and will become quickly obsolete.
The centering model (section 2.2.2) is capable of handling more complex constructions than simple commands, but in order to function properly it needs information about sentence structure, syntax, and the role of the various phrases. For this a deep parser will be needed, which will increase processing time, system requirements, and will not operate robustly in an environment where speech recognition errors are common, and where utterances are not grammatically correct, according to text grammar rules.
Never Look Back (section 2.2.3) is equally capable of handling more complex constructions than simple commands, and according to [Str98] it performs even better than the centering model. To calculate the most salient entity, no information about sentence structure, syntax and the role of the various phrases are needed, so it is expected to work well with a shallow parser. For intrasentential references, information about sentence structure is still needed though, to determine whether a salient entity meets the binding constraints (section 2.1.1.1). It is expected that this will occur rarely though.
With the heuristic approaches to solve references (section 2.2.4) a lot of rules are needed before good results are achieved. It is highly probable that satisfying results will not be obtained within the specified period for this project, especially because most rules given in the papers are not suitable for either English language or are too specific for a certain domain. In addition, most rules for these heuristic approaches need information about syntax and the role of the various phrases, which is not available with a shallow parser. An interesting approach is described in [Ken96] though, where the parser is substituted with a tagger, which provides information on the role of each word in the sentence.

Never Look Back achieves higher performances in resolving anaphora than the centering model and the simple model based on a history list. It is difficult to compare it with the heuristic approaches, but since it is easier to implement, and does not need information to determine the most salient entity, which the shallow parser cannot provide, this method will form the basis of the reference resolution module in SPICE-EPG. The simple model based on a history list, may be adequate for the SPICE-EPG program, and is easier to implement, but Never Look Back has more potential, and is expected to be usable in more advanced applications as well. Therefore Never Look Back will form the basis for the reference resolution module.

3.4. Grammar requirements for the solution

In this section the requirements for the grammar are specified so that the references within the scope of the project can be solved. These requirements are based on the examples from appendix A.

3.4.1. Recognition of references

Before references can be resolved they must first be recognized as such. The forms of the references which are encountered are as follows:

- third person pronouns
- demonstratives
- definite descriptions modified by a definite article or a demonstrative
- one anaphora

The pronouns I, me, my, mine, we, us, our, ours, you, your, yours are not interesting in this concept, because pronouns in the first person will always refer to the one speaking and pronouns in the second person will always refer to the one spoken to. Besides that, they do not add any information relevant for performing any task of the EPG.
Third person pronouns and demonstratives are easily recognized.
For definite descriptions and the descriptive form of one anaphora every possible noun phrase must be specified. This is done by looking at the examples and extracting general information on the forms which are expected to be encountered. In appendix B the part of the grammar is listed which is used to recognize these forms.
One anaphora in its single form is more difficult to recognize with a shallow parser. With a syntactic parser, the word '*one*' will be recognized as such because it fills the role of a subject or an object. With a shallow parser no such information is available and the word '*one*' can be either a number, to specify an amount or time, or a reference. One anaphora of this type does not refer to a specific object, but rather to a general class of objects. For example: *Are there any movies tonight? Is there maybe one with Robert Redford?* In this case '*one*' refers to the class *movie*. In the SPICE-EPG system the category movie from the first sentence will be kept, and in the second sentence the constraint '*Robert Redford*' will be added. Since the system is already capable of handling these kind of situations, and these forms are difficult to recognize, attempts to device a solution to do so are set to low priority.

3.4.2. Recognition of objects which can be referred to

In order to solve references, it is not only important that the references are recognized, but naturally the objects which can be referred to must be recognized by the parser. In the SPICE-EPG the following objects can be in principle referred to:

- date
- time
- genre
- channel
- title
- actor
- director
- protagonist
- selection list
- record list
- remind list

From these objects the date, time, genre, and channel are already in the grammar. Actor, director and protagonist are not part of the grammar, but it is not possible to put them there. This will cause a conflict in the system, because once put in the grammar, it cannot be recognized as part of the content description anymore. The result is than that the system will not look in the description for this information.
Date, time, genre, channel, title, selection list, record list and remind list are objects which are displayed on the screen, and which can be referred to. Objects which are displayed on the screen must therefore be known to the reference resolution module.
In addition to that, date, time, genre, channel and title are objects which can be pointed to by the user. Therefore it is necessary that information is available about what is pointed to, which is generally not an easy task, because pointing events are not synchronous with the text, and it is often unclear what the user meant when something is pointed at. For instance when the user points at a time on a certain line in the screen, the user may refer to this time, or to the program in the same line, or even the whole line. Fortunately this is already solved in a previous project [Phi00].

3.4.3. Recognition of phrases adding contextual constraints

It is often not enough to recognize the references only and select the most salient entity to resolve the reference. Often words in the sentence provide a context which narrow down the scope of possible referents, and the most salient entity may just be not in this scope. Constraints from the context may come from verbs which object or subject specifies the type of referent. For example: *Record it!* or *Are there any other movies where she stars?*. *Record* can only apply to a program, so *it* probably refers to the program in focus. The subject of *stars* must be an actor, so *she* is probably the female actor in focus. Therefore

it is necessary to recognize the words which add information on the context, and where these words apply to.

3.4.4. Recognition of expletives

One last thing which must be recognized are the expletives, uses of *it* and *that* which do not refer to anything particular. Failure to recognize these expletives may cause the system to put unwanted information in slots, disrupting the task the user does. Appendix C shows a list of the forms in which expletives are encountered.

3.4.5. Adaptation of the SPICE-EPG Grammar

The grammar used in the SPICE-EPG system is a stochastic context-free grammer, based on shallow parsing, in which only meaningful concepts are recognized [Kel00]. A concept is a sequence of words or concepts and has a set of attributes. There are also no dependencies between concepts. The following example shows what the grammar may look like:

```
<PROGRAMME> ::= programme
<PROGRAMME> ::= programmes
…

<CATEGORY> ::= movie
  genre := ‘movie’
<CATEGORY> ::= movies
  genre := ‘movie’
<CATEGORY> ::= news
 genre := ‘news’
…

<REFERENCE> ::= this
<REFERENCE> ::= that
…

<INDEX> ::= first
<INDEX> ::= second
…

<DEFINITE_DESCRIPTION> ::= the <PROGRAMME>
<DEFINITE_DESCIRPTION> ::= the <INDEX> <PROGRAMME>
<DEFINITE_DESCRIPTION> ::= <REFERENCE> <PROGRAMME>
<DEFINITE_DESCRIPTION> ::= the <CATEGORY>
<DEFINITE_DESCIRPTION> ::= the <INDEX> PROGRAMME
<DEFINITE_DESCRIPTION> ::= <REFERENCE> <CATEGORY>
…
```

A concept contains no information about the concepts it consists of, so the concept RECORD_PROGRAMME := <RECORD> <PROGRAMME> will not be recognized later as a composite concept consisting of the concepts <RECORD> and <PROGRAMME>. It also does not contain the attributes of these concepts, unless explicitly specified. With the

grammar it is also not possible to specify concepts of the form <concept_part1> … <concept_part2>.

The information gained from the shallow parser is quite limited, but is the only option which is robust enough to deal with badly recognized speech. Ideally there should be a second parser which goes deeper into the sentence structure, so that binding constraints can be used to determine incompatibilities in intrasentential references. This syntactic parser should also be based on a stochastic grammar, so that probablistic information about the concept graph from the shallow parser can be used to determine the most probable syntax tree [Bod96]. Other options are use of the EngCG tagger described in section 2.4.1 [Ken96], and/or a set of filters to find as much information as possible about the sentence structure. The tagger is able to provide syntactic information and information about word function in the sentence, without parsing the sentence, but does so by looking at the word forms and verbs. It is faster than a parser, but unfortunately also less accurate, and provides no information about sentence structure. A filter is a simple routine which looks for certain specific phrases, in the utterance and either modifies this phrase for later handling, or extracts information from this phrase.

The advantages of a second syntactic parser are:
- Information about sentence structure
- Information about dependencies
- Information about word function
- Might be robust parsing in combination with partial parsing

The disadvantages of a second syntactic parser are:
- Time intensive to implement / Expensive
- Large increase in processing time
- Increase in system requirements

The advantages of a tagger are:
- Information about word function
- Limited increase in processing time

The disadvantages of the tagger are:
- No information about sentence structure
- No information about dependencies
- Time intensive to implement / Expensive
- Increase in system requirements
- Not very robust with textual grammatically incorrect sentences

The advantages of filters are:
- Easy to implement
- Little increase in processing time
- Little increase in system requirements
- Can work with textual grammatically incorrect sentences

The disadvantages of filters are:

- No information about sentence structure
- Little information about dependencies
- No information about word function

This is summarized in table 7.

Table 7. Properties of methods to acquire syntactic information.

	struct	depend	word func	proc time	sys req	robust	imp
Parser	√	√	√	high	high	high	hard
Tagger	×	×	√	med	med	med	hard
Filters	×	√	×	low	low	high	easy

The problem of a second parser is that there is a large increase in processing time and system requirements. It is expected though that in combination with the stochastical partial parser, which is currently used, the syntactic parser is able to provide information about sentence structure and dependencies adequately.
Use of the EngCG tagger is an interesting option, because it provides information on word function, so that constraints for intrasentential anaphora can be used. A license for this tagger is quite expensive though, and implementation will take quite a long time. The processing time of the tagger is acceptable (this is tested with the demo on the website), but the increase in system requirements, though not really a problem, is undesired. The tagger has one problem though: when a very distorted sentence is processed, certain phrases will be incorrectly tagged.
Use of filters alone provide very little information but may be adequate in the SPICE-EPG environment. It is easy to implement, and requires little additional effort from the system. The filters will not get confused when dealing with strange structured sentences, which is traded off by the accuracy of the information it returns. Since it is expected that the use of filters alone will be adequate, the reference resolution method will make use of them. If this would prove to be inadequate, the tagger will have to be put into use.

3.4.6. Use of methods to compensate lack of syntactic information

In the ideal case every meaningful concept and subconcept (concepts part of another concept, like in *the program on CNN*, where *CNN* is a part of the concept *the program on CNN*) and the dependencies between them are recognized as such. As mentioned before, it is not possible, and in many cases it is not strictly necessary, because dependencies can be inferred by looking at the types of concepts. For instance in the phrase: *Please record it*, it can be inferred that *record* applies to *it*, and that *it* therefore must be a program. It is clear that *record* does not apply to *please*. This can be done for example by specifying that if a concept has the constraints *gender = neutral* and *abstract = nonabstract*, and the concept *record* is found in the utterance, which is of the concept type *command*, the constraint *type = program* can be added. So by looking at the concept values and the

types some dependencies can be inferred. In the module this is actually done by looking at the constraints of the concepts, to allow an even more accurate matching of dependencies between concepts. Having two seperate concepts is especially useful when the two phrases do not necessary follow each other immediately, but can have fillers or other concepts between them.
In other cases it is better to group the concept and its subconcept into a single concept, for instance in the phrase *the six p.m. news*, it would not be possible to recognize *news* as a definite description, because *the* would be missing when the concepts are split up into *six p.m.* and *news*. Therefore in order not to lose this information, it is necessary to have *the six p.m. news* as a single concept. But because *six p.m.* is a concept which can be referred to (for example: *that time*), it is necessary to have it recognized as a distinct concept, and not just as a constraint for movie. For this reason, a special attribute *subconcept* is created in the grammar for definite descriptions. In this attribute the concept type and value are stated. For the subconcept attribute a concept is created, which is linked to the concept.
Because there is no secondary parser, it is difficult to determine whether some words in a concept like *channel* and *5* in *the one on channel 5* belong together, so that is indeed recognized as *channel 5* instead of two sepparate words. This is necessary to assign the correct constraints to the concept (*channel* might introduce the constraint *type = channel*). It would be possible to try to find all possible combinations of word groups, but this would be essentially part of the parser and is very time consuming. To solve this problem, words like *channel 5* are modified during concept creation into something like *channel5*. The information that the program is on *channel 5* will be provided by the subconcept. It also would have been possible to split up the concept *the one on channel 5* into two concepts, *the one* and *channel 5*, but in this case additional constraints must be added to provide the context for the program. If the two concepts are separate, it is more difficult to correctly match *channel 5* as a constraint to the one, then having *channel 5* be a subconcept of *the one*. This already provides the needed data structure to work. Another possibility would have been to remove the word *channel 5* from the concept value instead of replacing it with *channel5*, but this decreases the ease of understanding what happens when debugging.
There is a case where dependencies are more difficult to handle. In for instance the phrase: t*he second program of the previous list*, it is necessary to recognize *the previous list* and *the second program* seperately, because otherwise it is not possible to resolve *the previous list*, but it is also necessary to recognize the dependencies between them. In addition, *the previous list* must be resolved before *the second program is handled*.
Considering this, a dilemma is created: recognize them as distinct concepts, and have trouble finding the dependencies, or group them together as a single concept, and have trouble distinguishing the seperate concepts, which have to be resolved. If the this phrase is handled as two distinct concepts, it is necessary to find a cue in the sentence, which specifies the dependency and which concept should be resolved first. This may become problematic, because for each modifier i.e. *from, of, etc*. must be specified which concept must be handled first. Splitting the concepts also increases the chance that the modifier is not recognized, resulting into two seperate concepts with no relation at all. Therefore it is best to group them as a single concept, and specify information which concept must be processed first. Because *the second program* is the topic of the concept, an attribute *concept* will be created with the value *the second program*, and because *the previous list*

is what it is part of, an attribute will be created named *superconcept* with the value *the previous list*. A filter will split this concept while translating grammar data into the internal data structure, and create two concepts with the needed dependency. The superconcept is processed first before the concept is resolved.
The grammar is also inable to create a concept which forms a group for concepts mentioned together. A filter is needed to find this summing of concepts and create a container concept containing these concepts, so that the concepts as a group can be referred to.
Another problem with the lack of syntactic grammar is that misrecognition of the utterance can produce total garbage which will be processed by the system, resulting in output which is totally out of context (for instance random noise, or simple misrecognition will often result in the output of words like *a, the, it,* etc. which will be matched to a title starting with *the* or *a*, or resolution of *it*). This is very frustrating for the user. To prevent this a filter is necessary to recognize and remove these kind of concepts. This should be done in the info retrieval engine, where a list of 'stop words' could be applied to filter out these words from queries to the database.
One last problem of lacking a syntactic grammar is the absence of binding constraints. Only one experimental constraint is implemented: an accusative pronoun cannot refer to the most recent concept. This is implemented so that at least references in the following most basic sentence structure can be resolved: '*Bob and Bill met each other at the mall, he gave him a book.*' Using this binding constraint '*him*' cannot refer to '*he*'. Ofcourse this will not work in more complex sentence structures.

3.4.7. Summary of grammar requirements

For the reference resolution module which is based on the model described by [Str98], the following grammar must be able to do the following:

- Recognize the references.
- Recognize the objects which can be referred to.
- Recognize phrases which add contextual constraints.
- Recognize forms where expletives occur.
- Provide information on relationship between concepts.

Recognition of the various concepts can be entirely done by the grammar. To find the information about the relationship between concepts, additional filters are required though. Determining whether a concept has such a relationship with a reference, that it can add contextual constraints to it is done by looking at the constraints of the concept. A concept which modifies the concept and adds contextual constraints to it, has some requirements before the contextual constraints are assigned. These requirements are tested against the constraints already found for the concept. Another method used to determine the relationship between two concepts which follow each other, is to create a single concept with both phrases in it and specify the relationship between them in the attributes.

3.5. General outline of the algorithm

Having determined the method to resolve the reference and the information provided by the grammar, the general outline of the algorithm can be specified. In general the anaphora resolution model in SPICE-EPG consists of:

- a set of filters,
- a database,
- a salience-list,
- a history list, and
- routines to find the referent.

The database consists of information concerning constraints and properties of objects which are expected to be encountered in the discourse and are used to determine the compatibility of the reference and candidate referent.
The salience-list consists of the objects which are currently in focus and are sorted from most salient to least salient according to [Str98]. Because the dialogue is infact a man-machine interaction, the entities not used in an utterance are only removed in the user turn, instead of each utterance as proposed by Strube.
The history list consists of all objects encountered in the discourse, which are grouped in type lists according to type and sorted in order of recency of use, to increase accessibility. The history list is used for references to entities out of focus.
The filters are used to filter out uninteresting phrases and limit the scope for searching possible referents.
The processing of the information can be split into two parts, namely the system information processing part and the user information processing part. The system information processing part is as follows:

- The SPICE-EPG display provides information on the items on the screen.
- These items are converted into an internal representation of concepts similar to the concepts provided by the grammar.
- List concepts of the different types are created which contain the concepts of the appropriate type. All non-program types are added as subconcept to the corresponding program concept.
- Group concepts are created depending on several grouping criteria, so that the concepts can be referred to as a whole.
- Concepts are sorted.
- The group concepts are added to the salience list and all concepts are added to the corresponding type list of the history list. Concepts are ordered so that the ones at the top of the display will be accessed first.

The user information processing part is as follows:

- The SPICE-EPG grammar provides information on the phrases from the user's utterance and pointing events.
- Uninteresting phrases like fillers and expletives are filtered out.
- Actor, director and protagonist information is filtered out from the content type concepts and appropriate concepts are created.

- Concepts created by pointing events are filtered out and tagged as SITUATIONALLY EVOKED in the salience-list. The concept is also added to the appropriate type list in the history list.
- For each phrase, except titles and content types, is determined whether it is a reference or not.
- If the concept is not a reference, it is tagged in the salience list and added the appropriate type list in the history list.
- If the phrase is a reference, the form of the reference is determined: pronoun, demonstrative, definite description or one anaphora.
- A list of constraints is created based on the implicit information of the phrase, which is provided by a database.
- A list of constraints is created based on the information of other phrases in the utterance, which provide contextual constraints for the reference. This is provided by a database.
- For each object in the salience list a list of properties is created.
- This list of properties is compared to the list of constraints for compatibility.
- The first compatible object is returned as the referent.
- If no referent is found, concepts which are out of focus, are compared with the reference if the anaphora is of descriptive form (*that movie at ten p.m., the second one from below, the previous list*). Otherwise no referent is returned.
- Type lists from the history list which are not compatible with the reference are filtered out.
- The most recent compatible referent is looked up in the remaining compatible groups.
- The salience and history lists are updated.

At the time the module was designed it was not clear where exactly the module would be placed in the context interpretation module: before the best hypothesis is selected, or after. The best hypothesis is chosen by determining the probability of a phrase based on how well the acoustic data matches with the phrase, on how probable it is that the phrase occurs. To make it possible to process the N-best hypothesis, the salience and history lists are only temporary updated during the user turn. For each hypothesis a backup of the update is made. During the system turn, the system should inform the reference resolution module which hypothesis was chosen as the most probable, so that the update of the best hypothesis is saved and the other updates discarded. The system may penalize hypotheses which have references which could not be resolved, or have referents which do not make sense in the context according to the dialogue manager.

In figure 14. the flow chart for this model is presented.

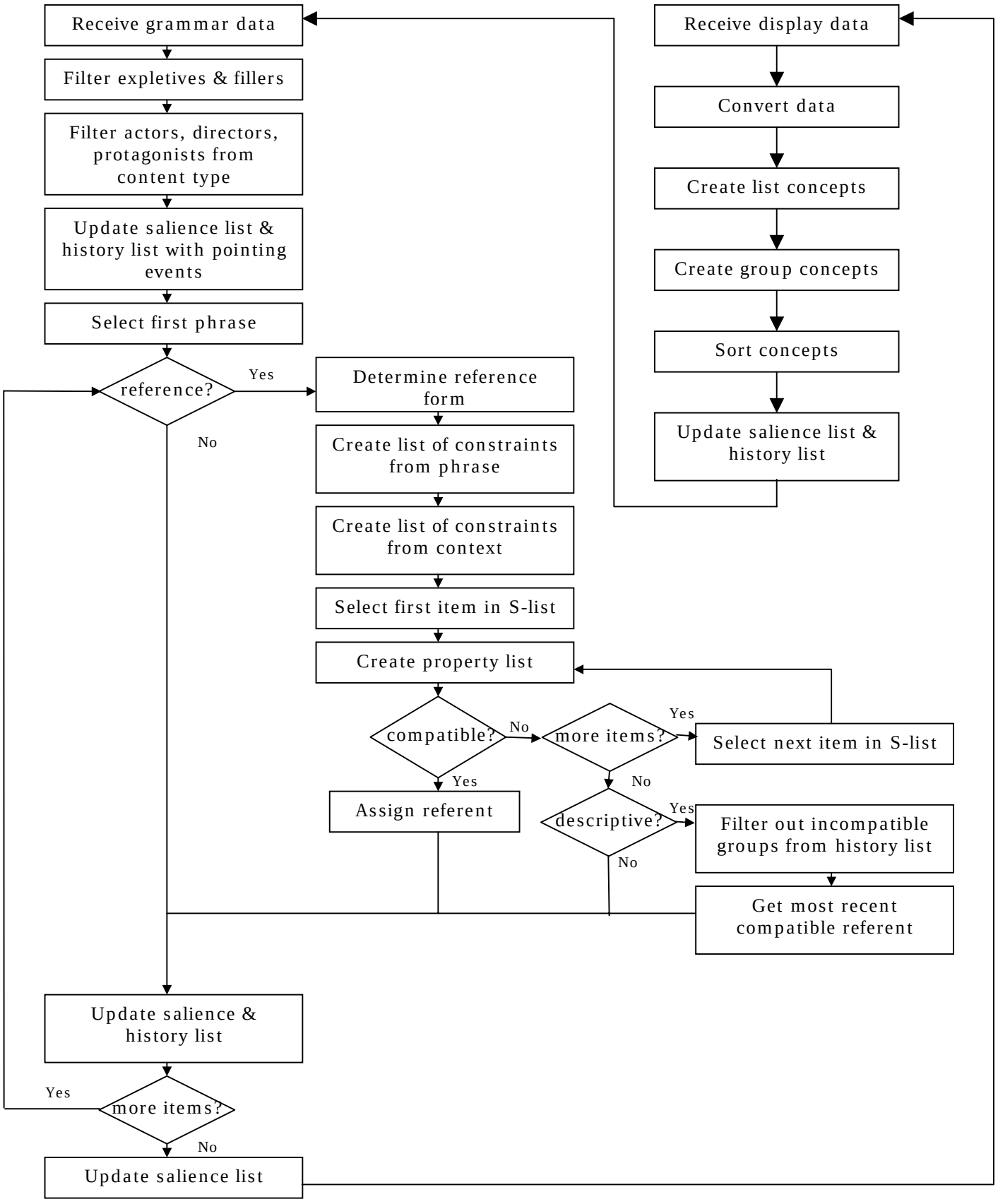


Figure 14. The flowchart of the anaphora resolution module in SPICE.

3.6. System Design

In this section the steps made to design the system are described. In the first subsection an overview is given of the objects, how they relate to each other and how the information flow is between these objects. In the second subsection an overview is given of each of the classes and the tasks they perform.

3.6.1. Defining the objects

In appendix D several examples are used to illustrate the objects needed and the general behavior of them. The following main objects can be defined:

- **concept**, this object is a data structure parallel to the concept produced by the grammar. It is used to store information about the relevant parsed phrases from the user utterance and items from the system display.
- **salience list**, this object is a list of concepts used to determine which concepts are currently in focus. The concepts are ordered according to the algorithm described in [Str98].
- **history list**, this object is a group of lists of concepts which contain all concepts encountered in the discourse which can be referred to. The concepts are grouped by their type and ordered by recency. This list is used to find referents outside the direct focus.
- **constraint list**, this is a list of constraints mapped to the concept values and types. This information is used to match constraints to references and candidate referents for compatibility checks.
- **main interface**, this object is used to read the data produced by the grammar and, convert them to concepts. Filtering of the data is also done here. The data is send to the main engine.
- **display reader**, this object is used to read the data which is displayed on the screen, and convert them to concepts. The data is send to the main interface.
- **grouping module**, this object is used to create lists and groups from the concepts which are displayed on the screen, according to some grouping criteria.
- **main engine**, this object is the part of the program which decides on the course of action given a concept. The main engine receives data from the main interface. The grouping module, deixis filter, reference filter, constraint filter, resolution modules, and update module are triggered by the main engine.
- **deixis filter**, this object is used to filter the concepts which are derived from deictic input (pointing events) out from the list of concepts.
- **reference filter**, this object is used to determine the referential property of a concept.
- **constraints detection module**, this object is used to determine the constraints and properties for the references and candidate referents. In order to do this it uses the constraint list.
- **update module**, this object is used to update the salience list and history list with processed concepts.
- **one anaphora resolution module**, this object is used to resolve one anaphora.
- **demonstrative resolution module**, this object is used to resolve demonstratives.

- **definite description resolution module**, this object is used to resolve definite descriptions. The concept type filter is used to limit the search of possible referents.
- **pronoun resolution module**, this object is used to resolve pronouns.
- **concept type filter**, this object is used to determine which concept types are compatible with a definite description.

Figure 15. shows the objects and their relations.

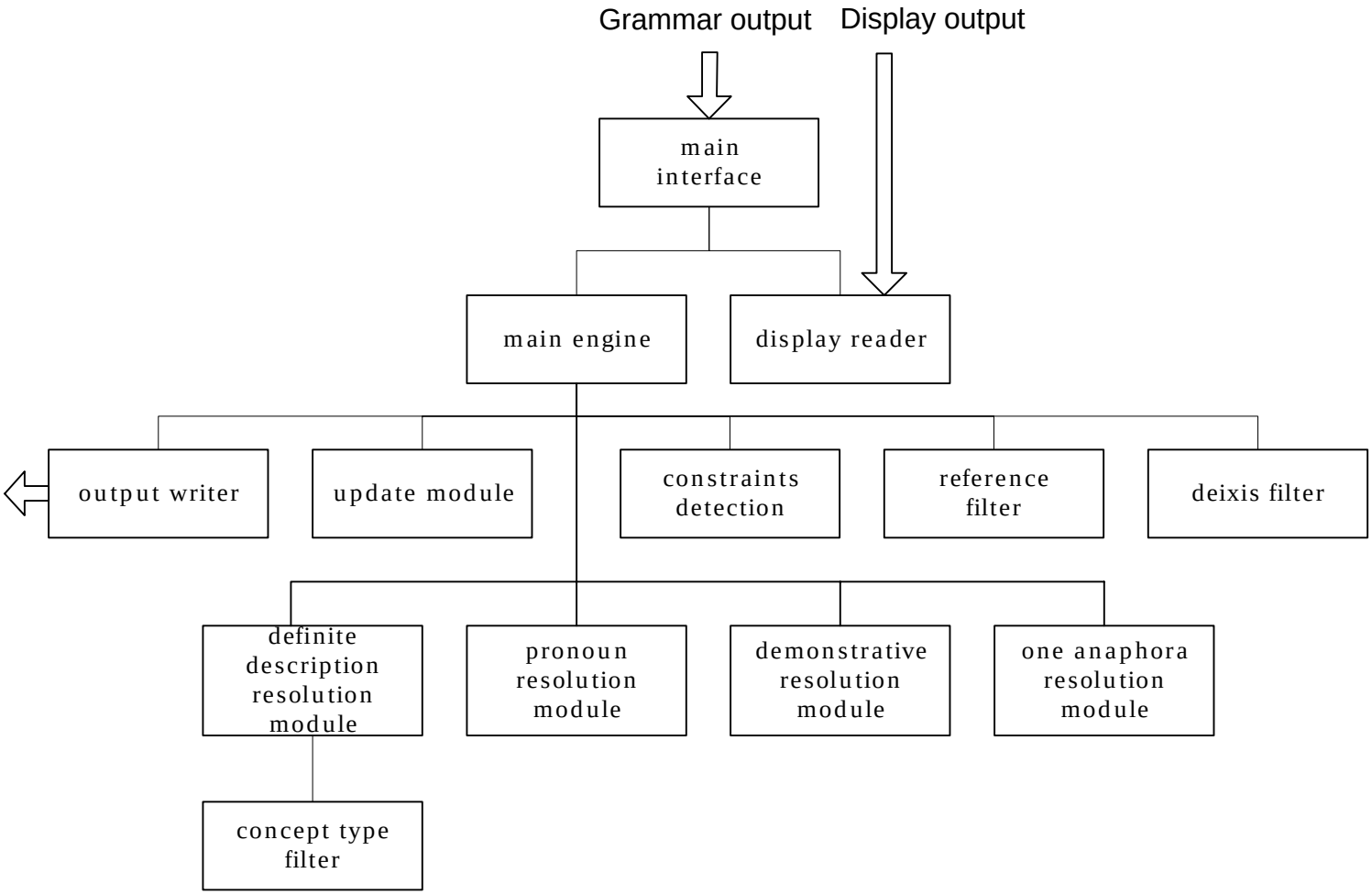


Figure 15. Objects and their relations.

The information flows between the objects are stated in the following subsections for several cases:

- processing display data.
- processing user utterance with a reference to a concept in focus (pronoun).
- processing user utterance with a reference to a concept in focus (demonstrative).
- processing user utterance with a reference to a concept out of focus (definite description).
- processing user utterance with a reference to a concept out of focus (one anaphora).

- processing user utterance with a compound reference (definite description). This is a reference like *the first movie from the previous list*. In this phrase *the previous list* is a reference and *the first movie* is a reference to a property of *the previous list.*
- processing user utterance with a reference to a deictic concept.
- processing user utterance without a reference.

3.6.1.1. processing display data

Figure 16. shows the flow of the data between the objects for the processing of display data.

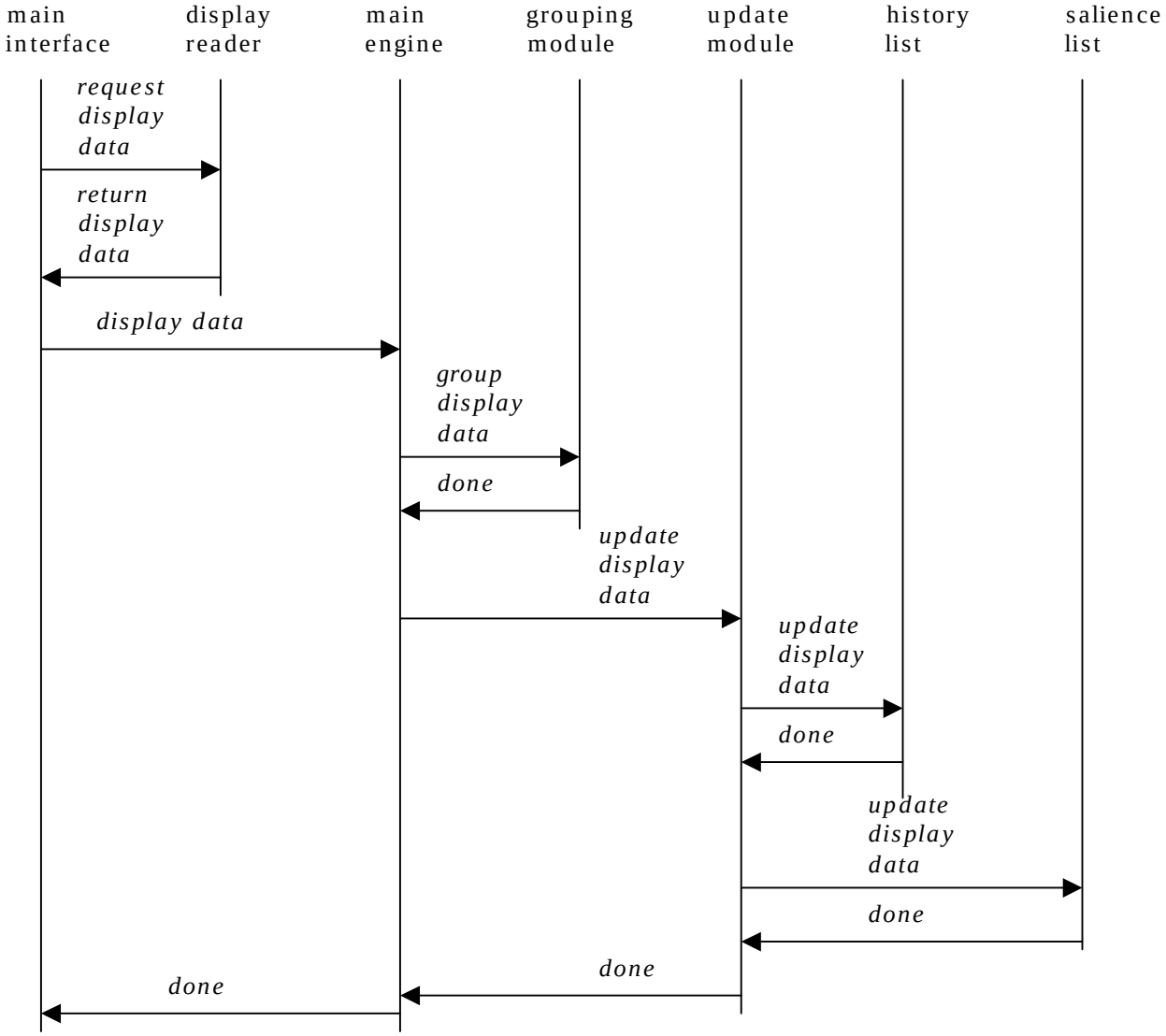


Figure 16. Data flow between objects for the processing of display data.

To process display data the following steps are made:

- main interface requests display data from display reader
- display reader returns display data to display reader

- main interface sends display data to main engine
- main engine requests grouping module to group and add the display data to some lists
- grouping module returns the lists and the grouped data to main engine
- main engine requests update module to update the salience and history list
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- main engine tells main interface it is done

```
updating screen info
   name of list: SELECTION_LIST
determining input :system
handle system input
starting system list processor
System List Processor is ON
start processing
group added: programmes 0
update
adding: list (SELECTION_LIST 0) to list
list (SELECTION_LIST 0) added to concept list
adding: programme (programmes 0) to list
programme (programmes 0) added to concept list
programme (programmes 0) added to slist
list size = 0
programmes 0 put at the end of the list
S-list (1): programmes 0 (deixis),
used size is now: 1
S-list (0):
adding: programme (animal x) to list
programme (animal x) added to concept list
adding: programme (working lunch) to list
programme (working lunch) added to concept list
adding: programme (fortune) to list
programme (fortune) added to concept list
adding: programme (family affairs) to list
programme (family affairs) added to concept list
adding: programme (bewitched) to list
programme (bewitched) added to concept list
adding: programme (real rooms) to list
programme (real rooms) added to concept list
adding: programme (last stand at saber river 1997) to list
programme (last stand at saber river 1997) added to concept list
adding: programme (the front line) to list
programme (the front line) added to concept list
adding: date list (date list 0) to list
date list (date list 0) added to concept list
adding: start time list (start time list 0) to list
start time list (start time list 0) added to concept list
adding: end time list (end time list 0) to list
end time list (end time list 0) added to concept list
adding: channel list (channel list 0) to list
channel list (channel list 0) added to concept list
adding: category list (category list 0) to list
category list (category list 0) added to concept list
S-list (1): programmes 0 (deixis),
slist finalized
lists are updated
```

Figure 17. Sample output from the reference resolution module handling system data.

3.6.1.2. processing user utterance with a reference to a concept in focus (pronoun)

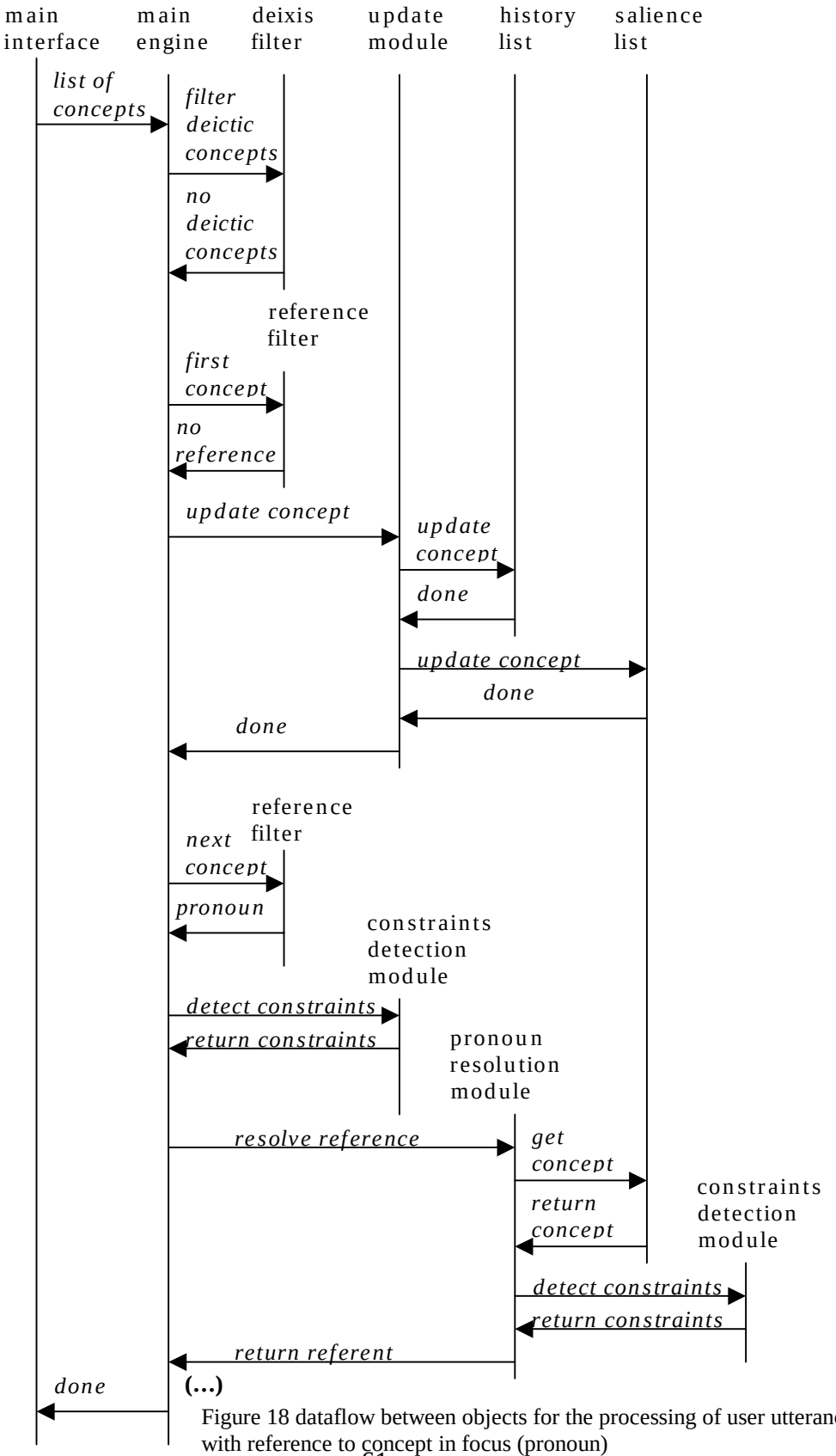


Figure 18 dataflow between objects for the processing of user utterance with reference to concept in focus (pronoun)

Figure 18. shows the flow of the data between the objects for the processing of user utterance with a reference to a concept in focus using a pronoun.
To process user utterance with a reference to a concept in focus (pronoun) the following steps are made:

- main interface sends list of concepts from user utterance to main engine
- main engine requests deixis filter to filter out deictic concepts
- deixis filter returns no deictic concepts
- main engine requests reference filter to determine referential property for the first concept
- reference filter returns no referential property
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- main engine requests reference filter to determine referential property for the next concept.
- reference filter returns referential property is pronoun
- main engine requests constraints detection module to find constraints for concept
- constraints detection module returns constraints
- main engine requests pronoun resolution module to resolve reference
- pronoun resolution module requests salience list for first concept
- salience list returns first concept
- pronoun resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from s-list until compatible referent is found ...)
- pronoun resolution module returns compatible referent
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- (... do the same for all meaningful concepts in the user utterance ...)
- main engine tells main interface it is done

```
CONCEPT:REFERENCE (he)
detect and classify
pronoun detected
looking for constraints within the concept
constraints within the concept as a whole found
no subconcepts to look constraints for
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for REFERENCE (he) :
   constrant: gender (male)
   contraint: number (singular)
   contraint: abstract (no)
end of constraints
constraints found, start resolving pronouns...
look up first compatible entry. size of s-list:2
s-list is at position 0, programmes 1
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
programmes has constraints to add, index = 253
constraint type added: type, programme
constraint type added: person, nonperson
constraint type added: number, plural
constraint type added: abstract, no
no subconcepts to look constraints for
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (programmes 1) :
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: number (plural)
   contraint: abstract (no)
end of constraints
checking for compatibility, size of constraints is 2
constraint type: number
s-list is at position 1, robert redford
looking for constraints within the concept
constraints within the concept as a whole found
no subconcepts to look constraints for
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for actor (robert redford) :
   contraint: type (actor)
   contraint: person (person)
   contraint: number (singular)
   contraint: gender (male)
   contraint: abstract (no)
end of constraints
checking for compatibility, size of constraints is 3
robert redfordis compatible
referent value is: robert redford
temp adding type: actor, value: robert redford
now tagging
can it be tagged as deixis?
does it has a referent?
referent = robert redford
he evoked
S-list (3): robert redford (actor), programmes 1 (deixis), programmes 0 (deixis),
used size is now: 3
S-list (3): robert redford (actor), programmes 1 (deixis), programmes 0 (deixis),
added this to s-list, size is now: 3
```

Figure 19. Sample output from the reference resolution module when handling a pronoun.

3.6.1.3. processing user utterance with a reference to a concept in focus (demonstrative)

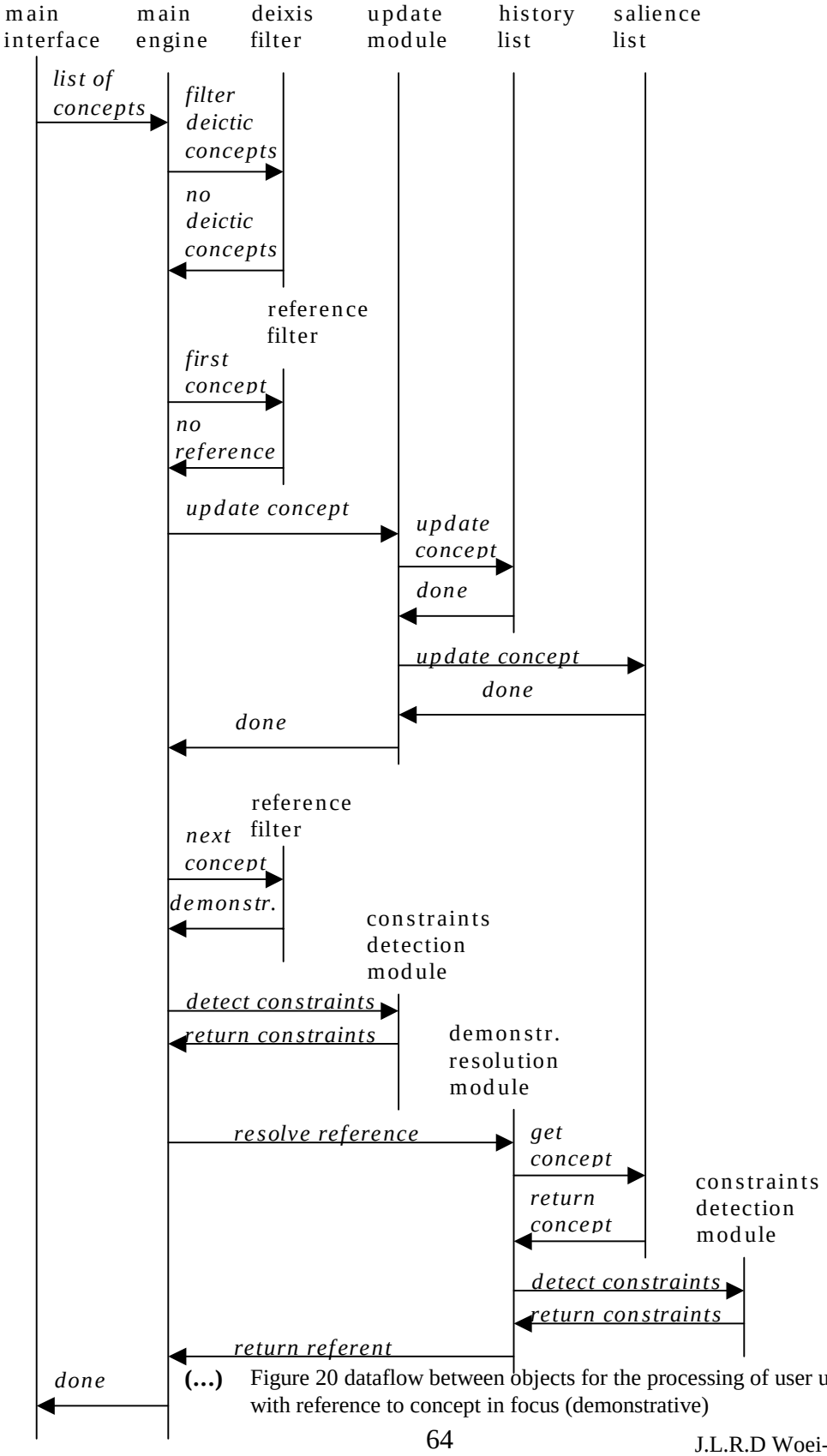


Figure 20 dataflow between objects for the processing of user utterance with reference to concept in focus (demonstrative)

Figure 20. shows the flow of the data between the objects for the processing of user utterance with a reference to a concept in focus using demonstrative.
To process user utterance with a reference to a concept in focus (demonstrative) the following steps are made:

- main interface sends list of concepts from user utterance to main engine
- main engine requests deixis filter to filter out deictic concepts
- deixis filter returns no deictic concepts
- main engine requests reference filter to determine referential property for the first concept.
- reference filter returns no referential property
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- main engine requests reference filter to determine referential property for the next concept.
- reference filter returns referential property is demonstrative
- main engine requests constraints detection module to find constraints for concept
- constraints detection module returns constraints
- main engine requests demonstrative resolution module to resolve reference
- demonstrative resolution module requests salience list for first concept
- salience list returns first concept
- demonstrative resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from s-list until compatible referent is found ...)
- demonstrative resolution module returns compatible referent
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- (... do the same for all meaningful concepts in the user utterance ...)
- main engine tells main interface it is done

```
CONCEPT:DEICTIC (these)
detect and classify
demonstrative detected
looking for constraints within the concept
constraints within the concept as a whole found
no subconcepts to look constraints for
looking for constraints in the concept list
working on concept: record
done checking each of the premisses
premisses hold
  adding constraint type: type, programme
the following constraints were determined for DEICTIC (these) :
   contraint: number (plural)
   contraint: abstract (no)
   contraint: type(programme)
end of constraints
constraints found, start resolving pronouns...
look up first compatible entry. size of s-list:2
s-list is at position 0, programmes 4
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
programmes has constraints to add, index = 253
constraint type added: type, programme
constraint type added: person, nonperson
constraint type added: number, plural
constraint type added: abstract, no
no subconcepts to look constraints for
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (programmes 4) :
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: number (plural)
   contraint: abstract (no)
end of constraints
checking for compatibility, size of constraints is 3
programmes 4is compatible
referent value is: programmes 4
temp adding type: programme, value: programmes 4
now tagging
can it be tagged as deixis?
does it has a referent?
referent = programmes 4
these evoked
S-list (2): programmes 4 (evoked), programmes 3 (deixis),
used size is now: 2
S-list (2): programmes 4 (evoked), programmes 3 (deixis),
added this to s-list, size is now: 2
```

Figure 21. Sample output from the reference resolution module when handling a demonstrative.

3.6.1.4. processing user utterance with a reference to a concept out of focus (definite description)

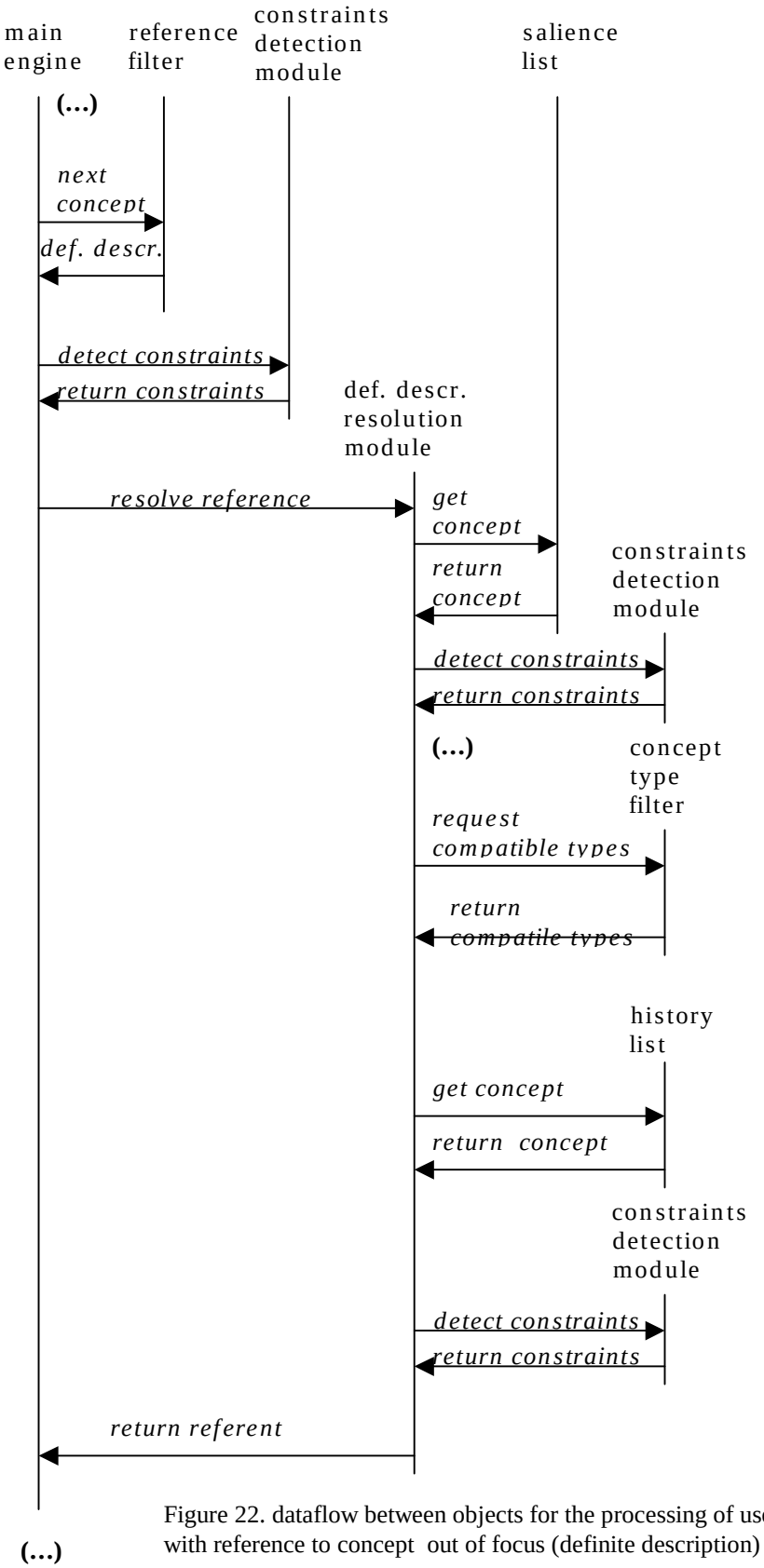


Figure 22. dataflow between objects for the processing of user utterance with reference to concept out of focus (definite description)

Figure 22. shows the flow of the data between the objects for the processing of user utterance with a reference to a concept out of focus using a definite description.
To process user utterance with a reference to a concept out of focus (definite description) the following steps are made:

- main interface sends list of concepts from user utterance to main engine
- main engine requests deixis filter to filter out deictic concepts
- deixis filter returns no deictic concepts
- main engine requests reference filter to determine referential property for the first concept.
- reference filter returns no referential property
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- main engine requests reference filter to determine referential property for the next concept.
- reference filter returns referential property is definite description
- main engine requests constraints detection module to find constraints for concept
- constraints detection module returns constraints
- main engine requests definite description resolution module to resolve reference
- definite description resolution module requests salience list for first concept
- salience list returns first concept
- definite description resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from s-list ...)
- definite description resolution module requests concept type filter for a list with compatible concept types
- concept type filter returns a list with compatible concept types
- definite description resolution module request history list for most recent concept of compatible concept type
- definite description resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from history list until referent is found ...)
- definite description resolution module returns most recent compatible referent
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- (... do the same for all meaningful concepts in the user utterance ...)
- main engine tells main interface it is done

```
CONCEPT:DEFINITE_DESCRIPTION (the second programme from below)
detect and classify
definite description detected
looking for constraints within the concept
constraints within the concept as a whole found
no subconcepts to look constraints for
looking for constraints in the concept list
working on concept: record
done checking each of the premisses
premisses hold
  adding constraint type: type, programme
the following constraints were determined for DEFINITE_DESCRIPTION (the second programme
from below) :
   contraint: listentry (-2)
   contraint: number (singular)
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: type (programme)
end of constraints
constraints found, start resolving definite description...
start determining concept types
referent is a list entry
concept types determined: list.listentries
looking at the listentries of : list
checking for compatibility of: football
isCompatible: detect constraints for candidate referent :football
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
no constraints found in concept value, search in concept type
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (football) :
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (18:10)
   contraint: end time (20:25)
   contraint: channel (Channel_5)
   contraint: start time (18:10)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (18:10) added
  constraint: end time (20:25) added
  constraint: channel (Channel_5) added
  constraint: start time (18:10) added
done checking constraints, constraint size = 6
football is compatible, now checking for position
checking for compatibility of: football
isCompatible: detect constraints for candidate referent :football
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
no constraints found in concept value, search in concept type
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (football) :
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (20:25)
   contraint: end time (23:00)
   contraint: channel (Channel_5)
```

```
   contraint: start time (20:25)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (20:25) added
  constraint: end time (23:00) added
  constraint: channel (Channel_5) added
  constraint: start time (20:25) added
done checking constraints, constraint size = 6
football is compatible, now checking for position
checking for compatibility of: world sport
isCompatible: detect constraints for candidate referent :world sport
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
sport has constraints to add, index = 244
constraint type added: number, singular
constraint type added: category, sport
constraint type added: type, programme
constraint type added: person, nonperson
constraint type added: abstract, no
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (world sport) :
   contraint: number (singular)
   contraint: category (sport)
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (23:30)
   contraint: end time (0:00)
   contraint: channel (CNN)
   contraint: start time (23:30)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (23:30) added
  constraint: end time (0:00) added
  constraint: channel (CNN) added
  constraint: start time (23:30) added
done checking constraints, constraint size = 6
world sport is compatible, now checking for position
checking for compatibility of: international match of the day
isCompatible: detect constraints for candidate referent :international match of the day
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
day has constraints to add, index = 321
constraint type added: type, date
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (international match of the day)
:
   contraint: type (date)
   contraint: date (11-09-2000)
   contraint: start time (23:50)
   contraint: end time (0:45)
   contraint: channel (BBC1)
   contraint: start time (23:50)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (23:50) added
  constraint: end time (0:45) added
  constraint: channel (BBC1) added
  constraint: start time (23:50) added
done checking constraints, constraint size = 6
not compatible because: programme!=date
```

```
checking for compatibility of: world sport
checking for compatibility of: football
footballis compatible
referent value is: football
temp adding type: programme, value: football
now tagging
can it be tagged as deixis?
does it has a referent?
referent = football
referent not found in the list
the second programme from below evoked
list size = 2
football put at position0
S-list (3): football (evoked), programmes 18 (deixis), programmes 17 (deixis),
used size is now: 3
S-list (3): football (evoked), programmes 18 (deixis), programmes 17 (deixis),
added the second programme from below to s-list, size is now: 3
finalize temp
hislist finalized
type history list finalized
usedSize = 3
tempList size = 0
tempList size after update = 1
size of last entry in tempList = 2
S-list (2): programmes 18 (deixis), programmes 17 (deixis),
slist temp finalized
```

Figure 23. Sample output from the reference resolution module when handling a definite description.

3.6.1.5. processing user utterance with a reference to a concept out of focus (one anaphora)

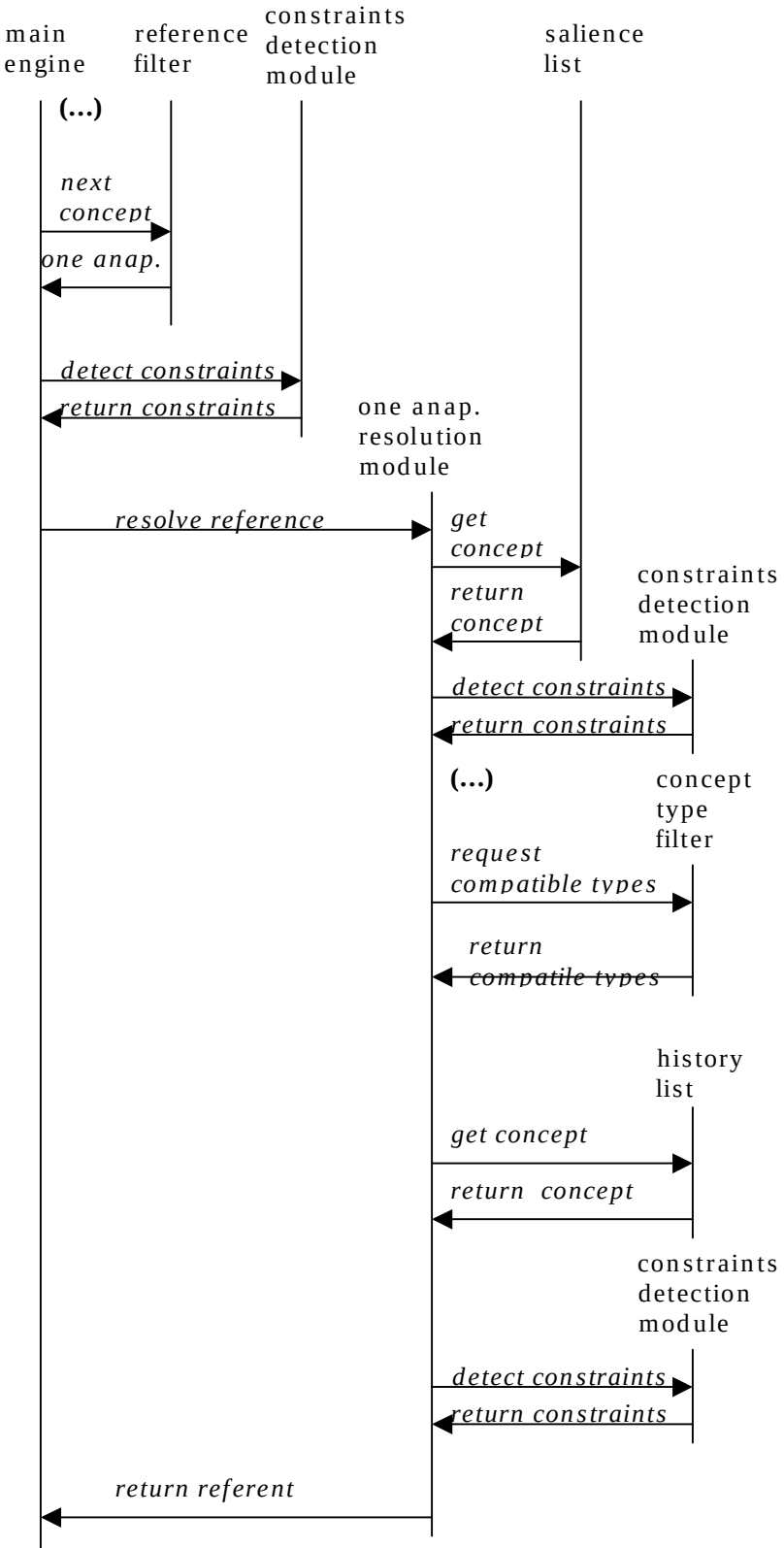


Figure 24. dataflow between objects for the processing of user utterance with reference to concept out of focus (one anaphora)

Figure 24. shows the flow of the data between the objects for the processing of user utterance with a reference to a concept out of focus using one anaphora.
To process user utterance with a reference to a concept out of focus (one anaphora) the following steps are made:

- main interface sends list of concepts from user utterance to main engine
- main engine requests deixis filter to filter out deictic concepts
- deixis filter returns no deictic concepts
- main engine requests reference filter to determine referential property for the first concept.
- reference filter returns no referential property
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- main engine requests reference filter to determine referential property for the next concept.
- reference filter returns referential property is one anaphora
- main engine requests constraints detection module to find constraints for concept
- constraints detection module returns constraints
- main engine requests one anaphora resolution module to resolve reference
- one anaphora resolution module requests salience list for first concept
- salience list returns first concept
- one anaphora resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from s-list ...)
- one anaphora resolution module requests concept type filter for a list with compatible concept types
- concept type filter returns a list with compatible concept types
- one anaphora resolution module request history list for most recent concept of compatible concept type
- one anaphora resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from history list until referent is found ...)
- one anaphora resolution module returns most recent compatible referent
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- (... do the same for all meaningful concepts in the user utterance ...)
- main engine tells main interface it is done

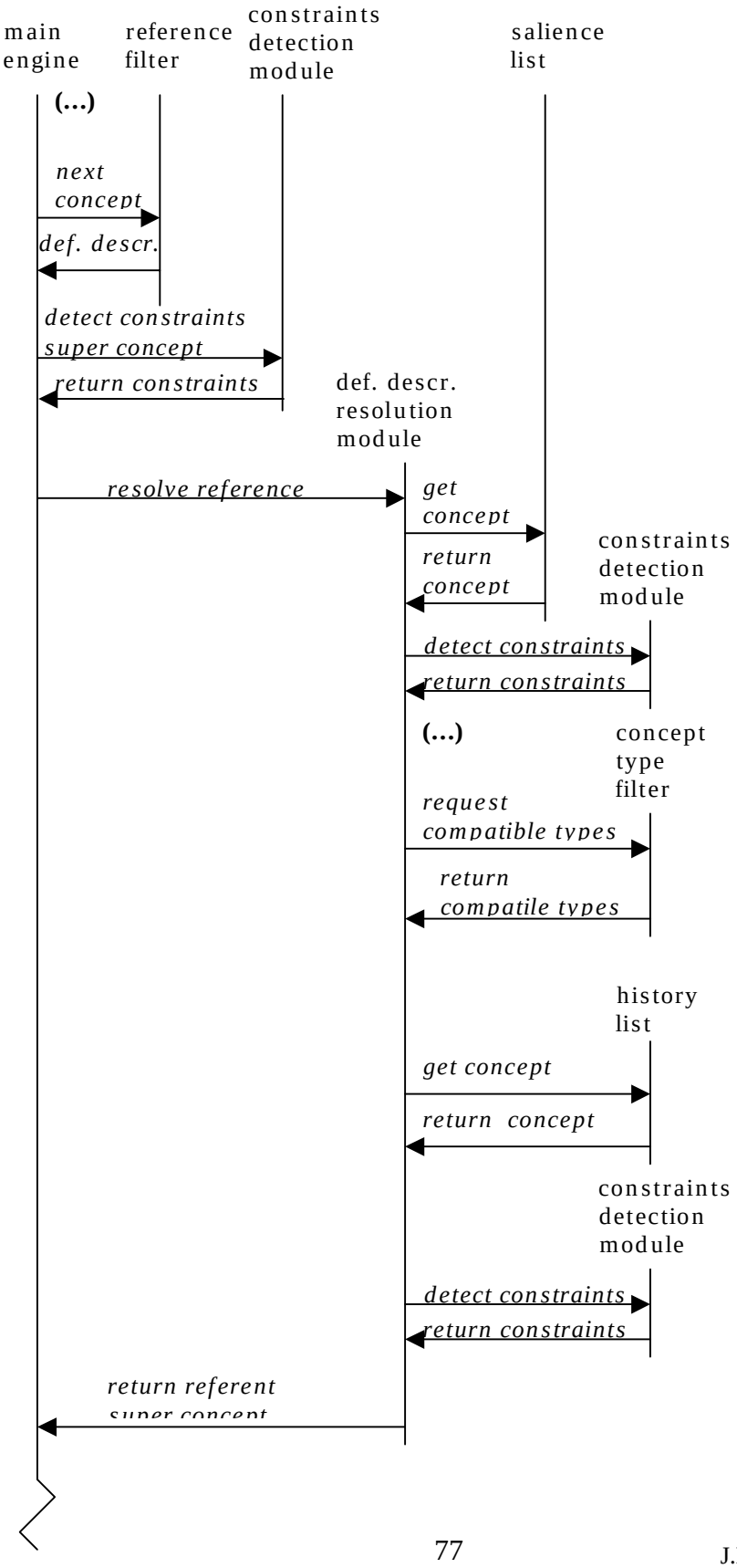
```
CONCEPT:DEFINITE_DESCRIPTION (the second one from below)
detect and classify
one anaphora detected
looking for constraints within the concept
constraints within the concept as a whole found
no subconcepts to look constraints for
looking for constraints in the concept list
working on concept: record
done checking each of the premisses
premisses hold
  adding constraint type: type, programme
the following constraints were determined for DEFINITE_DESCRIPTION (the second one from
below) :
   contraint: listentry (-2)
   contraint: number (singular)
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: type (programme)
end of constraints
constraints found, start resolving definite description...
start determining concept types
referent is a list entry
concept types determined: list.listentries
looking at the listentries of : list
checking for compatibility of: football
isCompatible: detect constraints for candidate referent :football
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
no constraints found in concept value, search in concept type
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (football) :
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (18:10)
   contraint: end time (20:25)
   contraint: channel (Channel_5)
   contraint: start time (18:10)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (18:10) added
  constraint: end time (20:25) added
  constraint: channel (Channel_5) added
  constraint: start time (18:10) added
done checking constraints, constraint size = 6
football is compatible, now checking for position
checking for compatibility of: football
isCompatible: detect constraints for candidate referent :football
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
no constraints found in concept value, search in concept type
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (football) :
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (20:25)
   contraint: end time (23:00)
   contraint: channel (Channel_5)
```

```
   contraint: start time (20:25)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (20:25) added
  constraint: end time (23:00) added
  constraint: channel (Channel_5) added
  constraint: start time (20:25) added
done checking constraints, constraint size = 6
football is compatible, now checking for position
checking for compatibility of: world sport
isCompatible: detect constraints for candidate referent :world sport
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
sport has constraints to add, index = 244
constraint type added: number, singular
constraint type added: category, sport
constraint type added: type, programme
constraint type added: person, nonperson
constraint type added: abstract, no
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (world sport) :
   contraint: number (singular)
   contraint: category (sport)
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (23:30)
   contraint: end time (0:00)
   contraint: channel (CNN)
   contraint: start time (23:30)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (23:30) added
  constraint: end time (0:00) added
  constraint: channel (CNN) added
  constraint: start time (23:30) added
done checking constraints, constraint size = 6
world sport is compatible, now checking for position
checking for compatibility of: international match of the day
isCompatible: detect constraints for candidate referent :international match of the day
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
day has constraints to add, index = 321
constraint type added: type, date
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (international match of the day)
:
   contraint: type (date)
   contraint: date (11-09-2000)
   contraint: start time (23:50)
   contraint: end time (0:45)
   contraint: channel (BBC1)
   contraint: start time (23:50)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (23:50) added
  constraint: end time (0:45) added
  constraint: channel (BBC1) added
  constraint: start time (23:50) added
done checking constraints, constraint size = 6
not compatible because: programme!=date
```

```
checking for compatibility of: world sport
checking for compatibility of: football
footballis compatible
referent value is: football
temp adding type: programme, value: football
now tagging
can it be tagged as deixis?
does it has a referent?
referent = football
referent not found in the list
the second one from below evoked
list size = 2
football put at position0
S-list (3): football (evoked), programmes 18 (deixis), programmes 17 (deixis),
used size is now: 3
S-list (3): football (evoked), programmes 18 (deixis), programmes 17 (deixis),
added the second one from below to s-list, size is now: 3
finalize temp
hislist finalized
type history list finalized
usedSize = 3
tempList size = 0
tempList size after update = 1
size of last entry in tempList = 2
S-list (2): programmes 18 (deixis), programmes 17 (deixis),
slist temp finalized
```

Figure 25. Sample output from the reference resolution module when handling one anaphora.

3.6.1.6. processing user utterance with a compound reference (definite description)



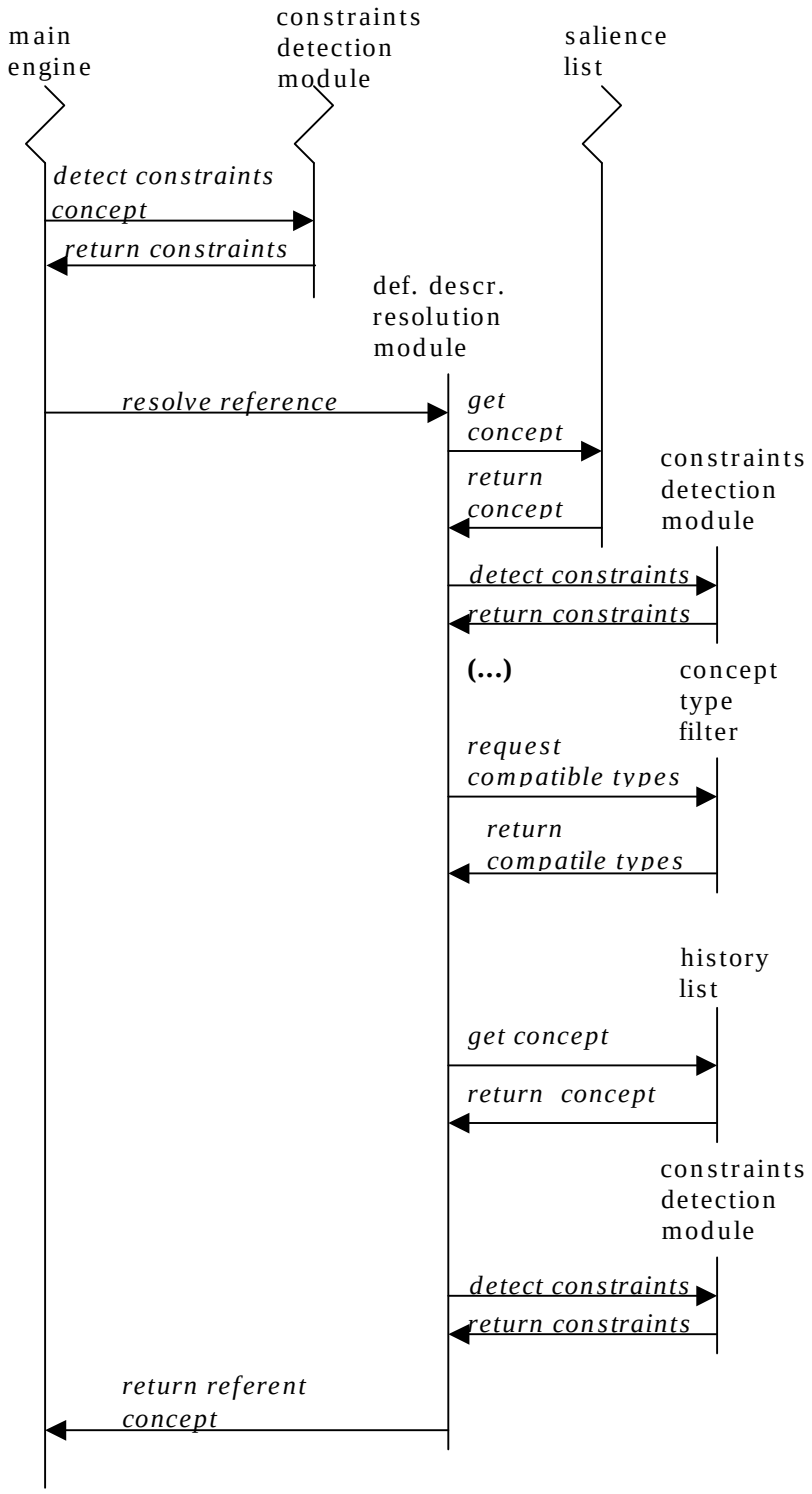


Figure 26. dataflow between objects for the processing of user utterance with a compound reference.

Figure 26. shows the flow of the data between the objects for the processing of user utterance with a compound reference.
To process user utterance with a compound reference (definite description) (This is a reference like *the first movie from the previous list*. In this phrase *the previous list* is a reference and *the first movie* is a reference to a property of *the previous list*, which will be called here the superconcept*)* the following steps are made:

- main interface sends list of concepts from user utterance to main engine
- main engine requests deixis filter to filter out deictic concepts
- deixis filter returns no deictic concepts
- main engine requests reference filter to determine referential property for the first concept.
- reference filter returns no referential property
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- main engine requests reference filter to determine referential property for the next concept.
- reference filter returns referential property is definite description
- main engine requests constraints detection module to find constraints for superconcept
- constraints detection module returns constraints
- main engine requests definite description resolution module to resolve reference
- definite description resolution module requests salience list for first concept
- salience list returns first concept
- definite description resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from s-list ...)
- definite description resolution module requests concept type filter for a list with compatible concept types
- concept type filter returns a list with compatible concept types
- definite description resolution module request history list for most recent concept of compatible concept type
- definite description resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from history list until referent is found ...)
- definite description resolution module returns most recent compatible referent
- main engine requests update module to update history and salience list with superconcept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done

- update module tells main engine it is done
- main engine requests constraints detection module to find constraints for concept with superconcept
- constraints detection module returns constraints
- main engine requests definite description resolution module to resolve reference
- definite description resolution module requests salience list for first concept
- salience list returns first concept
- definite description resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from s-list ...)
- definite description resolution module requests concept type filter for a list with compatible concept types
- concept type filter returns a list with compatible concept types
- definite description resolution module request history list for most recent concept of compatible concept type
- definite description resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from history list until referent is found ...)
- definite description resolution module returns most recent compatible referent
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- (... do the same for all meaningful concepts in the user utterance ...)
- main engine tells main interface it is done

```
CONCEPT:DEFINITE_DESCRIPTION (the previous list)
detect and classify
definite description detected
looking for constraints within the concept
constraints within the concept as a whole found
no subconcepts to look constraints for
looking for constraints in the concept list
working on concept: record
done checking each of the premisses
premisses do not hold
  adding constraint type: type, programme
the following constraints were determined for DEFINITE_DESCRIPTION (the previous list) :
   contraint: number, singular, 1
   contraint: person, nonperson, 1
   contraint: type, list, 1
   contraint: abstract, no, 1
   contraint: recency, -1, 1
end of constraints
constraints found, start resolving definite description...
start determining concept types
concept types determined: list
checking for compatibility of: SELECTION_LIST 78
isCompatible: detect constraints for candidate referent :SELECTION_LIST 78
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
no constraints found in concept value, search in concept type
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for list (SELECTION_LIST 78) :
   contraint: type (list)
   contraint: person (nonperson)
   contraint: abstract (no)
end of constraints
done checking constraints, constraint size = 3
SELECTION_LIST 78 is compatible, now checking for recency
checking for compatibility of: SELECTION_LIST 77
isCompatible: detect constraints for candidate referent :SELECTION_LIST 77
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
no constraints found in concept value, search in concept type
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for list (SELECTION_LIST 77) :
   contraint: type (list)
   contraint: person (nonperson)
   contraint: abstract (no)
end of constraints
done checking constraints, constraint size = 3
SELECTION_LIST 77 is compatible, now checking for position
referent value is: SELECTION_LIST 77
temp adding type: list, value: SELECTION_LIST 77
now tagging
can it be tagged as deixis?
does it has a referent?
referent = SELECTION_LIST 77
referent not found in the list
SELECTION_LIST 77 evoked
list size = 2
SELECTION_LIST 77 put at position0
S-list (3): SELECTION_LIST 77 (evoked), programmes 18 (deixis), programmes 17 (deixis),
used size is now: 3
S-list (3): SELECTION_LIST 77 (evoked), programmes 18 (deixis), programmes 17 (deixis),
added the second one from below to s-list, size is now: 3
CONCEPT:DEFINITE_DESCRIPTION (the second one from below)
detect and classify
```

```
one anaphora detected
looking for constraints within the concept
constraints within the concept as a whole found
no subconcepts to look constraints for
looking for constraints in the concept list
working on concept: record
done checking each of the premisses
premisses hold
  adding constraint type: type, programme
the following constraints were determined for DEFINITE_DESCRIPTION (the second one from
below) :
   contraint: listentry (-2)
   contraint: number (singular)
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: type (programme)
end of constraints
constraints found, start resolving definite description...
start determining concept types
referent is a list entry
concept types determined: SELECTION_LIST 77.listentries
looking at the listentries of : SELECTION_LIST 77
checking for compatibility of: football
isCompatible: detect constraints for candidate referent :football
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
no constraints found in concept value, search in concept type
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (football) :
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (18:10)
   contraint: end time (20:25)
   contraint: channel (Channel_5)
   contraint: start time (18:10)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (18:10) added
  constraint: end time (20:25) added
  constraint: channel (Channel_5) added
  constraint: start time (18:10) added
done checking constraints, constraint size = 6
football is compatible, now checking for position
checking for compatibility of: football
isCompatible: detect constraints for candidate referent :football
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
no constraints found in concept value, search in concept type
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (football) :
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (20:25)
   contraint: end time (23:00)
   contraint: channel (Channel_5)
   contraint: start time (20:25)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
```

```
  constraint: start time (20:25) added
  constraint: end time (23:00) added
  constraint: channel (Channel_5) added
  constraint: start time (20:25) added
done checking constraints, constraint size = 6
football is compatible, now checking for position
checking for compatibility of: world sport
isCompatible: detect constraints for candidate referent :world sport
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
sport has constraints to add, index = 244
constraint type added: number, singular
constraint type added: category, sport
constraint type added: type, programme
constraint type added: person, nonperson
constraint type added: abstract, no
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (world sport) :
   contraint: number (singular)
   contraint: category (sport)
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (23:30)
   contraint: end time (0:00)
   contraint: channel (CNN)
   contraint: start time (23:30)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (23:30) added
  constraint: end time (0:00) added
  constraint: channel (CNN) added
  constraint: start time (23:30) added
done checking constraints, constraint size = 6
world sport is compatible, now checking for position
checking for compatibility of: international match of the day
isCompatible: detect constraints for candidate referent :international match of the day
looking for constraints within the concept
constraints within the concept as a whole not found
look for each word in the string for constraints
day has constraints to add, index = 321
constraint type added: type, date
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (international match of the day)
:
   contraint: type (date)
   contraint: date (11-09-2000)
   contraint: start time (23:50)
   contraint: end time (0:45)
   contraint: channel (BBC1)
   contraint: start time (23:50)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (23:50) added
  constraint: end time (0:45) added
  constraint: channel (BBC1) added
  constraint: start time (23:50) added
done checking constraints, constraint size = 6
not compatible because: programme!=date
checking for compatibility of: world sport
checking for compatibility of: football
footballis compatible
referent value is: football
```

```
temp adding type: programme, value: football
now tagging
can it be tagged as deixis?
does it has a referent?
referent = football
referent not found in the list
the second one from below evoked
list size = 3
football put at position0
S-list (4): football (evoked), SELECTION_LIST 77 (evoked), programmes 18 (deixis),
programmes 17 (deixis),
used size is now: 4
S-list (4): football (evoked), SELECTION_LIST 77 (evoked), programmes 18 (deixis),
programmes 17 (deixis),
added the second one from below to s-list, size is now: 4
```

Figure 27. Sample output from the reference resolution module when handling a compound reference

3.6.1.7. processing user utterance with a reference to a deictic concept

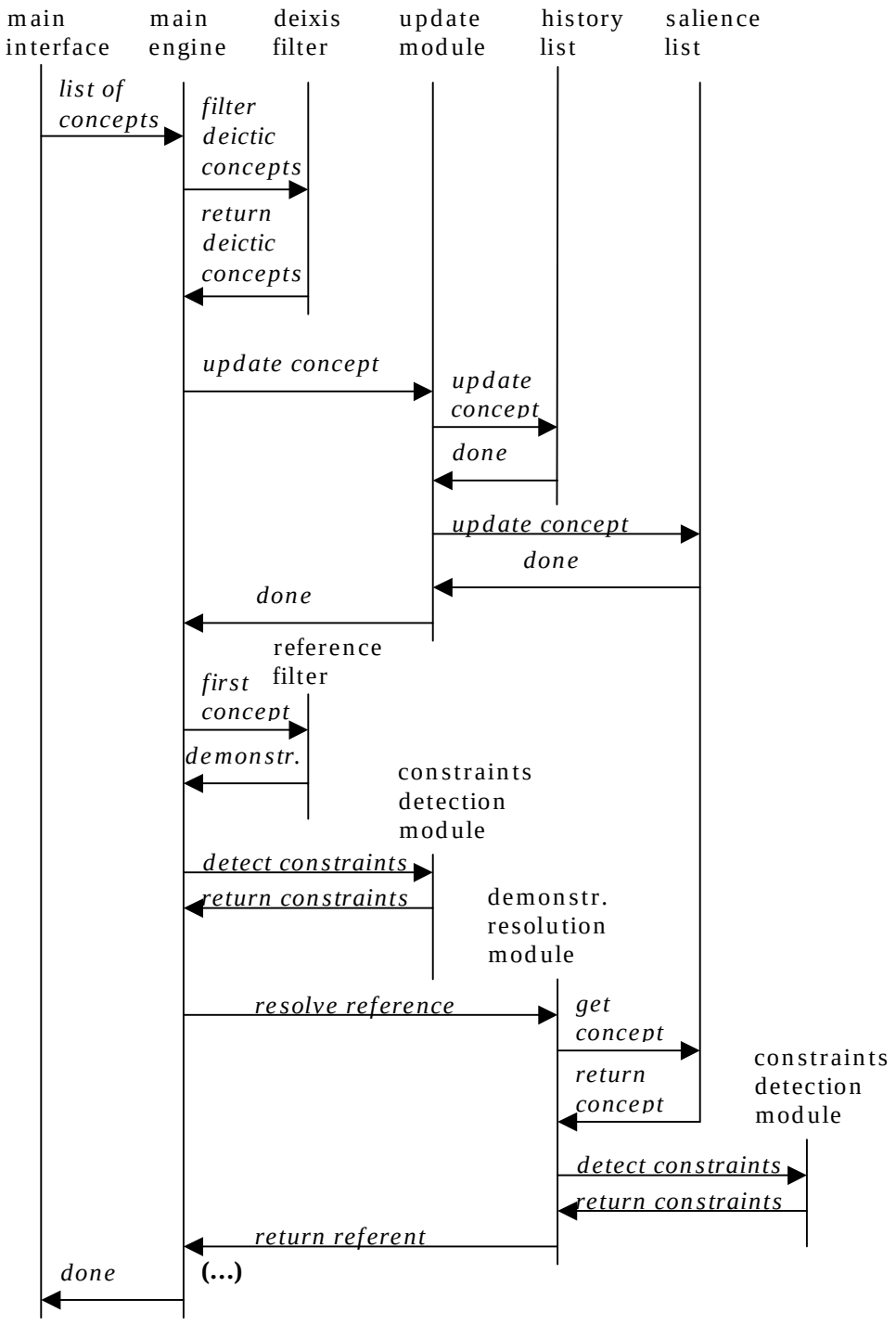


Figure 28. dataflow between objects for the processing of user utterance with a reference to a deictic concept.

Figure 28. shows the flow of the data between the objects for the processing of user utterance with a reference to a deictic concept.
To process user utterance with a reference to a deictic concept the following steps are made:

- main interface sends list of concepts from user utterance to main engine
- main engine requests deixis filter to filter out deictic concepts
- deixis filter returns the deictic concept
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- main engine requests reference filter to determine referential property for the first concept.
- reference filter returns referential property is demonstrative
- main engine requests constraints detection module to find constraints for concept
- constraints detection module returns constraints
- main engine requests demonstrative resolution module to resolve reference
- demonstrative resolution module requests salience list for first concept
- salience list returns first concept
- demonstrative resolution module requests constraint detection module to find constraints for concept
- (... repeat looking for compatible concepts from s-list ...)
- demonstrative resolution module returns compatible referent
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done
- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- (... do the same for all meaningful concepts in the user utterance ...)
- main engine tells main interface it is done

```
detect deixis, size = 2
deixis detected: football
deixis detected: 11-09-2000
deixis detected: 18:10
deixis detected: 20:25
deixis detected: channel5
deixis present, updated
temp adding type: programme, value: football
now tagging
can it be tagged as deixis?
football deixis
list size = 2
football put at position0
S-list (3): football (deixis), programmes 4 (deixis), programmes 3 (deixis),
used size is now: 2
temp adding type: date, value: 11-09-2000
now tagging
can it be tagged as deixis?
11-09-2000 deixis
list size = 3
11-09-2000 put at position1
S-list (4): football (deixis), 11-09-2000 (deixis), programmes 4 (deixis), programmes 3
(deixis),
used size is now: 3
temp adding type: start time, value: 18:10
now tagging
can it be tagged as deixis?
18:10 deixis
list size = 4
18:10 put at position2
S-list (5): football (deixis), 11-09-2000 (deixis), 18:10 (deixis), programmes 4
(deixis), programmes 3 (deixis),
used size is now: 4
temp adding type: end time, value: 20:25
now tagging
can it be tagged as deixis?
20:25 deixis
list size = 5
20:25 put at position3
S-list (6): football (deixis), 11-09-2000 (deixis), 18:10 (deixis), 20:25 (deixis),
programmes 4 (deixis), programmes 3 (deixis),
used size is now: 5
temp adding type: channel, value: channel5
now tagging
can it be tagged as deixis?
channel5 deixis
list size = 6
channel5 put at position4
S-list (7): football (deixis), 11-09-2000 (deixis), 18:10 (deixis), 20:25 (deixis),
channel5 (deixis), programmes 4 (deixis), programmes 3 (deixis),
used size is now: 6

CONCEPT:DEICTIC (this)
detect and classify
demonstrative detected
looking for constraints within the concept
constraints within the concept as a whole found
no subconcepts to look constraints for
looking for constraints in the concept list
working on concept: record
done checking each of the premisses
premisses hold
  adding constraint type: type, programme
the following constraints were determined for DEICTIC (this) :
   contraint: number (singular)
   contraint: abstract (no)
   contraint: type(programme)
end of constraints
constraints found, start resolving pronouns...
```

```
look up first compatible entry. size of s-list:7
s-list is at position 0, football
looking for constraints within the concept
checking for compatibility of: football
constraints within the concept as a whole not found
look for each word in the string for constraints
no constraints found in concept value, search in concept type
looking for constraints in the subconcept list
looking for constraints in the concept list
constraints in the concept list not found...
the following constraints were determined for programme (football) :
   contraint: type (programme)
   contraint: person (nonperson)
   contraint: abstract (no)
   contraint: date (11-09-2000)
   contraint: start time (18:10)
   contraint: end time (20:25)
   contraint: channel (Channel_5)
   contraint: start time (18:10)
end of constraints
subconcepts detected
  constraint: date (11-09-2000) added
  constraint: start time (18:10) added
  constraint: end time (20:25) added
  constraint: channel (Channel_5) added
  constraint: start time (18:10) added
done checking constraints, constraint size = 6
checking for compatibility, size of constraints is 3
footballis compatible
referent value is: football
temp adding type: programme, value: football
now tagging
can it be tagged as deixis?
does it has a referent?
referent = football
this evoked
S-list (7): football (deixis), 11-09-2000 (deixis), 18:10 (deixis), 20:25 (deixis),
channel5 (deixis), programmes 4 (deixis), programmes 3 (deixis),
used size is now: 7
S-list (7): football (evoked), 11-09-2000 (deixis), 18:10 (deixis), 20:25 (deixis),
channel5 (deixis), programmes 4 (deixis), programmes 3 (deixis),
added this to s-list, size is now: 7
```

Figure 29. Sample output from the reference resolution module when handling a deictic reference.

3.6.1.8. Processing user utterance without a reference

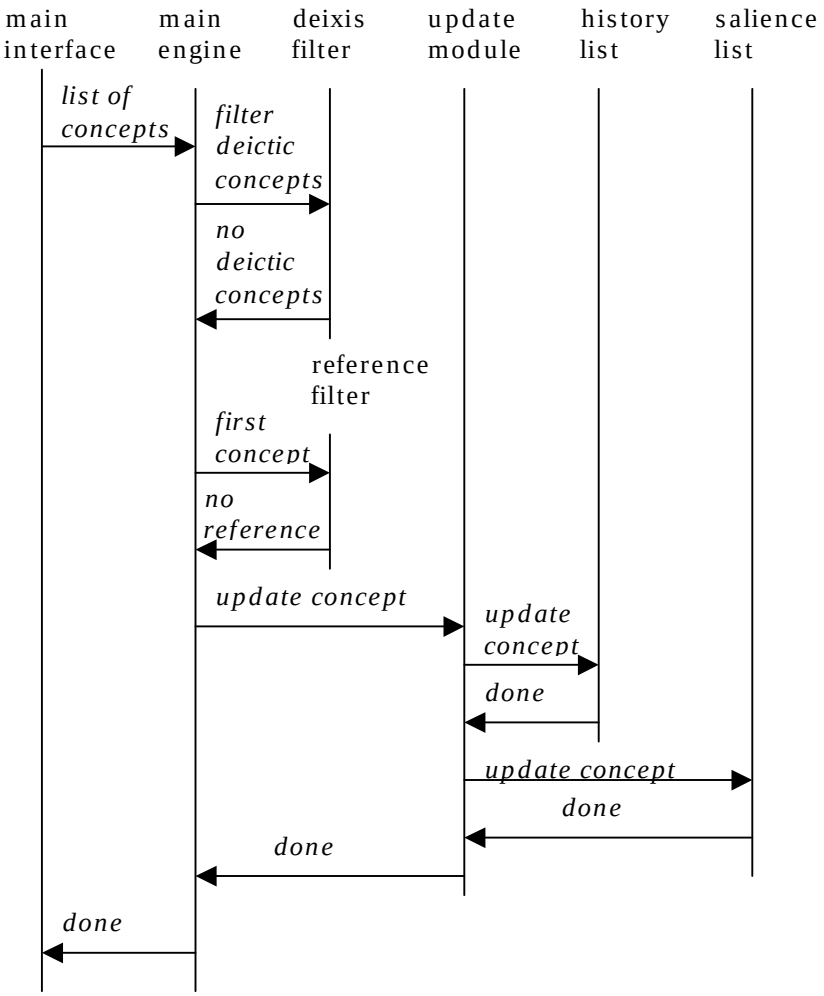


Figure 30. dataflow between objects for the processing of user utterance without a reference.

Figure 30. shows the flow of the data between the objects for the processing of user utterance with a reference to a concept in focus using a pronoun.
To process user utterance with a reference to a concept in focus (pronoun) the following steps are made:

- main interface sends list of concepts from user utterance to main engine
- main engine requests deixis filter to filter out deictic concepts
- deixis filter returns no deictic concepts
- main engine requests reference filter to determine referential property for the first concept
- reference filter returns no referential property
- main engine requests update module to update history and salience list with concept
- update module requests the history list to add data
- history list tells update module it is done

- update module requests the salience list to add data
- salience list tells update module it is done
- update module tells main engine it is done
- (... do the same for all meaningful concepts in the user utterance ...)
- main engine tells main interface it is done

```
reading conceptgraph
      BEGIN_LATTICE
done filtering noise
1 112 @contents 514.3000 1 112
first concept read
concept type 'contents' read
done filtering noise
text robert redford
tag is: text
concept value: robert redford
END_LATTICE
starting main engine
determining input :user
handle user input
increasing sentence number
removing not used entities from list, new size will be: 1
S-list (1): programmes 0 (deixis),
set next sentence
detect deixis, size = 1

CONCEPT:actor (robert redford)
detect and classify
no referential property detected
temp adding type: actor, value: robert redford
now tagging
can it be tagged as deixis?
does it has a referent?
is it an inferrable?
check for indicators
is the concept already in the list?
is the concept value already in the list?
is a substring already in the list, or is it a substring of a value already in the list?
is it a name or a title?
robert redford unused
S-list (1): robert redford (actor), programmes 0 (deixis),
used size is now: 2
S-list (1): robert redford (actor), programmes 0 (deixis),
added robert redford to s-list, size is now: 2
finalize temp
hislist finalized
type history list finalized
usedSize = 2
tempList size = 0
tempList size after update = 2
size of last entry in tempList = 2
slist temp finalized
```

Figure 31. Sample output from the reference resolution module when handling a concept without referential properties.

3.6.2. Overview of the classes

In this section a detailed overview will be given of the different classes used in the reference resolution module. In appendix E the actual implementation of these classes are given.

3.6.2.1. Main Interface

The main interface performs the following operations:

1. **Initialize**: load data about which information must be extracted from the data presented by the parser and how. Open input stream. Load filter data. Create display reader object.
2. **Process data**: extract information from the data presented by the parser, and translate them into concepts. Some filtering is also done here.
   a) Find begin of next grammar data or user input.
   b) Find begin of next concept, ignore fillers and expletives.
   c) Read and filter attributes.
      - Filter out noise.
      - Read attributes.
      - Ignore attributes not specific to the resolution module.
      - Filter content_type concepts for actors, directors, and protagonists.
      - Filter concept values for ambiguous information.
      - Create sub- and superconcepts if necessary.
   d) Create concept object.
   e) assign index number to concept
   f) Repeat b) – d) until all concepts are read.
3. **Call Main Engine**: request Main Engine to handle the concepts from the grammar.
4. **Call Display Reader**: request Display Reader for display information.
5. **Call Main Engine**: request Main Engine to handle the concepts from the display.
6. **Repeat 2 till 5**.

3.6.2.2. Display Reader

The display reader performs the following operations:

1. **Initialize**: open the input stream
2. **Read display data**: extract information from the data presented by the parser, and translate them into concepts. Each concept is assigned the same index number. The concepts are put in the list of the appropriate concept type and the non-program concepts are also set as subconcept of the corresponding program concepts.
3. **Return data**.

3.6.2.3. Main Engine

The main engine performs the following operations:

1. **Initialize**: create deixis filter object, reference detection & classification module, update module, one anaphora resolution module, demonstrative resolution module, definite description resolution module, pronoun resolution module.
2. **Determine input**: determines whether the concepts originate from the user, or the display.
3. **Handle system input**: save the updates from the best hypothesis, update the salience and history list with display data.
   a) find concept indicating the best hypothesis.
   b) request Update Module to save the updates from the best hypothesis.
   c) request Grouping Module to create groups for the concepts.
   d) request Update Module to add the concepts to the salience and history list
4. **Handle user input**: find and resolve references if applicable.

a) request Deixis Filter to find deictic input.
b) request Update Module to add deictic input to salience and history list.
c) for each object:
   - request Reference Detection and Classification Module to determine the referential property.
   - if the reference is a compound reference, find the constraints for the superconcept and resolve and update the salience and history list with it.
   - request Constraints Detection Module to determine constraints for the reference.
   - request appropriate resolution module to solve reference.
   - request Update Module to add concept to salience and history list.
d) request Update Module to finalize the salience and history list: remove unused concepts from the salience list and backup the updates for this hypothesis.
e) request Output Module to generate output.

3.6.2.4. Update Module

The update module performs the following operations:

1. **Save**: save the temporary updates from the right hypothesis, and discard the rest.
2. **Update**: updates the salience and the history list.
3. **Temporarily Update**: updates a temporary salience and history list.
4. **Finalize**: removes unused concepts from the salience list.
5. **Temporarily Finalize**: removes unused concepts from the temporary salience list and creates a backup of the temporary salience and history list.

3.6.2.5. Salience List

The salience list performs the following operations:

1. **Save**: save the temporary updates from the right hypothesis, and discard the rest.
2. **Add**: adds the concept to the salience list.
   a) Tag the concept as OLD, MED, or NEW, according to the tagging criteria described in section 2.2.4.
   b) Insert the concept to the salience list according to the sorting criteria described in section 2.2.4.

3. **Temporarily Update**: adds the concept to the temporary salience list.
   a) Tag the concept as OLD, MED, or NEW, according to the tagging criteria described in section 2.2.4.
   b) Insert the concept to the temporary salience list according to the sorting criteria described in section 2.2.4.
4. **Finalize**: removes unused concepts from the salience list.
5. **Temporarily Finalize**: removes unused concepts from the temporary salience list and creates a backup of the temporary list.

3.6.2.6. History List

The history list performs the following operations:
1. **Save**: save the temporary updates from the right hypothesis, and discard the rest.
2. **Add**: adds the concept to the history list.
   a) Determine the type of the concept.
   b) Determine whether a list for the type exists. Create if necessary.
   c) Add the concept to the type list.
   d) Remove oldest concept if the number of concepts exceeds the maximum (sliding window technique).
3. **Temporarily Update**: adds the concept to the temporary salience list.
   a) Determine the type of the concept.
   b) Determine whether a list for the type exists. Create if necessary.
   c) Add the concept temporary to the type list.
   d) Remove oldest concept if the number of concepts exceeds the maximum (sliding window technique).

3.6.2.7. Grouping Module

The grouping module performs the following operations:
1. **Create groups**: groups are created as follows:
   a) Find the list concept containing the programs in the list of concepts.
   b) Create groups of programs using the items in the list concept.
      - for each concept get the subconcepts.
      - for each subconcept create a group based on it if it does not already exist.
      - add the concept to the group.
      - assign the subconcepts to the group.
      - when all concepts are done, remove the groups with only one concept.
      - create a group which contains every concept.
   c) Add the groups to the list of concepts.
   d) Add the individual programs to the list of programs, so that the concepts at the top of the display will be accessed first from the history list.

3.6.2.8. Deixis filter

The deixis filter performs the following operations:
1. **Find deixis**: the following is done during this operation:
   a) For each concept in the list of concepts.
   b) Check if the input origin is deixis.

c) Remove the deictic concept from the list of concepts.
d) Add the deictic concept to the list of deictic concepts
e) Return the list of deictic concepts.

3.6.2.9. Reference Detection & Classification Module

The Reference Detection and Classification Module performs the following operations:

1. **Detect & Classify**: to determine whether a concept has a refential property the following is done:
   a) Look at the type whether it indicates the referential property.
   b) Filter out titles and concept_types
   c) Look at the value for clues about the referential property. This is done because certain concepts have a higher preference than the reference concept, and otherwise will not be recognized as such.
   d) return the referential property.

3.6.2.10. Constraint Detection Module

The Constraint Detection Module performs the following operations:

1. **Detect Constraints**: in order to find the constraints the following is done:
   a) Try to detect constraints for complete value.
   b) If failed, test whether the concept is a compound reference.
   c) In case of a compound referent, add the superconcept as a constraint.
   d) Try to detect constraints for each word of the concept value. In case of conflict, select the one with the highest priority, in case of equal priority, assign value 'mixed'.
   e) Look at the subconcepts for constraints.
   f) If still no constraints found, look at type for constraints.
   g) Look at the complete sentence (list of concepts) for contextual constraints.
   h) return constraints.

3.6.2.11. Pronoun Resolution Module

The Pronoun Resolution Module performs the following operations:

1. **Resolve**: the following steps are taken to resolve a pronominal reference:
   a) Determine whether the pronoun is reflexive, possessive or personal.
   b) Request the salience list for the most salient concept, until referent is found.
   c) Request Constraint Detection Module to determine constraints for the salient concept.
   d) Check whether the concepts are compatible according to the constraints.
   e) Check whether the concepts are compatible according to binding constraints (*lacking a grammar to provide binding constraints, only one experimental constraint is implemented: a non-reflexive, non-possesive and non-personal pronoun cannot refer to the most recent concept. This is implemented so that at least references in the following most basic sentence structure can be resolved:* 'Bob and Bill met each other at the mall, he gave him a book.' *Using this binding constraint* 'him' *cannot refer to* 'he'. *This will not work in more complex sentence structures. Should the tagger be used, then the following binding constraints should be used: 1) A pronoun which has the function of subject or direct object,*

cannot co-refer with a direct object, indirect object or oblique item, which follow the pronoun, without an intervening subject. 2) A pronoun with a non-nil embed value cannot refer to an object which precedes it, if there is no object in between with the embed value of nil).

f) Return the most salient compatible concept.

3.6.2.12. Demonstrative Resolution Module

The Demonstrative Resolution Module performs the following operations:

1. **Resolve**: the following steps are taken to resolve a demonstrative reference:
   a) Request the salience list for the most salient concept, until referent is found.
   b) Request Constraint Detection Module to determine constraints for the salient concept.
   c) Check whether the concepts are compatible according to the constraints.
   d) Return the most salient compatible concept.

3.6.2.13. Definite Description Resolution Module

The Definite Description Resolution Module performs the following operations:

1. **Resolve**: the following steps are taken to resolve a definite description:
   a) Request Concept Type Filter to determine the compatible concept types.
   b) If the concept type is part of the list of a concept, and the referent must be the earliest or latest start time in the list. Return the earliest or latest compatible concept. Lookup of the start time is done in the list of subconcepts of the concept(*Compatibility is checked by requesting the Constraint Detection Module to determine the constraints for the candidate referent and compare them with the constraints of the reference).*
   c) If the concept type is a subconcept of a concept, find the compatible subconcept.
   d) If the concept type is part of the list of a concept, and the referent must be the n^{th} concept of the (sub)list, find the n^{th} compatible concept (*The referent can be the n^{th} program of the list, or the n^{th} movie of the list, in which case the list of movies is a sublist of the entire list).*
   e) Else request the salience list for the most salient concept, and find the first compatible concept.
   f) If no compatible concept is found, look at the compatible concept types for the most recent compatible concepts.
   g) Return the referent.

3.6.2.14. One Anaphora Resolution module

The Definite Description Resolution Module performs the following operations:

2. **Resolve**: the following steps are taken to resolve a definite description:
   h) Request Concept Type Filter to determine the compatible concept types.
   i) If the concept type is part of the list of a concept, and the referent must be the earliest or latest start time in the list. Return the earliest or latest compatible concept. Lookup of the start time is done in the list of subconcepts of the concept (*Compatibility is checked by requesting the Constraint Detection Module to determine the constraints for the candidate referent and compare them with the constraints of the reference).*

j) If the concept type is a subconcept of a concept, find the compatible subconcept.
k) If the concept type is part of the list of a concept, and the referent must be the n^{th} concept of the (sub)list, find the n^{th} compatible concept (*The referent can be the n^{th} program of the list, or the n^{th} movie of the list, in which case the list of movies is a sublist of the entire list).*
l) Else request the salience list for the most salient concept, and find the first compatible concept.
m) If no compatible concept is found, look at the compatible concept types for the most recent compatible concepts.
n) Return the referent.

3.7. Summary

In this chapter the requirements of the reference resolution module were specified. These were narrowed down to the following:

Must haves
- Resolution of references to an entity from another modality.
- Resolution of references to an entity introduced previously via a noun phrase.
- Resolution of references to a property of an entity from another modality.
- Operational within SPICE-EPG.
- Operational in real-time.
- Not dependent on an extensive lexicon.

Should haves
- Robustness
- Adaptable for other applications
- Parameterized settings
- Resolution of references to a superset of individual entities from another modality.
- Filter out references to nothing at all
- No increase in system requirements
- No increase in processing time
- Written in C++

Based on these requirements the reference resolution model from [Str98] was choosen, and the grammar requirements for the reference resolution module were specified:
- Recognize the references.
- Recognize the objects which can be referred to.
- Recognize phrases which add contextual constraints.
- Recognize forms where expletives occur.
- Provide information on relationship between concepts.

Recognition of the various concepts can be entirely done by the grammar. To find the information about the relationship between concepts, additional filters are required

though. Determining whether a concept has such a relationship with a reference, that it can add contextual constraints to it is done by looking at the constraints of the concept. A concept which modifies the concept and adds contextual constraints to it, has some requirements before the contextual constraints are assigned. These requirements are tested against the constraints already found for the concept. Another method to determine the relationship between two concepts which follow each other, is to create a single concept with both phrases in it and specify the relationship between them in the attributes.
Having determined the method to resolve the reference and the information provided by the grammar, the general outline of the algorithm can be specified. In general the anaphora resolution model in SPICE-EPG consists of a set of filters, a database, a salience-list, a history list and routines to find the referent. The processing of the information can be split into two parts, namely the system information processing part and the user information processing part. The system information processing part in short is as follows:

- The SPICE-EPG display provides information on the items on the screen.
- These items are converted into an internal representation
- Lists containing the concepts are created. Dependencies between concepts are set.
- Group concepts are created depending on several grouping criteria.
- Concepts are sorted.
- Concepts are added to the lists.

The user information processing part is in short as follows:

- The SPICE-EPG grammar provides information on the phrases from the user's utterance and pointing events.
- Uninteresting phrases like fillers and expletives are filtered out.
- Actor, director and protagonist information is filtered out.
- Concepts created by pointing events are filtered out and added to the lists.
- For each phrase is determined whether it is a reference or not.
- If the concept is not a reference, it is added to the lists.
- If the phrase is a reference, the form of the reference is determined.
- A list of constraints is created based on the implicit information of the phrase.
- A list of constraints is created based on the context.
- For each object in the salience list a list of properties is created.
- This list of properties is compared to the list of constraints for compatibility.
- The first compatible object is returned as the referent.
- If no referent is found, concepts which are out of focus, are compared with the reference if the anaphora is of descriptive form. Otherwise no referent is returned.
- Type lists from the history list which are not compatible with the reference are filtered out.
- The most recent compatible referent is looked up in the remaining compatible groups.
- The salience and history lists are updated.

Chapter 4.

Evaluation

In this chapter the evaluation of the reference resolution module and the problems encountered during the evaluation are discussed. The reference resolution module is tested both offline as well as online. With offline testing is meant that the input is received from typed text which has been manually parsed into concepts, which mirror the structure produced by the grammar. With online testing is meant that the input is received from spoken text which has been parsed by the SPICE-EPG grammar. During offline testing the examples in appendix A were used, which are within the scope of the program. During online testing a set of tasks were used, which were developed to test the usability of the SPICE-EPG system by a student from Nijmegen Catholic University [Goe01]. These tasks can be found in appendix F. Evaluation of the reference module is in principle performed the same way for offline and online testing, although for offline testing more complex sentence structures could be used. Also during offline testing, no system processing is done, so the display output consisted of some dummy program information. For online evaluation the system had to be voice trained before testing could be done, otherwise recognition would be too bad to get any results.

4.1. Evaluation method

In general iterative implementation was used as the evaluation method for both offline and online testing:

1. Process the training corpus.
2. Note the reference resolution errors.
3. Analyze the steps taken by the reference resolution module.
4. Isolate the source of the reference resolution errors.
5. Modify the source of the reference resolution errors.
6. Perform step 1 through 6 until all errors which can be corrected are solved.
7. Determine the number of correctly resolved references and the number of incorrectly resolved references based on the test corpus.

4.2. Choice of the corpus

Before the testing can be done, a suitable corpus or set of suitable corpera must be choosen. Preferably this corpus consists of spoken or transcribed text from a conversation in which information is requested and assignments for tasks are given. The most obvious choice would have been the test corpus already used to test the SPICE-EPG functionality. Unfortunately, since the SPICE-EPG was not able to solve references at that time, any form of reference was consciously avoided in that corpus. Another available corpus was

the *REIS* corpus. This corpus consists of conversations between a caller and a public transportation travel information service, where the caller asks for directions with public transportation. Although the corpus does contain some references, very few suitable references are found in the text.
A possibility would have been to create a corpus, trying to use as many references as possible for testing, but these examples would be very artificial. To find a compromise between thinking out a corpus with artificial examples and performing natural tasks within the environment, the task descriptions for the usability test for the SPICE-EPG was taken (see appendix F), and performed using references where they would naturally occur.
In addition, because the set of examples from appendix A, which fall within the scope. formed the basis for the design of the reference resolution module, these examples were tested too in the offline evaluation.

4.3. Errors and problems encountered during testing

Several errors were found during offline or online testing. Most errors were related to the lack of syntactic information and the filters used. Some of these are already described in section 3.4.6.

4.3.1. Conflicting constraints

In certain cases some constraints from different words in a single concept conflict whith each other. For instance in the phrase, *Billy Crystal and Meg Ryan* the name *Billy Crystal* and *Meg Ryan* add the constraint *singular*, while *and* add the constraint *plural*. It is necessary to check for conflicts, because if this is not done, and the constraints simply added, references cannot be resolved. For example when *they* is encountered and tried to resolve to *Billy Crystal and Meg Ryan*, it encounters both the constraint *singular* as well as *plural*, resulting in an incompatibility of the concepts. *And* will always indicate that the concept is a plural, it should have higher precedence then the constraint *singular* derived from *Billy Crystal* and *Meg Ryan*, and it is not really handy to calculate the sum of different constraints, i.e. *singular + singular = plural*, it has been decided that each constraint should have a number indicating its precedence, so that in case of conflicting constraints, the one with the highest value takes precedence.
In the same example there is another conflict, namely *male* and *female*. When constraints of the same type are encountered, and there is no precedence for either constraint, the constraint value is set to *mixed* to indicate two different constraints apply.
Conflicting constraints can also occur from outside the concept, from the context. It may be possible that the constraint from the context has a higher or lower precedence than the constraints derived from the concept itself.

4.3.2. Constraints differ for different concept types

There are cases where the contextual constraints will differ for the different concept types encountered, for example in *the program where she stars*. the verb *stars* indicates that *she* is an actor, whereas for *the program* it indicates that it is a movie. It is possible to let the concept add constraints to just one concept type, but in some cases the extra information may be necessary to resolve the reference correctly. Therefore the possibility to specify the contextual constraints depending on the different concept types is implemented.
The same applies for contraints for certain concept values. In certain cases, the constraints will depend on the concept types, because the concept value is ambigues. For example *movie* may refer to the category movie, or refer to a program of the category movie. The constraints for these two possibilities differ, and thus the possibility to specify the constraints depending on the concept types is implemented.

4.3.3. Display contains less than actual data

Because the SPICE-EPG system consists of many more or less independent modules, which often do not contain information about each other status, other than waiting for input, and transmitting output, no information is available on how much of the data to be displayed is actually displayed. This caused some errors where a reference to an item on the screen resulted in a referent not on the screen. To remedy this problem, the ordering of the data had to be reversed, so that the items on top of the screen are accessed first, instead of the items which are at the end, and maybe off screen. However, this is only a temporary problem, as in a future version it would be possible to limit the information to the programs displayed.

4.3.4. Grammar conflicts with content description

The content description concept is a special concept, which matches part of a phrase which is not recognized in the grammar to part of the content description of a program. The problem here lies in the fact that the SPICE-EPG system can only process the content description concept for a lookup of the program. If the grammar creates a non-content description concept of a phrase referring to the content description, no content descrition concept is created, and the content description cannot be matched to the program. Actor, director and protagonist are part of the content description, and in order to solve references to them they must be recognized from the text by the grammar. This results in that the SPICE-EPG system cannot find programs with these persons. The solution lies in removing them from the grammar and creating a filter wich searches the content description concepts for actors, directors and protagonist, and create a concept for the reference resolution module. A problem still unsolved is that references to the actor, director or protagonist will not be recognized as a content description by the SPICE-EPG system. This must be solved in the SPICE-EPG system, which is only accessible by the co-workers of the group.

4.3.5. Misassignment of constraints

The lack of a syntactic parser sometimes resulted in a misassignment of constraints. This was caused when a concept consisted of multiple words and this concept was part of a larger concept. For example in the concept *the program on channel 5*, the concept *channel 5* consists of two words and is part of the concept *the program on channel 5*. When constraints are assigned, the constraint detection module recognizes *program* and assigns the constraints for *program*, i.e. *type=program*, *gender=neutral*, *number=single*, *abstract=no*, when *channel* is encountered, the constraints for *channel* are added instead of *channel 5*. To solve this a filter is built to replace the words *channel 5* with *channel5*.

4.3.6. Empty concept graph

Sometimes the system returns an empty concept graph. This is the case when nothing the user said was recognized. Because the reference resolution module assumes that during the user turn always something is said, no checks were implemented for this situation. Naturally this resulted in a crash of the module. To solve this, a dummy concept is created with only dummy values, so that it will not interfere with the other data.

4.3.7. Misrecognition causing to look for lists

Sometimes the system misrecognizes the user, and returns a reference to a program at a certain position on the list, while the list is empty. For example: *the last program*, while nothing is on the screen. To avoid system crashes, checks must be made before items of the list are accessed.

4.3.8. Concepts overriding reference concepts

Because the grammar is trained on tasks to be performed by the system, some concepts are given a higher probability. For example *the second movie* would be split up into *(date) the second*, and *(category) movie*, instead of *(definite description) the second movie*. Attempts to increase the probability for definite descriptions of this form initially failed. This was the result of a preference for short concepts over longer concepts. Changing this preference resulted in a preference for long filler concepts. Lowering the preference for filler concepts finally remedied this problem.

4.3.9. Misrecognition of pronouns

Sometimes when noise is present, when the system stops listening to the user to soon, or when the system simply misrecognizes the user, pronouns are returned. This may cause in unwanted shifts in focus. This is not really problematic, since the user is interacting

with the system, and has the option to lead the system into the correct direction when things go wrong. Still it is annoying, and certain situations can be avoided. Often when pronouns are wrongfully recognized, they are the only meaningful concept. For instance the system recognizes only *it*. The system cannot do anything with this information, and it causes only an unintentional shift in focus in the reference resolution system. In these cases it is best if the reference is simply ignored. The same applies of recoginition of single articles. Sometimes the articles *the* and *a* are recognized, and some titles are matched to these articles. This is really annoying, and it would be best if these are filtered out also, since it is highly improbable that the user actually meant the titles which are matched to the articles.

4.3.10. Non-recognition of articles

For the reference resolution model it is very important to recognize whether a definite or indefinite article is used, since a definite article means a concept is in focus, while an indefinite article means it is out of focus. Recognition of the definite article is also important for recognition of a definite description. *The movie* indicates a reference to a movie, while *movie* will be recognized as a category. The SPICE-EPG system sometimes has a problem with recognizing articles, and as such references are not recognized. In certain cases it is possible to infer that a definite article should have been present, for instance *second movie* almost certain should have been *the second movie*. Relaxing the criteria for recognizing definite description will solve some of the problems. In the other cases better training of the system would have to do.

4.3.11. Two different input streams

This problem was not directly related to the reference resolution model, but surfaced when the reference resolution module was integrated and tested in the SPICE-EPG system, and caused some errors. Originally it was assumed that both the display data as well as the parsed sentences would be received using the same stream and that the data representation would be the same. This was not the case, so an additional object had to be implemented to read the display data. The problem which caused some strange errors was that concept types were not called the same in both data streams, so that references to items of the display data were not matched correctly. A filter was implemented to find the non-matching concept type names and match them accordingly.

4.4. Perfomance of the reference resolution module

During offline evaluation, the examples from appendix A were tested. Not all examples were tested though, because many of them are not solvable in the environmental constraints from SPICE-EPG. It should be noted that not all references were correctly resolved on the first try, but after modificating the code several times most references could be resolved correctly.

During online evaluation, several tasks from [Goe01] were used to test the online performance of the reference resolution module. These tasks were meant to test the usability of the SPICE-EPG system in general.

4.4.1. Test results

Considering the number of tests performed it is hard to really give a performance ratio. The reason that the number of tests is low, is that no suitable corpus was available, and the SPICE-EPG system and the grammar only allow a limited set of possibilities and constructions which can be tested. In addition, testing is very time consuming, especially the online testing, because recognition is slow and performance poor. Commands must be repeated very often before the utterance is correctly recognized, and often in a sequence of utterances, one of the utterances is recognized very badly, resulting in multiple incorrectly filled slots, which conflict with the user task. Often the slots must be reset, and the process can start over again. The input data can be saved for later test runs, but often the grammar had to be adapted, in which case the old input data becomes obsolete and the whole process of inputting speech commands to the SPICE-EPG system must be done over again. It is not possible to use the wave files, because the recognition and behaviour of the EPG will differ each time the grammar has been changed. In the end all correctly recognized references were resolved correctly.
Offline testing is not as tedious as online testing, but still takes much time, because the parsing and display information had to be manually simulated. Once the simulation data is present, it is possible to reuse it over and over again, after each adaptation, though. Most errors caused by bugs in the reference resolution model could be corrected during offline testing. In the end 33 of the 35 references in the corpus were resolved correctly. The incorrectly resolved references are discussed in section 4.4.1.1.

4.4.1.1. Offline evaluation

In the offline tests almost all references were finally correctly resolved to its referent. In only one case it failed. This was when a definite description was not resolved correctly due to the lack of information, which was necessary, and a pronoun referred to it later. In the utterance *The serial I saw last night, when will it be continued?* the definite description *the serial I saw last night* refers to an action or state of the user, which might or might not be stored in the knowledge of the dialogue manager. The anaphora resolution module in any case does not have any access to this information, so *the serial I saw last night* could not be resolved to any concept in the salience or history list of the module. Because the definite description could not be resolved, an error occured when an attempt was made to tag the definite description as OLD, MED, or NEW. This is the result of the fact that no information is present how to deal with unresolved references. Due to this error, the definite description was tagged as UNTAGGED, a tag reserved for concepts which should not be put in the salience list. So when the pronoun *it* was encountered, the salience list contained no concept which was compatible, resulting in another failure of correct resolution of a referent. It could be argued that rules must be introduced to correctly tag a failed reference, so that later references can be resolved to the failed reference. On the other hand a failed reference does not contain any information, and is thus useless to the system for the purpose of performing tasks. Any reference to the failed reference will be equally useless.

So the following references were succesfully tested during offline evaluation:

- reference to an entity from another modality in focus,
- reference to an entity from another modality out of focus,
- reference to an entity introduced into the discourse with a noun phrase in focus,
- reference to an entity introduced into the discourse with a noun phrase out of focus,
- reference to a superset of an entity,
- reference to a property of an entity.

4.4.1.2. Online evaluation

In the online tests reference resolution is much more difficult, because recognition errors introduce concepts which are not in the utterance. Also, less complex senteces are used, because this increases the chance that the systems does not understand what is said. The speech recognizer has trouble recognizing certain words, of which some are important for reference resolution. Words like *he*, *she*, and the definite article *the* are not often recognized. Another problem is that certain concepts which can be referred to are badly recognized. This is the case with names of actors and other persons. For instance the question whether a movie with *Robert de Niro* would be running this night, was not recognized in the twenty attempts made, and other actors fared little better. In the few cases where the actor was indeed recognized, the reference *he* or *she* was not or misrecognized, causing the actor to leave the focus of attention of the reference resolution module. It is not possible to solve this problem in the reference resolution module directly, because the module has no way to determine whether a shift in focus was due misrecognition of the user or because the user shifted his focus. Because of this pronominal anaphora to persons was not tested in the online evaluation. However, considering the fact these kind of references were succesfully resolved in the offline test, it is expected to work correctly should the referent and reference be recognized. Sometimes during online testing when a misrecognition occurs, a refence is wrongfully introduced, usually the pronouns *it* and *its* or the demonstratives *that*, *this*, and *them*. When this happens three things may occur: the reference is not resolved to anything, the reference is resolved to a concept currently in the focus of the user, or a concept out of the focus of the user.

Reference resolved to nothing

In the online tests there were six instances where a misrecognition of the user's utterance resulted in a reference to nothing:

- *reset* was misrecognized as *that* : in the previous turn, there was no user utterance, but the system started recognizing. Apparently this silence was matched to the word *ok*. *Ok* is not a noun phrase and as such is not put into the salience list. After the turn, the everything which was not used during the turn is removed from the salience list. Since nothing is used, the salience list is empty, except for the group of movies from the display. When the reference *that* was encountered, a look up is done in the salience list for a compatible concept, but since non is found, no referent is assigned to *that*.
- *reset* was misrecognized as *eight that*: in the previous turn, *reset* was misrecognized as *that*, which did not have a referent. Consequently, nothing was put in the salience

list. When the reference *that* was encountered, a look up is done in the salience list for a compatible concept, but since none is found, no referent is assigned to *that*.

- *record the second program* was misrecognized as *record this*: in the previous turn, the utterance *are there any sports tonight* was correctly recognized. In this utterance no relevant noun phrases are encountered, which can be put in the salience list. So when the reference *this* was encountered, a look up is done in the salience list for a compatible concept. The only concept in the salience list is the group of news programs which are displayed on the screen. Because the group of programs has the constraint *number = plural* and the reference *this* has the constraint *number = singular*, so no compatible referent is found.
- *record the ten o'clock news* was misrecognized as *record it that o'clock news*: in the previous turn, the utterance *what news is on tonight* was correctly recognized. In this utterance no relevant noun phrases are encountered, which can be put in the salience list. So when the reference *it* was encountered, a look up is done in the salience list for a compatible concept. The only concept in the salience list is the group of news programs which are displayed on the screen. Because the group of programs has the constraint *number = plural* and the reference *it* has the constraint *number = singular*, so no compatible referent is found.
- *record the ten o'clock news* was misrecognized as *record it them o'clock news*: in the previous turn, the utterance *what news is on tonight* was correctly recognized. In this utterance no relevant noun phrases are encountered, which can be put in the salience list. So when the reference *it* was encountered, a look up is done in the salience list for a compatible concept, but since none is found, no referent is assigned to *it*.
- *record the last shows program* was misrecognized as *record the last shows program*: in the previous turn, the utterance *what news is on tonight* was correctly recognized. This resulted in a list of news programs to be displayed on the screen. This list was added to the history of lists. When the reference *the last shows program* was encountered, the constraints detection module detected the word *last* and added the constraint *listentry = -1*, indicating that the referent is part of a list. Because no list is specified, the most recent list is used as default to lookup *the last shows program*. The reference also has the constraint *category = show*, so the last program in the list with the constraint *show* must be found. Since only programs with the constraint *category = news* are in the list, no referent can be assigned to *the last shows program*.

Reference resolved to something in focus

In the online test there were two instances where a misrecognition resulted in a reference to something in focus:

- *switch to that channel* misrecognized as *this the a channel*: in the previous turn, the utterance *what's on cnn right now* was correctly recognized. Since *cnn* is a relevant noun phrase, it is added at the front of the salience list. When the salience list is cleaned up from the concepts which were not used during the turn, the concept *cnn* remains. So when the reference *this* is encountered, a look up in the salience list is done, and the channel *cnn* is encountered first. Since *cnn* and *this* are compatible, the reference resolution module assumes *this* refers to *cnn*.
- *switch to that channel* misrecognized as *its that a channel*: Since *cnn* is a relevant noun phrase, it is added at the front of the salience list. When the salience list is

cleaned up from the concepts which were not used during the turn, the concept *cnn* remains. So when the reference *its* is encountered, a look up in the salience list is done, and the channel *cnn* is encountered first. Since *cnn* and *this* are compatible, the reference resolution module assumes *this* refers to *cnn*.

Reference to something out of focus

In the online test there were two instances where a misrecognition resulted in a reference to something not in the user's focus:

- *record the ten o'clock news* was misrecognized as *record it that o'clock news*: in the previous turn, the utterance *what news is on tonight* was correctly recognized. In this utterance no relevant noun phrases are encountered, which can be put in the salience list. So when the reference *it* was encountered, a look up is done in the salience list for a compatible concept. The only concept in the salience list is the group of news programs which are displayed on the screen. Because the group of programs has the constraint *number = plural* and the reference *it* has the constraint *number = singular*, so no compatible referent is found. Because the reference *it* does not contain a referent, it is incorrectly tagged, and added to the salience list. When the reference *that* is encountered, the only concept in the salience list is *it*, and as such is assigned to the concept *that*.
- *record the ten o'clock news* was misrecognized as *record it them o'clock news*: in the previous turn, the utterance *what news is on tonight* was correctly recognized. During system turn, the group of news programs is put on the salience list, since it is possible to refer to them as a whole. When the reference *it* is encountered, no compatible referent is found. Because the reference *it* does not contain a referent, it is incorrectly tagged, and added to the salience list. When the reference *them* is encountered, the reference resolution module assigns the constraint *number = plural* to the reference, so when a lookup is done in the salience list, the first and only compatible concept is the group of programs.

It was found however that recognition errors do not really create strange shifts in the focus of attention of the system, which would cause correctly recognized references to be resolved wrongly. During the tests, some misrecognitions contained references, which were resolved to the concept in focus, so no shifts in focus occured. Also when the system would move away from the desired task, for example display a totally different topic, the user will try to move back to the task at hand, rather than just relentlessly trying to have the system recognize the utterance.

Chapter 5.

Conclusion

In section 1.1 the goals of the second part of the project are stated. These goals were:

- Find a method to compensate for the lack of syntactic data in a shallow parsing environment.
- Implement the proposed model for operation in a speech recognition environment.
- Test the proposed model in a speech recognition environment.

5.1. Finding a method to compensate for lack of syntactic data

It is found that syntactic data is very important in reference resolution. While it is not necessary to determine the focus of attention using the model described in [Str98] (most other models do depend on syntactic information [Kam93] [All95] [Ken96] [Mcc96] [Mur96]), it is important to have syntactic information to determine dependencies between concepts. These dependencies are very important to determine syntactic and contextual constraints.
There is no single method to compensate for the lack of syntactic data, rather a group of methods to provide information to fill in some of the blanks left by the lack of syntactic data. Some problems can be overcome using a set of filters or a set of rules, which mirror a downgraded version of the rules when syntactic data is available. These are usually much less accurate than the rules with syntactic data though.
Three methods are used to determine the dependencies between phrases: phrases are split up and constraints determine whether a phrase modifies another or not, phrases are grouped together and use of a subconcept determines the relationship, and phrases are grouped together and use of a superconcept determines the relationship.
Splitting up phrases into seperate concepts and use constraints to determine whether a phrase modifies another or not, is especially useful when two concepts do not necessarily follow each other (for example *show me it* and *show it)*, but it is expected that other concepts or fillers can come in between. In the example *show it*, *show* indicates that *it* is a program. This is implemented by looking at the constraints of the reference when the command *show* is encountered when looking for additional constraints from the context. If the reference has the constraints *gender = neutral* and *abstract = nonabstract*, then the constraint *type = program* is added.
Sometimes it is easier to group phrases together, especially when the relationship is crucial for correct resolution of the reference, and no or very few different concepts can come between them. For example *the news program on cnn*. Sometimes it is even necessary that this construction is used, because the phrases cannot be split up, without compromising the recognition of the reference, for example *the eight p.m. news*. The consequence of splitting this phrase into *eight p.m.* and *news* would be that *news* can no

longer be recognized as a reference. Therefore *the eight p.m. news* is grouped together and an attribute *subconcept = start_time* : *eight p.m.* is assigned to the concept. This way the two concepts can still be recognized as two seperate concepts, while it they are still recognized as a reference and information about their relation is available.
Another construct which is similar to the previous one was necessary for phrases like *from the previous list the second program* and *the second program of the previous list*. Again the relationship is very crucial to correctly resolve the reference, but in addition to this, *the previous list* must be resolved before *the second program*. In the second phrase, *the previous list* comes after *the second program*, while the algorithm used resolves references in the order they are encountered. This is again done by including information about their relationships in the attribute. In this example the attributes *concept = the second program* and *superconcept = the previous list* are added. The attribute *superconcept* indicates that it should be resolved first, and the attribute *concept* indicates that its referent is the one which is needed. With this construct both the information which concept must be resolved first and the information about the relationship is known.
Finally a set of binding constraints must be defined. Because no syntactic information is available only one simple binding constraint is defined, which has the purpose of preventing the object and the subject to co-refer: *a pronoun in its accusative form cannot refer to the most recent noun phrase in the same sentence*. This will only work in utterances like: *Bob gave him a present*, where the object directly follows the subject. In the application, this constraint is never used, because descriptive phrases like this are irrelevant to the application.

5.2. Implementation of the proposed model

In the nine months for this project a functional reference resolution prototype was implemented, with the following properties:

- Resolution of references to an entity from another modality.
- Resolution of references to an entity introduced previously via a noun phrase
- Resolution of references to a property of an entity from another modality.
- Resolution of references to a superset of individual entities from another modality.
- Filters out references to nothing at all
- Operational within SPICE-EPG
- Operational in real-time
- Not dependent on an extensive lexicon
- Adaptable for other applications
- Parameterized settings
- No increase in system requirements
- No increase in processing time
- Written in C++

The code can be found in appendix E.

5.3. Test results.

Considering the number of tests performed it is hard to really give a performance ratio. The reason that the number of tests is low, is that no suitable corpus was available, and the SPICE-EPG system and the grammar only allow a limited set of possibilities and constructions which can be tested. In addition, testing is very time consuming, especially the online testing, because recognition is very slow and very bad. Commands must be repeated very often before the utterance is correctly recognized, and often in a sequence of utterances, one of the utterances is recognized very badly, resulting in multiple incorrectly filled slots, which conflict with the user task. Often the slots must be reset again, and the process can start over again. The input data can be saved for later test runs, but often the grammar had to be adapted, in which case the old input data becomes obsolete and the whole process of inputting speech commands to the SPICE-EPG system must be done over again. It is not possible to use the wave files, because the recognition and behaviour of the EPG will differ.
Offline testing is not as tedious as online testing, but still takes a while, because the parsing and display information had to be manually simulated. Once the simulation data is present, it is possible to reuse it over and over again, after each adaptation, though. Most errors caused by bugs in the reference resolution model could be corrected during offline testing.
In the offline tests almost all references were finally correctly resolved its referent. In only one case it failed. This was when a definite description was not resolved correctly due to the lack information, which was necessary, and a pronoun referred to it later.
So the following references were succesfully tested during offline evaluation:

- reference to an entity from another modality in focus,
- reference to an entity from another modality out of focus,
- reference to an entity introduced into the discourse with a noun phrase in focus,
- reference to an entity introduced into the discourse with a noun phrase out of focus,
- reference to a superset of an entity,
- reference to a property of an entity.

In the online tests reference resolution is much more difficult, because recognition errors introduce concepts which are not in the utterance. Also, less complex senteces are used, because this increases the chance that the systems does not understand what is said. The speech recognizer has trouble recognizing certain words, of which some are important for reference resolution. Words like *he, she,* and the definite article *the* are not often recognized. Another problem is that certain concepts which can be referred to are badly recognized. This is the case with names of actors and other persons. For instance the question whether a movie with *Robert de Niro* would be running this night, was not recognized in the twenty attempts made, and other actors fared little better. In the few cases where the actor was indeed recognized, the reference *he* or *she* was not or misrecognized, causing the actor to leave the focus of attention of the reference resolution module. It is not possible to solve this problem in the reference resolution module directly, because the module has no way to determine whether a shift in focus was due misrecognition of the user or because the user shifted his focus. Because of this pronominal anaphora to persons was not tested in the online evaluation. However,

considering the fact that these kind of references were succesfully resolved in the online test, it is expected to work correctly should the referent and reference be recognized. Sometimes during online testing when a misrecognition occurs, a refence is wrongfully introduced, usually the pronouns *it* and *its* or the demonstratives *that*, *this*, and *them*. When this happens three things may occur: the reference is not resolved to anything, the reference is resolved to a concept currently in the focus of the user, or a concept out of the focus of the user.
It was found however that recognition errors do not really create strange shifts in the focus of attention of the system, which would cause correctly recognized references to be resolved wrongly. During the tests, some misrecognitions contained references which were resolved to the concept in focus, so no shifts in focus occured. Also when the system would move away from the desired task, for example display a totally different topic, the user will try to move back to the task at hand, rather than just relentlessly trying to have the system recognize the utterance.

Chapter 6.

Recommendations

Having spent a few months working with the SPICE-EPG and the working prototype of the reference resolution module, part of which was for the purpose of testing the reference resolution module, several recommendations can be made to improve the performance of the SPICE-EPG with the integrated reference resolution module. Also recommendations can be done about things to try out with reference resolution for which no time was available in during this project.
Recommendations to increase the performance are:

- Filter out non-filler concepts which make no sense.
- Relax the grammar for reference recognition.
- Use a second parser to allow more complex concepts.
- Determine references for all hypothesis.
- Penalize hypotheses with unresolved reference.
- Find a way to process references to content description.
- Find a way to tag content description and add the tagged information to the concept.

Recommendations for things to try out are:

- Use a filter to determine when to skip the salience list.
- Solve one anaphora using the salience list.

6.1. Filter out non-filler concepts which make no sense

During the testing of the system, it often occured that misrecognition of an utterance resulted in non-filler concepts, which made absolutely no sense. Often concepts of the type title or content description were created with only the value *a* or *the*. This resulted in the system looking for programs starting with an *a* (*a and e)* and programs with *the* in the content description. It is very unlikely that when these words are encountered, the programs returned are indeed meant by the user. It is therefore recommended to filter out concepts like these. Other concepts encountered due to misrecognitions are references like *it, them, that,* and *this*. Sometimes these references are the only concepts in the utterance, sometimes they are in combination with other concepts. If the reference is the only concept in the utterance, it is certain that misrecognition has occured, either of the reference or another important word in the sentence. Either way the system cannot do anything with the information, because no new information is provided to perform a task. Therefore it is recommended that references are filtered out, if no command or additional information is presented, like *record it,* or *are there any movies with that actor tomorrow?* where *tomorrow* and *movies* for example are classified as additional information. A method should be implemented though to prevent the system loosing

focus, when filtering occurs. It is expected that this will improve the general performance of the system.

6.2. Relax the grammar for reference recognition

The definite article *the* and the demonstratives *this, that, these,* and *those* are important indicators that descriptive references are indeed references. For example without *this* in *this movie* the concept would just be *movie,* and does not contain any information about being a reference, it is more likely to be a category of a program. For this reason these indicators were used to determine whether a concept is a reference or not. Currently the grammar is to strict to correctly recognize these descriptive references when they occur most of the time. This is because speech recognition is still not 100%, and often the definite article *the* and demonstratives *this* and *that* are not recognized. In some cases it is possible to infer that the indicator was not recognized though, and the concept could still be recognized as a reference. For example in *record last movie,* it is clear that *last movie* is a reference to the last movie, so the grammar specification should be relaxed, so that in cases where it is still clear that a reference is meant, the concept is still recognized as a reference.

6.3. Use a second parser to allow more complex concepts

Currently certain relationships between concepts are specified by assigning one of these concepts as a subconcept of the other. For example in *the six p.m. news* contains the subconcept *start time: six p.m.* In the grammar this is done by assigning the attribute *subconcept* with the value *start time: six p.m.* and later this attribute is translated into a concept. This method works fine most of the time, but it does not allow specification of really complex relations, such as the subconcept having a subconcept, which also has a subconcept or a superconcept (see section 3.4.6 for explanation about the relationship subconcept and superconcept). A second parser which processes the partially parsed phrases would be able to solve this problem.

6.4. Determine references for all hypothesis

In the current system only the references of the best hypothesis are resolved. However, the dialogue manager does not have to accept the best hypothesis from the context interpretation module and can select a different hypothesis based on its own knowledge. During testing this did not happen, but when this happens, the salience and the history list might contain incorrect information, and the dialogue manager may have to work with unresolved references. The reference resolution module already has the ability to work with different hypotheses without gettng confused. The only problem may be the increase in processing time and memory usage.

6.5. Penalize hypotheses with unresolved reference

Should be decided that the reference resolution module has to determine the references for all hypotheses, than the result of the reference resolution may be used to determine which hypothesis is better. Hypotheses which contain references which are not resolved to any concept, can be penalized, because it can be expected that this is a result of misrecognition.

6.6. Find a way to process references to content description

During online testing, it was found that if a concept which is part of the content description is specified in the grammar, it would not be processed as part of the content description. For example when *Mel Gibson* was recognized as an *actor* by the grammar, the SPICE-EPG would not look for programs with *Mel Gibson* in the content description. This problem was solved by removing these concepts from the grammar, and when a content description is encountered, checked whether it contains actor information. However, considering the way the SPICE-EPG works, references to an actor will not result in a look up of programs with that actor. So even though a reference to *Mel Gibson* is correctly resolved, the SPICE-EPG is not able to search for movies with *Mel Gibson*. Therefore a way must be found to process references to content descriptions.

6.7. Find a way to tag content description and add the tagged information to the concept

Currently the only information given with the programs are the category, the channel, the date, the end time, the start time, and the title. So it is not possible to refer to the program with parts of the content description, for example: *record the science fiction movies*. If a way could be found to extract the interesting information from the content description and add them as subconcepts to the programs on the list, it would be possible to refer to programs with references like *switch to the serie with Ross Kemp*.

6.8. Use a filter to determine when to skip the salience list

In [Byr99] it is stated that descriptive anaphora is usually used to refer to something which is currently not in focus, in contrast to pronominal anaphora which always referes to something in focus. It is however not impossible or even uncommon for descriptive anaphora to refer to something in focus, for example: *Yesterday we went to the cinema to see Shrek. The movie was very funny*. However, when additional modifiers are used, for example in *Last week's movie left a much deeper impression though*, it is very probable

that it refers to something which is currently out of focus. To decrease processing time, although it currently is not an issue, it could be possible to use a filter to determine when additional modifiers are used, so that the concepts out of focus can be directly accessed, instead of checking each concept in the salience list first.

6.9. Solve one anaphora using the salience list

Even though it is difficult with the current grammar to recognize one anaphora like *I would like to see one tomorrow too*, because no definite article or demonstrative is present to identify *one* as one anaphora, some thought is given how one anaphora could be resolved. One anaphora in cases like this will refer to a general type, rather than a specific concept which was introduced via noun phrase into the discourse before. It can be expected that the referent is currently in focus, because very little information is present to find a concept out of focus. Therefore it can be expected that the referent is in the salience list. So the first compatible referent in the salience list can be used to create a new concept. *Is there a movie with Robert de Niro today? How about one tomorrow?* the reference *one* would refer to the general type *movie with Robert de Niro*, so a copy of the concept *a movie of Robert de Niro* can be used to create the referent for *one*. Conflicting constraints from the one anaphora should override the properties of the copy, for example in *Are there any movies with Robert de Niro? How about one with Al Pacino?* the reference *one* is singular, while *movies* is plural, so *movies* should become *movie*. Also the concept *movies* would have the subconcept *actor: Robert de Niro*, which should be replaced with *Al Pacino*. Compatibility of the concepts in the salience list, can be determined by looking at contextual constraints and perhaps some filters to compensate for the lack of syntactic constraints. An example of a filter might be that the referent must have a subconcept type in common with the reference, like in *a movie with Robert de Niro* and *one with Al Pacino* both have a subconcept of the type *actor*. Another filter might be that the referent cannot be of a type which is already in the utterance, for example in *Bob has a beautiful girlfriend. Bill wants one too, one* does not refer to *Bob* or *Bill*, which precede *a beautiful girlfriend* in the salience list. Because *Bob* would be of type *guy* and *Bill* is too, according to this filter *one* cannot refer to a general type of *Bob* or *Bill*. Whether this would really work is something which can be tested in the future.

Bibliography

[All95] J.F. Allen. Natural Language Understanding – 2nd Edition. The Benjamin/Cummings Publishing Company, 1995.

[Aus95] H. Aust, M. Oerder, F. Seide, and V. Steinbiss. The Philips automatic train timetable information system. Speech Communication, 17(3-4):249-262, Nov. 1995.

[Bea99] D.L. Bean, E. Rillof. Corpus-Based Identification of Non-Anaphoric Noun Phrases. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*. 1999.

[Bla99] C.A. Black. A step-by-step introduction to the Government and Binding theory of syntax. http://www.sil.org/americas/mexico/ling/E002-IntroGB.pdf. Summer Institute of Linguistics. February 1999.

[Bod96] R. Bod & R. Scha. Data-oriented language processing: An overview. Technical report, ILLC, University of Amsterdam, Amsterdam, The Netherlands. 1996.

[Bre87] S.E. Brennan, M.W. Friedman & C.J. Pollard. A centering approach to pronouns. In *Proceedings of the Association for Computational Linguistics*, pp. 155-162. July 1987.

[Byr99] D.K. Byron. Resolving Pronominal Reference to Abstract Entities. Ph.D. thesis, University of Rochester Department of Computer Science. June 1999.

[Cho81] N. Chomsky. Lectures on Government and Binding. Dordrecht: Foris. 1981.

[Coh98] P. Cohen, M. Johnston, S. Oviatt, J. Clow, & I. Smith. The efficiency of multimodal interaction: a case study. In *Proceedings of the International Conference on Spoken Language*, 1998.

[Dal92] R. Dale. Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes. Cambridge, MA: MIT Press. 1992.

[Eck99] M. Eckert & M. Strube. Resolving Discourse Deictic Anaphora in Dialogues. In *Proceedings of the 9th Conference of the European Chapter of the Association of Computational Linguistics*, pages 37-44, 1999.

[Eck99b] M. Eckert & M. Strube. Dialogue Acts, Synchronizing Units and Anaphora Resolution. To appear in Journal of Semantics 2001 (not the final version), 1999.

[Eck99c] M. Eckert & M. Strube. Dialogue Acts, Synchronizing Units and Anaphora Resolution. In *Amstelogue '99: Workshop on Dialogue.* Amsterdam, The Netherlands, May 5-7, 1999.

[Fer00] A. Ferrández & J. Peral. A computational Approach to Zero-pronouns in Spanish. 2000.

[Fox87] B. Fox. Discourse Structure and Anaphora. Written and conversational English. Cambridge Studies in Linguistics. Cambridge University Press, Cambridge, 1987.

[Gar97] S. Garnsey, N. Pearlmutter, E. Myers & M. Lotocky. Contributions of verb bias and plausiblity to the comprehension of temporarily ambiguous sentences. Journal of Memory and Language, 37:58-93. 1997.

[Goe01] C. Goetheer. No information about title is available yet. Katholieke Universiteit Nijmegen. Publication date is expected at the end of 2001.

[Hah96] U. Hahn & M. Strube. Incremental Centering and Center Ambiguity. In *Proceedings of the 18th Annual Meeting of the Cognitive Science Society.* LaJolla, CA, 1996.

[Hob86] J. Hobbs. Resolving pronoun reference. In *Readings in Natural Language Processing.* Morgan Kaufmann, 1986.

[Kam93] M. Kameyama, R. Passonneau, and M. Poesio. Temporal centering. In: *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 70-77, Columbus, OH, June 1993.

[Kap93] H. Kamp & U. Reyle. From Discourse to Logic. Dordrecht: Kluwer. 1993.

[Kar95] F. Karlsson, A. Voutilainen, J. Heikkila, & A. Antilla. Constraint grammar: A language-independent system for parsing free text. Mouton de Gruyter, Berlin/New York, 1995.

[Keh93] A. Kehler. Intrasentential Constraints on Intersentential Anaphora in Centering theory. Workshop on Centering Theory in Natural Occurring Discourse, University of Pennsylvania, 1993.

[Keh93b] A. Kehler. A discourse copying algorithm for ellipsis and anaphora resolution. In *Proceedings of the Sixth Conference of the European*

Chapter of the Association for Computational Linguistics (EACL-93), pages 203--212, April 1993.

[Ken96] C. Kennedy, B. Boguraev - Anaphora for everyone: pronominal anaphora resolution without a parser. Proceedings of the 16th International Conference on Computational Linguistics COLING'96, Copenhagen, Denmark, 5-9 August 1996.

[Kel00] A. Kellner, S. Martin, P. Philips, T. Portele & B. Souvignier. SPICE - a first research prototype. Conversational User Interfaces, PFL-Aachen Report 1463/00, July 2000.

[Mar00] P. Martínez-Barco, & M. Palomar. Dialogue structure influence over anaphora resolution. In *MICAI 2000: Advances in Artificial Intelligence, Lecture Notes in Artificial Intelligence, vol.1793*. O. Cairo, L.E. Sucar and F.J. Cantú Eds. Acapulco (Mexico). Springer-Verlag. pp. 515-525. ISBN: 3-540-67354-7, April 2000.

[MaS00] S. Martin, B. Souvignier. Problems in Grammar Design, Technical Report PFL-Aachen 1452/00 April 2000]

[Mcc96] J. McCarthy. A Trainable Approach to Coreference Resolution for Information Extraction. University of Massachusetts Amherst Department of Computer Science. September 2000.

[Mit98] R. Mitkov. Robust pronoun resolution with limited knowledge. In *Proceedings of ACL '98*, pages 869-875, 1998.

[Mur96] M. Murata. Anaphora Resolution in Japanese Sentences Using Surface Expressions and Examples. Ph.D. thesis, Kyoto University. December 1996.

[Pas89] R.J. Passonneau. Getting at discourse referents. In: *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 51-59, Vancouver, BC, 1989.

[Pas93] R. J. Passonneau. Getting and keeping the center of attention. Challenges in Natural Language Understanding, M. Bates & R.M. Weischedel, pages 179-227. Cambridge University Press. 1993.

[Pas96] R.J. Passonneau. Interaction of Discourse Structure with Explicitness of Discourse Anaphoric Noun Phrases. Computational Linguist, 17(1): 21-48.

[Phi00] Philips, P. Multimodal Integration – A stochastic Framework for the Integration of Speech recognition and Pointing Input. Technical Report PFL-Aachen 1442/00 February 2000.]

[Pin00] L. Pineda. A Model for Multimodal Reference Resolution. Computational Linguistics Volume 26, Number 2, 2000.

[Pri81] E. F. Prince. Toward a taxonomy of given-new information. In P. Cole (Ed.), *Radical Pragmatics*, pp. 223-255. New York, N.Y.: Academic Press, 1981.

[Ram93] O. Rambow. Pragmatic aspects of scrambling and topicalization in German. *Workshop on Centering Theory in Naturally-Occuring Discourse.* Institute of Research in Cognitive Science, Philadelphia, Penn, University of Pennsylvania, May 1993.

[Sac74] H. Sacks, E. Schegloff & G. Jefferson. A simplest systematic for the organization of turn taking for conversation. *Language*, 50(4):696-735, 1974.

[Sid83] Candace L. Sidner. Focusing in the comprehension of definite anaphora. In M. Brady and R.C. Berwick, editors, Computational Models of Discourse, pages 267-330. The MIT Press, Cambridge, Massachusetts, 1983.

[Sou00] B. Souvignier, A. Kellner, B. Reuber, H. Schramm & F. Seide. The thoughtful elephant: Strategies for spoken dialogueue systems. IEEE Transactions on Speech and Audio Processing, 8(1):51-62, January 2000.

[SVM99] Sound&Vision Magazine Feb/Mar 1999 pp73.

[Str95] M. Strube, U. Hahn. *ParseTalk* about sentence- and text-level anaphora. In *Proceedings of EACL-95*, pp. 291-296. 1995.

[Str96] M. Strube, U. Hahn. Functional centering. In *Proceedings of ACL-96*. pages 270-277, 1996.

[Str96b] M. Strube. Processing Complex Sentences in the Centering Framework. In *Proceedings of ACL-96*, pages 378-380, 1996.

[Str98] M. Strube. Never Look Back: An Alternative to Centering. In *Proceedings of ACL-98*, pages 1251-1257, 1998.

[Val90] E. Vallduvi. The informational Component. University of Pennsylvania, Department of Linguistics. 1990.

[Val96] E. Vallduvi & E. Engdahl. The linguistic realization of information packaging. In *Linguistics*, 34:459-519, 1996.

Appendix A

Examples of references to be solved in the ideal case

In this appendix an overview is given of what co-workers at Philips think the EPG should be able to handle. Note that for many example applies that even if the reference resolution module would able to solve the references, the dialogue manager would not be able to understand the situation.

SPICE, are there any movies starring Mel Gibson today?
How about this week?
- *References: ellipsis*
Can you show me more information about this movie? <Points to a movie in the list>
- *References: demonstrative, entity from another modality.*
Could you show me the list again?
- *References: definite description, entity from another modality.*
Please record the Mad Max movies
- *References: definite description, superset of individual entities from another modality.*

Are there any samurai movies today?
Who is the director of this one? <Points to a movie in the list>
- *References: definite description, a property of an entity from another modality, entity from another modality.*
Are there any other movies directed by him this month?
- *References: pronoun, entity that was introduced into the discourse via a noun phrase.*
Is there any news on the latest earthquake?
- *References: definite description, world knowledge not mentioned in the discourse.*
Any other news about that country?
- *References: demonstrative, property of an entity that was introduced into the discourse via a noun phrase.*
Are there any movies by Roman Polansky?
In which of these does he stars himself?
- *References: pronoun, entity that was introduced into the discourse via a noun phrase, superset of individual entities from another modality.*
Please record the most recent one.
- *References: definite description, an entity from another modality.*

Hello Computer...
Turn on TV!

Which channel is this?
- *References: demonstrative, entity from another modality.*
Will they bring some news after that series?
- *References: pronoun, demonstrative, property of an entity, entity from another modality.*
When is the next news broadcast?
- *References: definite description, world knowledge.*
Okay, go there.
- *References: entity that was introduced via a noun phrase.*
Oh, I've missed the interesting part. Record the next repetition of it.
- *References: definite description, property of an entity, world knowledge, entity introduced via a noun phrase.*
Show me the TV-guide for this channel.
- *References: definite description, world knowledge, property of an entity.*
Which station shows a movie tonight?
Has it started already?
- *References: pronoun, entity introduced via a noun phrase.*
Ok, then show it.
- *References: pronoun, entity introduced via a noun phrase.*
When will it end?
- *References: pronoun, entity introduced via a noun phrase.*
Is that Sandra Bullock?
- *References: demonstrative, entity from another modality.*
Isn't she beautiful?
- *References: pronoun, entity introduced via a noun phrase.*
Oh, I forgot you're a computer, you don't know anything about this.
- *References: demonstrative, fact*
Will she marry that guy in the end?
- *References: pronoun, entity introduced via a noun phrase, entity from another modality, world knowledge.*
Oh, what a surprise.... Don't you have another film with her in your database?
- *References: pronoun, entity introduced via a noun phrase.*
What a pity. Turn of TV and play that radio station I heard yesterday!
- *References: demonstrative, world knowledge.*

Gimme info about the fourth movie?
- *References: definite description, entity from another modality.*

Can I see the last one?
- *References: definite description description, entity from another modality.*

I want to see a James Bond movie.
Do you have other movies with him?
- *References: pronoun, entity introduced via a noun phrase.*

Gimme info on that movie.
- *References: demonstrative, entity introduced via a noun phrase/entity from another modality.*

When are the next news showing?
- *References: definite description, world knowledge*
And the one after that?
- *References: definite description, demonstrative, world knowledge, entity introduced via a noun phrase*

Do you have more information about the last thing.
- *References: definite description, entity introduced via a noun phrase/entity from another modality.*

Show me more about the film in the line with the cursor.
- *References: definite description, entity from another modality.*
The same for the film in the green-displayed line.
- *References: definite description, action, entity from another modality.*
Oh, not that one, the previous one, please.
- *References: definite description, definite description, entity from another modality.*
Can you repeat the last words.
- *References: definite description, entity from another modality.*

What is that about?
- *References: demonstrative, entity from another modality/entity introduced via a noun phrase.*
What time does it start?
- *References: pronoun, entity introduced via a noun phrase.*
Can you remove that film from the list?
- *References: demonstrative, definite description, entity from another modality/entity introduced via a noun phrase.*
Can you add the previous one to the list?
- *References: definite description, entity from another modality.*
No, not that one, the other one.
- *References: demonstrative, definite description, entity from another modality.*

Are there any movies with X next week?
Which of them is together with Y?
- *References: demonstrative, superset of individual entities from another modality.*

No, I don't like this one (those).
- *References: definite description, demonstrative, entity from another modality, superset of individual entities from another modality.*
With whom are the others?
- *References: definite description, superset of individual entities from another modality.*
O.k. so please record the first (second, etc.) one!
- *References: definite description, entity from another modality.*

Any basketball on TV tonight?
Could you record the second half of it (that).
- *References: definite description, pronoun, demonstrative, property of an entity from another modality, an entity from another modality.*

When are the next news on TV?
- *References: definite description, world knowledge.*
I don't like them.(I hate those/these guys.) Any news after that?
-*References: pronoun, demonstrative, superset of individual entities from another modality, property of an entity from another modality.*
Great, you can record that.
- *References: demonstrative, entity from another modality.*

<user>: "Can you record the next game of my favorite Baseball team?"
- *References: definite description, world knowledge.*
<system>: "Your favorite Baseball team are the San Francisco Giants. Their next game starts next Saturday at noon. I will record it."
<user>: "When is their next one [the game of the San Francisco Giants]?"
- *References: pronoun, definite description, entity introduced via a noun phrase, world knowledge.*
<system>: "The following game of the San Francisco giants starts next Sunday at 5p.m."
<user>: "Can you record this one [the game] as well, please?"
- *References: definite description, entity introduced via a noun phrase.*
<system>: "I am sorry, there will not be enough space on the tape."
<user>: "OK, let me exchange it [the tape]."
- *References: pronoun, entity introduced via a noun phrase.*
<system>: "Alright, please use a 600 minutes tape."
<user>: "Alright, I have done it [exchanging the tape]."
- *References: pronoun, entity introduced via a noun phrase.*
<system>: "Thank you, I will record the game on Sunday as well."
<user>: "On which channel will it [the game] be?"
- *References: pronoun, entity introduced via a noun phrase.*
<system>: "ESPN."
<user>: "Which baseball game is ABC showing on Sunday?"
<system>: "On Sunday, ABC shows Los Angeles Dodgers versus Arizona Diamondbacks."

<user>: "When does this one [the game] start?"
- *References: definite description, entity introduced via a noun phrase.*
<system>: "The game on ABC on Sunday starts at 1pm."
<user>: "Will there be other games on Sunday?"
<system>: "Yes, there will be four other baseball games next Sunday."
<user>: "Is there one [a game] with the New York Yankees?"
- *References: definite description, entity introduced via a noun phrase.*
<system>: "No. The New York Yankees do not play on Sunday."
<user>: "What about the other teams in their [New York Yankees] division?"
- *References: pronoun, entity introduced via a noun phrase.*
<system>: "The Boston Red Sox are playing against the Chicago White Sox on Sunday at 3pm."
<user>: "Will this one [the game] be on TV?"
- *References: definite description, entity introduced via a noun phrase.*
<system>: "Yes, Boston Red Sox versus Chicago White Sox will be broadcast on Fox Sports Net."
<user>: "Excellent. How did they [Boston Red Sox and Chicago White Sox] play in their [Boston Red Sox and Chicago White Sox] last games?"
- *References: pronoun, entity introduced via a noun phrase.*
<system>: "The Boston Red Sox are on a five game winning streak while the Chicago White Sox lost their last two games."
<user>: "What about the hockey teams in this area [Boston and Chicago]?"
- *References: demonstrative, entity introduced via a noun phrase.*
<system>: "The Boston Bruins have lost three games in a row and the Chicago Blackhawks have won the last two games."

I am looking for a movie with Kate Winslet where she plays an Australian girl.
- *References: pronoun, entity introduced via a noun phrase.*

From the last list of movies, the second one.
- *References: definite description, entity from another modality, property of an entity from another modality.*

I said: a movie with Robert Redford! He does not act in these ones.
- *References: pronoun, definite description, entity introduced via a noun phrase, superset of entities from another modality.*

The serial I saw last night, when will it be continued?
- *References: definite description, pronoun, world knowledge, entity introduced via a noun phrase.*

Is this a science fiction movie?
- *References: demonstrative, entity from another modality/entity introduced via a noun phrase.*

There is a movie with Billy Crystal and Meg Ryan where they play two singles in New York.
- *References: pronoun, entity introduced via a noun phrase.*

I am looking for a movie. It should start around 8pm.
- *References: pronoun, entity introduced via a noun phrase.*

Give more information on the last one.
- *References: definite description, entity from another modality.*

I want to remove that one.
- *References: Definite description, entity from another modality/entity introduced via a noun phrase.*

Put it on my recording list.
- *References: pronoun, entity from another modality/entity introduced via a noun phrase.*

Remove the earlier one. (when two items overlap on the recordlist)
- *References: Definite description, entity from another modality.*

Show me the one that has Julia Roberts in it (when a list of movies is displayed).
- *References: Definite description, entity from another modality.*

What about later?
- *References: ellipsis.*

Appendix B

Grammar to recognize reference forms

```
<REFERENCE> ::= (200) he
  value := 'he'
  number := 'singular'
<REFERENCE> ::= (200) she
  value := 'she'
  number := 'singular'
<REFERENCE> ::= (1) it
  value := 'it'
  number := 'singular'
<REFERENCE> ::= (1) they
  value := 'they'
  number := 'plural'
<REFERENCE> ::= (200) his
  value := 'his'
  number := 'singular'
<REFERENCE> ::= (200) her
  value := 'her'
  number := 'singular'
<REFERENCE> ::= (200) him
  value := 'his'
  number := 'singular'
<REFERENCE> ::= (1) its
  value := 'its'
  number := 'singular'
<REFERENCE> ::= (1) their
  value := 'their'
  number := 'plural'
<REFERENCE> ::= (1) that
  value := 'that'
  number := 'singular'
<REFERENCE> ::= (1) this
  value := 'this'
  number := 'singular'
;;<REFERENCE> ::= (1) which
;;  number := 'singular'
<REFERENCE> ::= (1) these
  value := 'these'
  number := 'singular'
<REFERENCE> ::= (1) those
   value := 'those'
  number := 'plural'
<REFERENCE> ::= (1) himself
  value := 'himself'
  number := 'singular'
<REFERENCE> ::= (1) herself
  value := 'herself'
  number := 'singular'
<REFERENCE> ::= (1) itself
  value := 'itself'
  number := 'singular'
<REFERENCE> ::= (1) themselves
  value := 'themselves'
  number := 'plural'

<director_type> ::= (1) director
  value := 'director'
<director_type> ::= (1) directors
  value := 'directors'

<actor_type> ::= (1) actor
  value := 'actor'
<actor_type> ::= (1) actors
  value := 'actors'
<actor_type> ::= (1) star
  value := 'star'
<actor_type> ::= (1) stars
  value := 'stars'
<actor_type> ::= (1) moviestar
  value := 'moviestar'
<actor_type> ::= (1) moviestars
  value := 'moviestars'
<actor_type> ::= (1) actress
  value := 'actress'
<actor_type> ::= (1) actresses
  value := 'actresses'

<person> ::= (1) guy
  value := 'guy'
<person> ::= (1) guys
  value := 'guys'
<person> ::= (1) man
  value := 'man'
<person> ::= (1) men
  value := 'men'
<person> ::= (1) mister
  value := 'mister'
<person> ::= (1) misters
  value := 'misters'
<person> ::= (1) boy
  value := 'boy'
<person> ::= (1) boys
  value := 'boys'
<person> ::= (1) gentleman
  value := 'gentleman'
<person> ::= (1) gentlemen
  value := 'gentlemen'
<person> ::= (1) hunk
  value := 'hunk'
<person> ::= (1) hunks
  value := 'hunks'
<person> ::= (1) lad
  value := 'lad'
<person> ::= (1) lads
  value := 'lads'
<person> ::= (1) gall
  value := 'gall'
<person> ::= (1) galls
  value := 'galls'
<person> ::= (1) woman
  value := 'woman'
<person> ::= (1) women
  value := 'women'
<person> ::= (1) lady
  value := 'lady'
<person> ::= (1) ladies
  value := 'ladies'
```

```
<person> ::= (1) girl
 value := 'girl'
<person> ::= (1) girls
 value := 'girls'
<person> ::= (1) babe
 value := 'babe'
<person> ::= (1) babes
 value := 'babes'
<person> ::= (1) chick
 value := 'chick'
<person> ::= (1) chicks
 value := 'chicks'
<person> ::= (1) lass
 value := 'lass'
<person> ::= (1) lasses
 value := 'lasses'
<person> ::= (1) person
 value := 'person'
<person> ::= (1) persons
 value := 'persons'

<constraint_adjective> ::= (1) australian
 value := 'australian'
 possible_concept := '-none-'
<constraint_adjective> ::= (1) green
 value := 'green'
 possible_concept := '-none-'
<constraint_adjective> ::= (1) red
 value := 'red'
 possible_concept := '-none-'
<constraint_adjective> ::= (1) james bond
 value := 'james bond'
 possible_concept := 'protagonist, james bond'

<brand_new_description> ::= (1) a <person>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) an <person>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) a <channel_>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) an <channel_>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) a <programme_s>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) an
<programme_s>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) a <schedule>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) an <schedule>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) a <db_category>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) an <db_category>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) a <specified_list>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) an
<specified_list>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) a <actor_type>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) an <actor_type>
 subconcept1 := '-none-'
<brand_new_description> ::= (1) a
<constraint_adjective> <person>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) an
<constraint_adjective> <person>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) a
<constraint_adjective> <channel_>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) an
<constraint_adjective> <channel_>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) a
<constraint_adjective> <programme_s>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) an
<constraint_adjective> <programme_s>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) a
<constraint_adjective> <schedule>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) an
<constraint_adjective> <schedule>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) a
<constraint_adjective> <db_category>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) an
<constraint_adjective> <db_category>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) a
<constraint_adjective> <specified_list>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) an
<constraint_adjective> <specified_list>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) a
<constraint_adjective> <actor_type>
 subconcept1 := <2>.possible_concept
<brand_new_description> ::= (1) an
<constraint_adjective> <actor_type>
 subconcept1 := <2>.possible_concept

<constraint_modifier> ::= (1) directed
<constraint_modifier> ::= (1) directs
<constraint_modifier> ::= (1) acted
<constraint_modifier> ::= (1) acts
<constraint_modifier> ::= (1) starred
<constraint_modifier> ::= (1) stars
<constraint_modifier> ::= (1) played
<constraint_modifier> ::= (1) plays

<specified_list> ::= (1) <schedule> of
<programme_s>
 value :=  <1>.value ~~ 'of' ~~ <3>.value

<DEFINITE_DESCRIPTION> ::= (1) the <channel_>
 value := 'the' ~~ <2>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <channels>
 value := 'the' ~~ <2>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the
<programme_s>
 value := 'the' ~~ <2>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <schedule>
 value := 'the' ~~ <2>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the
<db_category>
 value := 'the' ~~ <2>.value
 subconcept1 := '-none-'
```

<DEFINITE_DESCRIPTION> ::= (1) the <actor_type>
 value := 'the' ~~ <2>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <director_type>
 value := 'the' ~~ <2>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <person>
 value := 'the' ~~ <2>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the one
 value := 'the one'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <specified_list>
 value := 'the' ~~ <2>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <day_ordinal> <programme_s>
 value := 'the' ~~ <2>.value ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <day_ordinal> one
 value := 'the' ~~ <2>.value ~~ 'one'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <day_ordinal> <db_category>
 value := 'the' ~~ <2>.value ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <day_ordinal> <db_category> <programme_s>
 value := 'the' ~~ <2>.value ~~ <3>.value ~~ <4>.value
 subconcept1 := 'category,' ~~ <3>.value
<DEFINITE_DESCRIPTION> ::= (1) the <day_ordinal> <programme_s> from below
 value := 'the' ~~ <2>.value ~~ <3>.value ~~ 'from below'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <day_ordinal> from below
 value := 'the' ~~ <2>.value ~~ 'from below'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <day_ordinal> one from below
 value := 'the' ~~ <2>.value ~~ 'one from below'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <day_ordinal> <db_category> from below
 value := 'the' ~~ <2>.value ~~ <3>.value ~~ 'from below'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <day_ordinal> <db_category> <programme_s> from below
 value := 'the' ~~ <2>.value ~~ <3>.value ~~ <4>.value ~~ 'from below'
 subconcept1 := 'category,' ~~ <3>.value
<DEFINITE_DESCRIPTION> ::= (1) the last <programme_s>
 value := 'the last' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the last <schedule>
 value := 'the last' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the last one
 value := 'the last one'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the last <specified_list>
 value := 'the last' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the previous <programme_s>
 value := 'the previous' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the previous <schedule>
 value := 'the previous' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the previous <specified_list>
 value := 'the previous' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the previous one
 value := 'the previous one'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the earlier one
 value := 'the earlier one'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the earlier <programme_s>
 value := 'the earlier' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the earlier <db_category>
 value := 'the earlier' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the earlier <specified_list>
 value := 'the earlier' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the later one
 value := 'the later one'
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the later <programme_s>
 value := 'the later' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the later <db_category>
 value := 'the later' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the later <specified_list>
 value := 'the later' ~~ <3>.value
 subconcept1 := '-none-'
<DEFINITE_DESCRIPTION> ::= (1) the <constraint_adjective> <channel_>
 value := 'the' ~~ <3>.value
 subconcept1 := <2>.possible_concept
<DEFINITE_DESCRIPTION> ::= (1) the <constraint_adjective> <channels>
 value := 'the' ~~ <3>.value
 subconcept1 := <2>.possible_concept
<DEFINITE_DESCRIPTION> ::= (1) the <constraint_adjective> <programme_s>
 value := 'the' ~~ <3>.value
 subconcept1 := <2>.possible_concept
<DEFINITE_DESCRIPTION> ::= (1) the <constraint_adjective> <schedule>
 value := 'the' ~~ <3>.value
 subconcept1 := <2>.possible_concept
<DEFINITE_DESCRIPTION> ::= (1) the <constraint_adjective> <db_category>

```
  value := 'the' ~~ <3>.value
  subconcept1 := <2>.possible_concept
<DEFINITE_DESCRIPTION> ::= (1) the
<constraint_adjective> <actor_type>
  value := 'the' ~~ <3>.value
  subconcept1 := <2>.possible_concept
<DEFINITE_DESCRIPTION> ::= (1) the
<constraint_adjective> <director_type>
  value := 'the' ~~ <3>.value
  subconcept1 := <2>.possible_concept
<DEFINITE_DESCRIPTION> ::= (1) the
<constraint_adjective> <person>
  value := 'the' ~~ <3>.value
  subconcept1 := <2>.possible_concept
<DEFINITE_DESCRIPTION> ::= (1) the
<constraint_adjective> one
  value := 'the one'
  subconcept1 := <2>.possible_concept
<DEFINITE_DESCRIPTION> ::= (1) the
<constraint_adjective> <specified_list>
  value := 'the' ~~ <3>.value
  subconcept1 := <2>.possible_concept

<DEFINITE_DESCRIPTION> ::= (200) the
<given_date> <programme_s>
  value := 'the' ~~ <3>.value
  subconcept1 := 'date,' ~~ <2>.given_date.3 ~
(<2>.given_date.2 < 10 ? '-0' : '-') ~
<2>.given_date.2 ~ '-' ~ <2>.given_date.1
<DEFINITE_DESCRIPTION> ::= (200) the
<given_date> <db_category>
  value := 'the' ~~ <3>.value
  subconcept1 := 'date,' ~~ <2>.given_date.3 ~
(<2>.given_date.2 < 10 ? '-0' : '-') ~
<2>.given_date.2 ~ '-' ~ <2>.given_date.1
<DEFINITE_DESCRIPTION> ::= (200) the
<given_date> one
  value := 'the one'
  subconcept1 := 'date,' ~~ <2>.given_date.3 ~
(<2>.given_date.2 < 10 ? '-0' : '-') ~
<2>.given_date.2 ~ '-' ~ <2>.given_date.1
<DEFINITE_DESCRIPTION> ::= (200) the
<programme_s> on <given_date>
  value := 'the' ~~ <2>.value
  subconcept1 := 'date,' ~~ <4>.given_date.3 ~
(<4>.given_date.2 < 10 ? '-0' : '-') ~
<4>.given_date.2 ~ '-' ~ <4>.given_date.1
<DEFINITE_DESCRIPTION> ::= (200) the
<db_category> on <given_date>
  value := 'the' ~~ <2>.value
  subconcept1 := 'date,' ~~ <4>.given_date.3 ~
(<4>.given_date.2 < 10 ? '-0' : '-') ~
<4>.given_date.2 ~ '-' ~ <4>.given_date.1
<DEFINITE_DESCRIPTION> ::= (200) the one on
<given_date>
  value := 'the one'
  subconcept1 := 'date,' ~~ <4>.given_date.3 ~
(<4>.given_date.2 < 10 ? '-0' : '-') ~
<4>.given_date.2 ~ '-' ~ <4>.given_date.1


<DEFINITE_DESCRIPTION> ::= (5000) the
<time_and_time_duration> <programme_s>
  value := 'the' ~~ <3>.value
  subconcept1 := 'start time,' ~~ <2>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<time_and_time_duration> <programme_s>
  value := 'the' ~~ <3>.value
  subconcept1 := 'start time,' ~~ <2>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<time_and_time_duration> <category>
  value := 'the' ~~ <3>.value
  subconcept1 := 'start time,' ~~ <2>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<time_and_time_duration> one
  value := 'the one'
  subconcept1 := 'start time,' ~~ <2>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<programme_s> that starts at
<time_and_time_duration>
  value := 'the' ~~ <2>.value
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<category> that starts at
<time_and_time_duration>
  value := 'the' ~~ <2>.value
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the one
that starts at <time_and_time_duration>
  value := 'the one'
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<programme_s> which starts at
<time_and_time_duration>
  value := 'the' ~~ <2>.value
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<category> which starts at
<time_and_time_duration>
  value := 'the' ~~ <2>.value
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the one
which starts at <time_and_time_duration>
  value := 'the one'
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<programme_s> that starts on
<time_and_time_duration>
  value := 'the' ~~ <2>.value
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<category> that starts on
<time_and_time_duration>
  value := 'the' ~~ <2>.value
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the one
that starts on <time_and_time_duration>
  value := 'the one'
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<programme_s> which starts on
<time_and_time_duration>
  value := 'the' ~~ <2>.value
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<category> which starts on
<time_and_time_duration>
  value := 'the' ~~ <2>.value
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the one
which starts on <time_and_time_duration>
  value := 'the one'
  subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000) the
<programme_s> at <time_and_time_duration>
  value := 'the' ~~ <2>.value
  subconcept1 := 'start time,' ~~ <4>.time
```

<DEFINITE_DESCRIPTION> ::= (5000) the
<category> at <time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <4>.time
<DEFINITE_DESCRIPTION> ::= (5000) the one at
<time_and_time_duration>
 value := 'the one'
 subconcept1 := 'start time,' ~~ <4>.time
<DEFINITE_DESCRIPTION> ::= (1) the
<db_channel> <programme_s>
 value := 'the' ~~ <3>.value
 subconcept1 := 'channel,' ~~ <2>.channel
<DEFINITE_DESCRIPTION> ::= (1) the
<db_channel> <category>
 value := 'the' ~~ <3>.value
 subconcept1 := 'channel,' ~~ <2>.channel
<DEFINITE_DESCRIPTION> ::= (1) the
<db_channel> one
 value := 'the one'
 subconcept1 := 'start time,' ~~ <2>.channel
<DEFINITE_DESCRIPTION> ::= (1) the
<programme_s> on <db_channel>
 value := 'the' ~~ <2>.value
 subconcept1 := 'channel,' ~~ <4>.channel
<DEFINITE_DESCRIPTION> ::= (1) the <category>
on <db_channel>
 value := 'the' ~~ <2>.value
 subconcept1 := 'channel,' ~~ <4>.channel
<DEFINITE_DESCRIPTION> ::= (1) the one on
<db_channel>
 value := 'the one'
 subconcept1 := 'channel,' ~~ <4>.channel

<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <time_and_time_duration>
<programme_s>
 value := 'the' ~~ <3>.value
 subconcept1 := 'start time,' ~~ <2>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <time_and_time_duration>
<programme_s>
 value := 'the' ~~ <3>.value
 subconcept1 := 'start time,' ~~ <2>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <time_and_time_duration>
<category>
 value := 'the' ~~ <3>.value
 subconcept1 := 'start time,' ~~ <2>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <time_and_time_duration> one
 value := 'the one'
 subconcept1 := 'start time,' ~~ <2>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <programme_s> that starts at
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <category> that starts at
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> one that starts at
<time_and_time_duration>
 value := 'the one'
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <programme_s> which starts at
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <category> which starts at
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> one which starts at
<time_and_time_duration>
 value := 'the one'
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <programme_s> that starts on
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <category> that starts on
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> one that starts on
<time_and_time_duration>
 value := 'the one'
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <programme_s> which starts on
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <category> which starts on
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> one which starts on
<time_and_time_duration>
 value := 'the one'
 subconcept1 := 'start time,' ~~ <6>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <programme_s> at
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <4>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <category> at
<time_and_time_duration>
 value := 'the' ~~ <2>.value
 subconcept1 := 'start time,' ~~ <4>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> one at <time_and_time_duration>
 value := 'the one'
 subconcept1 := 'start time,' ~~ <4>.time
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <db_channel> <programme_s>
 value := 'the' ~~ <3>.value
 subconcept1 := 'channel,' ~~ <2>.channel
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <db_channel> <category>
 value := 'the' ~~ <3>.value
 subconcept1 := 'channel,' ~~ <2>.channel
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <db_channel> one

```
  value := 'the one'
  subconcept1 := 'channel,' ~~ <2>.channel
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <programme_s> on <db_channel>
  value := 'the' ~~ <2>.value
  subconcept1 := 'channel,' ~~ <4>.channel
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> <category> on <db_channel>
  value := 'the' ~~ <2>.value
  subconcept1 := 'channel,' ~~ <4>.channel
<DEFINITE_DESCRIPTION> ::= (5000)
<REFERENCE> one on <db_channel>
  value := 'the one'
  subconcept1 := 'channel,' ~~ <4>.channel

<COMPOUND_DEF_DESCR> ::= (10)
<DEFINITE_DESCRIPTION> 'of'
<DEFINITE_DESCRIPTION>
  super_concept := <3>.value
  concept := <1>.value
  subconcept1 := <1>.subconcept1
  subconcept2 := <3>.subconcept1
<COMPOUND_DEF_DESCR> ::= (10)
<DEFINITE_DESCRIPTION> 'of' <REFERENCE>
  super_concept := <3>.value
  concept := <1>.value
  subconcept1 := <1>.subconcept1
  subconcept2 := '-none-'
<COMPOUND_DEF_DESCR> ::= (10) from
<DEFINITE_DESCRIPTION><DEFINITE_DESCRIP
TION>
  super_concept := <2>.value
  concept := <3>.value
  subconcept1 := <2>.subconcept1
  subconcept2 := <3>.subconcept1
<COMPOUND_DEF_DESCR> ::= (10) from
<REFERENCE> <DEFINITE_DESCRIPTION>
  super_concept := <2>.value
  concept := <3>.value
  subconcept1 := '-none-'
  subconcept2 := <3>.subconcept1

<seem> ::= (1) seems
<seem> ::= (1) does seem
<seem> ::= (1) 'doesn\'t seem'
<seem> ::= (1) does not seem
<seem> ::= (1) seem
<seem> ::= (1) seemed
<seem> ::= (1) did seem
<seem> ::= (1) 'didn\'t seem'
<seem> ::= (1) not seem

<appear> ::= (1) appears
<appear> ::= (1) does appear
<appear> ::= (1) 'doesn\'t appear'
<appear> ::= (1) does not appear
<appear> ::= (1) appear
<appear> ::= (1) appeared
<appear> ::= (1) did appear
<appear> ::= (1) 'didn\'t appear'
<appear> ::= (1) not appear

<occur> ::= (1) occurs
<occur> ::= (1) does occur
<occur> ::= (1) 'doesn\'t occur'
<occur> ::= (1) does not occur
<occur> ::= (1) occur
<occur> ::= (1) occured
<occur> ::= (1) did occur
<occur> ::= (1) 'didn\'t occur'
<occur> ::= (1) not occur

<confirm> ::= (1) good
<confirm> ::= (1) alright
<confirm> ::= (1) okay
<confirm> ::= (1) fine
<confirm> ::= (1) nothing


<EXPLETIVE> ::= (1) it <seem>
<EXPLETIVE> ::= (1) did it <seem>
<EXPLETIVE> ::= (1) does it <seem>
<EXPLETIVE> ::= (1) it <seem> that
<EXPLETIVE> ::= (1) did it <seem> that
<EXPLETIVE> ::= (1) does it <seem> that
<EXPLETIVE> ::= (1) that <seem>
<EXPLETIVE> ::= (1) did that <seem>
<EXPLETIVE> ::= (1) does that <seem>
<EXPLETIVE> ::= (1) it <appear>
<EXPLETIVE> ::= (1) did it <appear>
<EXPLETIVE> ::= (1) does it <appear>
<EXPLETIVE> ::= (1) it <appear> that
<EXPLETIVE> ::= (1) did it <appear> that
<EXPLETIVE> ::= (1) does it <appear> that
<EXPLETIVE> ::= (1) that <appear>
<EXPLETIVE> ::= (1) did that <appear>
<EXPLETIVE> ::= (1) does that <appear>
<EXPLETIVE> ::= (1) it <occur>
<EXPLETIVE> ::= (1) did it <occur>
<EXPLETIVE> ::= (1) does it <occur>
<EXPLETIVE> ::= (1) it <occur> that
<EXPLETIVE> ::= (1) did it <occur> that
<EXPLETIVE> ::= (1) does it <occur> that
<EXPLETIVE> ::= (1) that <occur>
<EXPLETIVE> ::= (1) did that <occur>
<EXPLETIVE> ::= (1) does that <occur>
<EXPLETIVE> ::= (1) 'it\'s' <confirm>
<EXPLETIVE> ::= (1) 'it\'s not' <confirm>
<EXPLETIVE> ::= (1) it is <confirm>
<EXPLETIVE> ::= (1) 'it isn\'t' <confirm>
<EXPLETIVE> ::= (1) it is not <confirm>
<EXPLETIVE> ::= (1) is it <confirm>
<EXPLETIVE> ::= (1) 'isn\'t it' <confirm>
<EXPLETIVE> ::= (1) is it not <confirm>
<EXPLETIVE> ::= (1) <I> 'didn\'t mean that'
<EXPLETIVE> ::= (1) <I> 'didn\'t mean it'
<EXPLETIVE> ::= (1) 'that\'s not what I meant'
<EXPLETIVE> ::= (1) 'that isn\'t what I meant'
<EXPLETIVE> ::= (1) that is not what I meant
<EXPLETIVE> ::= (1) 'that\'s what I meant'
<EXPLETIVE> ::= (1) that is what I meant
<EXPLETIVE> ::= (1) that is exactly what I meant
<EXPLETIVE> ::= (1) 'that\'s not exactly what I
meant'
<EXPLETIVE> ::= (1) 'that is not exactly what I
meant'
<EXPLETIVE> ::= (1) that is precisely what I
meant
<EXPLETIVE> ::= (1) that was not what I meant
<EXPLETIVE> ::= (1) 'that wasn\'t what I meant'
<EXPLETIVE> ::= (1) that was what I meant
<EXPLETIVE> ::= (1) that was exactly what I
meant
<EXPLETIVE> ::= (1) that was not exactly what I
meant
<EXPLETIVE> ::= (1) that was precisely what I
meant
<EXPLETIVE> ::= (1) is that what I meant
```

<EXPLETIVE> ::= (1) was that what I meant
<EXPLETIVE> ::= (1) 'I didn\'t mean that'
<EXPLETIVE> ::= (1) 'I didn\'t mean it'
<EXPLETIVE> ::= (1) 'I didn\'t want that'
<EXPLETIVE> ::= (1) 'I didn\'t want it'
<EXPLETIVE> ::= (1) 'I didn\'t specify that'
<EXPLETIVE> ::= (1) 'I didn\'t specify it'

Appendix C

Phrases with expletives

```
it seems <that>
it does seem <that>
it doesn't seem <that>
it does not seem <that>
does it seem <that>
does it not seem <that>
it seemed <that>
it did seem <that>
it didn't seem <that>
it did not seem <that>
did it seem <that>
did it not seem <that>

it appears <that>
it does appear <that>
it doesn't appear <that>
it does not appear <that>
does it appear <that>
does it not appear <that>
it appeared <that>
it did appear <that>
it didn't appear <that>
it did not appear <that>
did it appear <that>
did it not appear <that>

it occurs <that>
it does occur <that>
it doesn't occur <that>
it does not occur <that>
does it occur <that>
does it not occur <that>
it occured <that>
it did occur <that>
it didn't occur <that>
it did not occur <that>
did it occur <that>
did it not occur <that>

it's ok <that>
it is ok <that>
it's not ok <that>
it isn't ok <that>
it is not ok <that>
is it ok <that>
isn't it ok <that>
is it not ok <that>

it's alright <that>
it is alright <that>
it's not alright <that>
it isn't alright <that>
it is not alright <that>
is it alright <that>
isn't it alright <that>
is it not alright <that>

that seems
that does seem
that doesn't seem
that does not seem
does that seem
does that not seem
that seemed
that did seem
that didn't seem
that did not seem
that it seem
that it not seem

that appears
that does appear
that doesn't appear
that does not appear
does that appear
does that not appear
that appeared
that did appear
that didn't appear
that did not appear
did that appear
did that not appear

that occurs
that does occur
that doesn't occur
that does not occur
does that occur
does that not occur
that occured
that did occur
that didn't occur
that did not occur
did that occur
did that not occur

that's ok
that is ok
that's not ok
that isn't ok
that is not ok
is that ok
isn't that ok
is that not ok

that's alright
that is alright
that's not alright
that isn't alright
that is not alright
is that alright
isn't that alright
is that not alright

that's not what I/you/he/she/etc. meant
that isn't what I/you/etc. meant
that is not what I/you/etc. meant
that's what I/you/etc. meant
that is what I/etc. meant
that is exactly what I/etc. meant
that's not exactly what I meant
that is precisely what I meant
that was ...
is that what I meant
```

was that what I meant
I didn't mean that
I didn't mean it
... wanted ...
... specified ...

it's all wrong

that's all wrong

it's nothing
it's fine
that's nothing
that's fine

Appendix D

System tasks and information requirements based on examples

U: SPICE, are there any movies starring Clint Eastwood today?
S:
Input: Concepts: (<input: user>) <genre: movie> <actor: Clint Eastwood> <date: today>
Main Interface: translates concepts into some internally used data structure.
Main Engine: detect deixis (<genre: movie> <actor: Clint Eastwood> <date: today>)
Deixis Detection Module: no concept tagged as deixis found.
Main Engine: detect & classify reference (genre: movies)
Reference Detection & Classification Module: *looks up in database?*(genre: movies; reference: NIL)
Main Engine: temporarily update (genre: movies)
Update Module: temp update concept type 'genre' history, temp update S-list
Main Engine: detect & classify reference (actor: Clint Eastwood)
Reference Detection & Classification Module: *looks up in database?* (actor: Clint Eastwood; reference: NIL)
Main Engine: temporarily update (actor: Clint Eastwood)
Update Module: temp update concept type 'actor' history, temp update S-list
There should be a timestamp or some other tag to indicate when it is encountered, because when a reference like 'the guy' is used, the algorithm should look into the different concept types with the attribute 'person' and find the most recent one.
Main Engine: detect & classify reference (date: today)
Reference Detection & Classification Module: *looks up in database?* (date: today; reference: NIL)
Main Engine: temporarily update (date: today)
Update Module: temp update concept type 'date' history, temp update S-list
Main Engine: translate output (<genre: movies> <actor: Clint Eastwood> <date: today)
Output Module: translates data into SPICE readable concepts.
S: <Shows list with today's movies starring Clint Eastwood>
Input: Concepts: (<input: system>) (<best: 1>) <list: info; programs: movies, genre: movie, date: today, [$movie_a$: …], …, [$movie_b$: …]> <programs: movies; genre: movie, date: today, [$movie_a$: …], …, [$movie_b$: …]> , [$movie_a$: …], …, [$movie_b$: …]
Main Interface: translates concepts into some internally used data structure.
Main Engine: update (best: 1)
Update Module: save updates from 1st try. Clear temporary updates.
Main Engine: update (list: info; …)
Update Module: update concept type 'list' history, update S-list
Main Engine: update (genre: movies; …)
Update Module: update concept type 'genre' history, update S-list
Main Engine: update (actor: Clint Eastwood)
Update Module: update concept type 'actor' history, update S-list
Main Engine: update (date: today)
Update Module: update concept type 'date' history, update S-list
U: Can you show me more information about this movie? <Points to a movie in the list>
S:
Input: Concepts: (<input: user>) <info: information> <ref: this movie> <deixis: $movie_e$>
Main Interface: translates concepts into some internally used data structure.
Main Engine: detect deixis (<info: information> <ref: this movie> <deixis: $movie_e$>)
Deixis Detection Module: $movie_e$ found.
Main Engine: temporarily update (deixis: $movie_e$)
Update Module: temporarily update concept type 'programs' history, temporarily update S-list

Main Engine: detect & classify reference (info: information)
Reference Detection & Classification Module: *looks up in database?* (info: information; reference: NIL)
Main Engine: temporarily update (info: information)
Update Module: temp update concept type 'info' history, temp update S-list
Main Engine: detect & classify reference (ref: this movie)
Reference Detection & Classification Module: *looks up in database?* (ref: this movie; reference: demonstrative)
Main Engine: detect constraints (<info: information> <ref: this movie> <deixis: $movie_e$>)
Constraints Detection Module: constraints: number: singular, genre: movie
Main Engine: resolve demonstrative (ref: this movie; constraints: number: singular, genre: movie)
Demonstrative Resolution Module: detected definite description properties, resolve definite description (ref: this movie, constraints: number singular, genre: movie)
Definite Description Resolution Module: determine concept types (ref: this movie, constraints: number: singular, genre: movie)
Concept Determiner Module: concept type: programs
Definite Description Resolution Module: look up first compatible in s- list with constraints: number: singular. (programs: this movie; referent: $movie_e$, genre: movie)
Main Engine: temporarily update (programs: this movie)
Update Module: temp update concept type 'programs' history, temp update S-list
Main Engine: translate output (<info: information> <programs: this movie; referent: $movie_e$>)
Output Module: translates data into SPICE readable concepts.
S: Shows information about the movie pointed to by the user
Input: Concepts: (<input: system>) (<best: 1>) <info: information; movie: $movie_e$> **Main Interface**: translates concepts into some internally used data structure.
Main Engine: update (best: 1)
Update Module: save updates from 1st try. Clear temporary updates.
Main Engine: update (…)
Update Module: update concept type '…' history, (update S-list)
U: Could you show me the list again?
S:
Input: Concepts: (<input: user>) <ref: the list>
Main Interface: translates concepts into some internally used data structure.
Main Engine: detect deixis (<ref: the list>)
Deixis Detection Module: no deixis found.
Main Engine: detect & classify reference (ref: the list)
Reference Detection & Classification Module: *looks up in database?* (ref: the list; reference: definite description)
Main Engine: detect constraints (<ref: the list>)
Constraints Detection Module: constraints: number: singular, type: list
Main Engine: resolve definite description (ref: the list; constraints: number: singular, type: list)
Definite Description Resolution Module: determine concept types (ref: the list, constraints: number: singular, type: list)
Concept Determiner Module: concept type: list
Definite Description Resolution Module: look up first compatible in s-list, then in history of concept 'list' with constraints: number: singular, type: list. (list: the list; referent: list 2)
Main Engine: temporarily update (movie: the list)
Update Module: temp update concept type 'movie' history, temp update S-list
Main Engine: translate output (<list: the list; referent: list 2:
Output Module: translates data into SPICE readable concepts.
S: <Shows list with today's movies starring Clint Eastwood>
Input: Concepts: (<input: system>) (<best: 1>) <list: info; programs: movies, genre: movie, date: today > <programs: movies; genre: movie, date: today, [$movie_c$: …], …, [$movie_d$: …]> <genre: movie> <[$movie_c$: …]>, …, <[$movie_d$: …]>
Main Interface: translates concepts into some internally used data structure.
Main Engine: update (best: 1)
Update Module: save updates from 1st try. Clear temporary updates.

Main Engine: update (list: info; …)
Update Module: update concept type 'list' history, (update S-list)
Main Engine: update (programs: movies; …)
Update Module: update concept type 'programs' history, update S-list
Main Engine: update (genre: movie)
Update Module: update concept type 'genre' history, (update S-list)
Main Engine: update (date: this week)
Update Module: update concept type 'date' history, (update S-list)

U: Are there any samurai movies today?
S:
Input: Concepts: (<input: user>) <genre: movie> <topic: samurai> <date: today>
Main Interface: translates concepts into some internally used data structure.
Main Engine: detect deixis (<genre: movie> <topic: samurai> <date: today>)
Deixis Detection Module: no concept tagged as deixis found.
Main Engine: detect & classify reference (genre: movie)
Reference Detection & Classification Module: *looks up in database?* (genre: movie; reference: NIL)
Main Engine: temporarily update (genre: movie)
Update Module: temp update concept type 'genre' history, temp update S-list
Main Engine: detect & classify reference (topic: samurai)
Reference Detection & Classification Module: *looks up in database?* (topic: samurai; reference: NIL)
Main Engine: temporarily update (topic: samurai)
Update Module: temp update concept type 'topic' history, temp update S-list
Main Engine: detect & classify reference (date: today)
Reference Detection & Classification Module: *looks up in database?* (date: today; reference: NIL)
Update Module: temp update concept type 'date' history, temp update S-list
Main Engine: translate output (<programs: movies> <genre: movie> <topic: samurai> <date: today;>)
Output Module: translates data into SPICE readable concepts.
S: <Shows list with samurai movies today>
Input: Concepts: (<input: system>) (<best: 1>) <list: info; programs: movies, genre: movie, topic: samurai, date: today, [$movie_a$: …], …, [$movie_b$: …]> <programs: movies; genre: movie, topic: samurai, date: today, [$movie_a$: …], …, [$movie_b$: …]> <genre: movie> <[$movie_a$: …]>, …, <[$movie_b$: …]>
Main Interface: translates concepts into some internally used data structure.
Main Engine: update (best: 1)
Update Module: save updates from 1st try. Clear temporary updates.
Main Engine: update (list: info; …)
Update Module: update concept type 'list' history, (update S-list)
Main Engine: update (programs: movies; …)
Update Module: update concept type 'programs' history, update S-list
Main Engine: update (genre: movie)
Update Module: update concept type 'genre' history, (update S-list)
Main Engine: update (topic:)
Update Module: update concept type 'actor' history, (update S-list)
Main Engine: update (date: today)
Update Module: update concept type 'date' history, (update S-list)
U: Who is the director of this one? <Points to a movie in the list>
S:
Input: Concepts: (<input: user>) <ref: the director of this one> <deixis: $movie_e$>
Main Interface: translates concepts into some internally used data structure.
Main Engine: detect deixis (<ref: the director> <ref: this one> <deixis: $movie_e$>)
Deixis Detection Module: $movie_e$ found.
Main Engine: temporarily update (deixis: $movie_e$)
Update Module: temp update concept type 'programs' history, temp update S-list
Main Engine: detect & classify reference (ref: the director of this one)
Reference Detection & Classification Module: *looks up in database?* (ref: the director; reference: definite description)

Main Engine: detect constraints: (<ref: the director> <ref: this one>)
Constraint Detection Module: constraints: number: singular *should 'this one' be a constraint and as such be solved first?*
Main Engine: resolve definite description (ref: the director; constraints: number singular)
Definite Description Resolution Module: determine concept types (ref: the director; constraints: number: singular)
Concept Determiner Module: concept types: director, programs.director
Definite Description Resolution Module: found as most recent director $movie_e$.director: Akira Kurasowa; (director: the director; referent: Akira Kurasowa)
Main Engine: temporarily update (director: the director)
Update Module: temp update concept type 'director' history, temp update S-list
Main Engine: detect & classify reference (ref: this one)
Reference Detection & Classification Module: *looks up in database?* (ref: this movie; reference: definite description)
Main Engine: detect constraints (<ref: this one> <ref: the director>)
Constraints Detection Module: constraints: concept type: programs *derived from the director, because only programs have directors*; number: singular
Main Engine: resolve definite description (ref: this one; constraints: concept type: programs; number: singular)
Definite description Resolution Module: detected definite description properties, *because one is limited only to a program, but it is also arguable that it has pronominal properties as it refers to the program in focus.* (ref: this one, constraints: concept type: programs; number: singular)
Definite Description Module: determine concept types (ref: this one, constraints: concept type: programs; number singular)
Concept Determiner Module: concept type: programs
Definite Description Module: look up first compatible in history of concept programs with constraints: number: singular. (programs: this movie; referent: $movie_e$)
Main Engine: temporarily update (programs: this movie)
Update Module: temp update concept type 'movie' history, temp update S-list
Main Engine: translate output (<director: the director; referent: Akira Kurasowa> <movie: this one; referent: $movie_e$>)
Output Module: translates data into SPICE readable concepts.
S: <Shows info on Akira Kurasowa>
Input: Concepts: (<input: system>) (<best: 1>) <info: information; director: Akira Kurasowa> <director: Akira Kurasowa>
Main Interface: translates concepts into some internally used data structure.
Main Engine: update (best: 1)
Update Module: save updates from 1st try. Clear temporary updates.
Main Engine: update (…)
Update Module: update concept type '…' history, (update S-list)
U: Are there any other movies directed by him this month?
S:
Input: Concepts: (<input: user>) <genre: movies> <constraint: directed> <ref: him> <date: this month>
Main Interface: translates concepts into some internally used data structure.
Main Engine: detect deixis (<programs: movies> <constraint: directed> <ref: him> <date: this month>)
Deixis Detection Module: nothing found.
Main Engine: detect & classify reference (programs: movies)
Reference Detection & Classification Module: *looks up in database?* (genre: movies; reference: NIL)
Main Engine: detect & classify reference (constraint: directed)
Reference Detection & Classification Module: *looks up in database?* (constraint: directed; reference: NIL)
Main Engine: detect & classify reference (programs: movies)
Reference Detection & Classification Module: *looks up in database?* (ref: him; reference: pronoun)
Main Engine: detect constraints: (<programs: movies> <constraint: directed> <ref: him> <date: this month>)
Constraint Detection Module: constraints: number: singular, concept type: director *derived from directed*

Main Engine: resolve pronoun (ref: him; constraints: number singular, concept type: director)
Pronoun Resolution Module: looks up fist compatible person within binding constraints up in the Salience list: (director: him; referent: Akira Kurasowa).
Main Engine: temporarily update (director: him)
Update Module: temp update concept type 'director' history, temp update S-list
Main Engine: detect & classify reference (date: this month)
Reference Detection & Classification Module: *looks up in database?* (date: this month; reference: date)
Main Engine: detect constraints (<date: this month> <programs: movies> <constraint: directed> <ref: him>)
Constraints Detection Module: constraints: NIL
Main Engine: resolve date (date: this month; constraints: NIL)
Main Engine: temporarily update (date: this month)
Update Module: temp update concept type 'date' history, temp update S-list
Main Engine: translate output (<programs: movies> <director: him; referent: …> <date: this month>)
Output Module: translates data into SPICE readable concepts.
S: <Shows list of movies this month directed by Akira Kurasowa>
Input: Concepts: (<input: system>) (<best: 1>) <list: info; programs: movies, genre: movie, date: this month, [$movie_a$: …], …, [$movie_b$: …]> <programs: movies; genre: movie, date: this month, [$movie_a$: …], …, [$movie_b$: …]> <genre: movie> <[$movie_a$: …], …, <[$movie_b$: …]>
Main Interface: translates concepts into some internally used data structure.
Main Engine: update (best: 1)
Update Module: save updates from 1st try. Clear temporary updates.
Main Engine: update (list: info; …)
Update Module: update concept type 'list' history, (update S-list)
Main Engine: update (programs: movies; …)
Update Module: update concept type 'programs' history, update S-list
Main Engine: update (genre: movie)
Update Module: update concept type 'genre' history, (update S-list)
Main Engine: update (date: this week)
Update Module: update concept type 'date' history, (update S-list)

U: Are there any movies by Roman Polansky?
S:
Input: Concepts: (<input: user>) <programs: movies> <genre: movie> <director: Roman Polansky>
Main Interface: translates concepts into some internally used data structure.
Main Engine: detect deixis (<programs: movies> <genre: movie> <director: Roman Polansky >)
Deixis Detection Module: no concept tagged as deixis found.
Main Engine: detect & classify reference (programs: movies)
Reference Detection & Classification Module: *looks up in database?* (programs: movies; reference: NIL)
Main Engine: temporarily update (programs: movies)
Update Module: temp update concept type 'programs' history, temp update S-list
Main Engine: detect & classify reference (genre: movie)
Reference Detection & Classification Module: *looks up in database?* (genre: movie; reference: NIL)
Main Engine: temporarily update (genre: movie)
Update Module: temp update concept type 'genre' history, temp update S-list
Main Engine: detect & classify reference (director: Roman Polansky)
Reference Detection & Classification Module: *looks up in database?* (director: Roman Polansky; reference: NIL)
Main Engine: temporarily update (director: Roman Polansky)
Update Module: temp update concept type 'director' history, temp update S-list
Main Engine: translate output (<programs: movies> <genre: movie> <director: Roman Polansky>)
Output Module: translates data into SPICE readable concepts.
S: <Shows list with movies directed by Roman Polansky>

Input: Concepts: (<input: system>) (<best: 1>) <list: info; programs: movies, genre: movie, [$movie_a$: …], …, [$movie_b$: …]> <programs: movies; genre: movie, [$movie_a$: …], …, [$movie_b$: …]> <genre: movie> <[$movie_a$: …]>, …, <[$movie_b$: …]>
Main Interface: translates concepts into some internally used data structure.
Main Engine: update (best: 1)
Update Module: save updates from 1st try. Clear temporary updates.
Main Engine: update (list: info; …)
Update Module: update concept type 'list' history, (update S-list)
Main Engine: update (programs: movies; …)
Update Module: update concept type 'programs' history, update S-list
Main Engine: update (genre: movie)
Update Module: update concept type 'genre' history, (update S-list)
Main Engine: update (director: Roman Polansky)
Update Module: update concept type 'director' history, (update S-list)
U: In which of these does he star himself?
S:
Input: Concepts: (<input: user>) <ref: these> <ref: he> <actor: himself>
Main Interface: translates concepts into some internally used data structure.
Main Engine: detect deixis (<ref: these> <ref: he> <actor: himself>)
Deixis Detection Module: nothing found.
Main Engine: detect & classify reference (ref: these)
Reference Detection & Classification Module: *looks up in database?* (ref: these; reference: demonstrative)
Main Engine: detect constraints: (<ref: these> <ref: he> <actor: himself>)
Constraint Detection Module: constraints: number: plural
Main Engine: resolve demonstrative (ref: these; constraints: number: plural)
Demonstrative Resolution Module: pronominal properties detected (ref: these; constraints: number: plural)
Pronoun Resolution Module: looks up in the S-list. <programs: these; referent: programs: movies; genre: movie, director: Roman Polansky, [$movie_a$: …], …, [$movie_b$: …]> should be returned.
Main Engine: temporarily update <programs: movies>
Update Module: temp update concept type 'programs' history, temp update S-list
Main Engine: detect & classify reference (ref: he)
Reference Detection & Classification Module: *looks up in database?* (ref: he; reference: pronoun)
Main Engine: detect constraints: (<ref: these> <ref: he> <actor: himself>)
Constraint Detection Module: constraints: number: singular, gender: male, person: person
Main Engine: resolve pronoun (ref: he; constraints: number: singular, gender: male, person: person)
Pronoun Resolution Module: looks up in the S-list, should return (director: he; referent: director: Roman Polansky)
Main Engine: temporarily update <director: he>
Update Module: temp update concept type 'director' history, temp update S-list
Main Engine: detect & classify reference (actor: himself)
Reference Detection & Classification Module: *looks up in database?* (actor: himself; reference: pronoun)
Main Engine: detect constraints: (<ref: these> <ref: he> <actor: himself>)
Constraint Detection Module: constraints: number: singular, gender: male, person: person, mode: reflective
Main Engine: resolve pronoun (actor: himself; constraints: number: singular, gender: male, person: person, mode: reflective)
Pronoun Resolution Module: Looks up in the S-list, should return (actor: himself; referent: director: he; referent: Roman Polansky)
Main Engine: temporarily update (actor: himself)
Update Module: temp update concept type 'actor' history, temp update S-list
Main Engine: translate output (<programs: these; referent: programs: movies> <director: he; referent: director: Roman Polansky> <actor: himself; referent: director: Roman Polansky)
Output Module: translates data into SPICE readable concepts.
S: <Shows list with movies directed by Roman Polansky with actor Roman Polansky>

Input: Concepts: (<input: system>) (<best: 1>) <list: info; programs: movies, genre: movie, [$movie_a$: …], …, [$movie_b$: …]> <programs: movies; genre: movie, [$movie_a$: …], …, [$movie_b$: …]> <genre: movie> <[$movie_a$: …]>, …, <[$movie_b$: …]>
Main Interface: translates concepts into some internally used data structure.
Main Engine: update (best: 1)
Update Module: save updates from 1st try. Clear temporary updates.
Main Engine: update (list: info; …)
Update Module: update concept type 'list' history, (update S-list)
Main Engine: update (programs: movies; …)
Update Module: update concept type 'programs' history, update S-list
Main Engine: update (genre: movie)
Update Module: update concept type 'genre' history, (update S-list)

Appendix E

Source Code

This appendix lists the source code of each of the classes used in the reference resolution module.

Main Interface

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: pinterface.cc
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: May 10, 2001
//
//  Description: This module reads and translates SPICE parse data into Anaphora Module
data.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include <set>
#include <fstream>
#include <iostream>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include "myUtils.h"
#include "maineng.h"
#include "displread.h"

using namespace std;

set <string> ignoreList; // list of concept types to ignore.
set <string> subConceptList; // list of subconcepts.
set <string> listEntriesList; // list of list entries tags.
set <string> valueList; //list of value tags.
set <string> superConceptList; //list of super concept tags.
set <string> thisConceptList; //list of this concept tags.
//FILE *df;
vector <DSConcept> screen;
vector <string> newcontentvalue; //list of values to be filtered out.
vector <string> newcontenttype; //list of types to replace the content_type.
DSMainEngine mainEngine;
int req_reader;
ifstream in;

/**
```

```
 * Initialize all variables.
 */

void init()
{
  myUtils util;
  string tmp, type, value, strTime, inputOrigin;
  int timestamp, subTimestamp;
  vector<DSConcept> *subConList, *listEntries, *subSubConList;

  // load files which contain filter data.
  //ifstream in;

  cout << "loading concept types to ignore..." << endl;
  in.open("ignore.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open ignore list data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#")!=string::npos || tmp=="") // comment read.
     {
        //cout << "comment: " << tmp << endl;
     }
     else
     {
        ignoreList.insert(tmp);
     }
  }
  in.close();
  cout << "loading subconcept indicators..." << endl;
  in.open("subCon.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open subconcepts data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#")!=string::npos || tmp=="") // comment read.
     {
        //cout << "comment: " << tmp << endl;
     }
     else
     {
        subConceptList.insert(tmp);
     }
  }
  in.close();
  cout << "loading list entry indicators..." << endl;
  in.open("listEntries.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open list entries data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#")!=string::npos || tmp=="") // comment read.
     {
        //cout << "comment: " << tmp << endl;
     }
     else
     {
        listEntriesList.insert(tmp);
     }
```

```
}
in.close();
cout << "loading superconcept indicators..." << endl;
in.open("superconcept.txt", ios::in);
if (!in)
{
   cerr << "Cannot open superconcept tag data file" << endl;
   exit;
}
while (!in.eof())
{
   getline(in, tmp);
   if (tmp.find("#")!=string::npos || tmp=="") // comment read.
   {
      //cout << "comment: " << tmp << endl;
   }
   else
   {
      superConceptList.insert(tmp);
   }
}
in.close();
cout << "loading this concept indicators..." << endl;
in.open("thisconcept.txt", ios::in);
if (!in)
{
   cerr << "Cannot open thisconcept tag data file" << endl;
   exit;
}
while (!in.eof())
{
   getline(in, tmp);
   if (tmp.find("#")!=string::npos || tmp=="") // comment read.
   {
      //cout << "comment: " << tmp << endl;
   }
   else
   {
      thisConceptList.insert(tmp);
   }
}
in.close();
cout << "loading value indicators..." << endl;
in.open("value.txt", ios::in);
if (!in)
{
   cerr << "Cannot open value list data file" << endl;
   exit;
}
while (!in.eof())
{
   getline(in, tmp);
   if (tmp.find("#")!=string::npos || tmp=="") // comment read.
   {
      //cout << "comment: " << tmp << endl;
   }
   else
   {
      valueList.insert(tmp);
   }
}
in.close();
/*  cout << "loading pipe number..." << endl;
in.open("reqReader.txt", ios::in);
if (!in)
{
   cerr << "Cannot open required reader data file" << endl;
   exit;
}
while (!in.eof())
{
```

```
      getline(in, tmp);
      if (tmp.find("#")!=string::npos || tmp=="") // comment read.
      {
        //cout << "comment: " << tmp << endl;
      }
      else
      {
        req_reader = util.str2Int(tmp);
      }
  }
  in.close();*/
  in.open("actordirector.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open actor director data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#")!=string::npos || tmp=="") // comment read.
     {
       //cout << "comment: " << tmp << endl;
     }
     else
     {
       newcontenttype.push_back(tmp.substr(0,tmp.find(",")));
       newcontentvalue.push_back(tmp.substr(tmp.find(",")+2));
     }
  }
  in.close();
  in.open("/tmp/dwoei/logfiles/log.lst", ios::in);
  if(!in)
  {
    cerr << "Cannot open user input data" << endl;
    exit;
  }
}

void filterAndReplace(string &value, string filter, string replace)
{
  if (value.find(filter)!=string::npos)
  {
    cout << value << " is changed into ";
    value.replace(value.find(filter), value.find(filter) + filter.size(), replace);
    cout << value << endl;
  }
}


/**
 * Read from the pipe, create a concept list and send it to the main engine.
 */

void processData()
{
  string tmp, tag, type, value, subconceptType, subconceptValue, pointingvalue,
superconcept, thisconcept;
  string text;
  vector<DSConcept> conList, *subConList;
  int conNr = -1;
  int timestamp = 0;
  bool ignore, deixis;
  vector<DSConcept>::iterator it;
  vector<string> deixisvalue, deixistype;
  DSDisplayReader displayReader;
  vector<string> noise;
  myUtils util;

  noise.push_back("<3:>");
  noise.push_back("<3:r>");
```

```
noise.push_back("<@U>");
noise.push_back("<@h>");
noise.push_back("<@h@>");
noise.push_back("<@h@h>");
noise.push_back("<@hA:>");
noise.push_back("<A:>");
noise.push_back("<A:h>");
noise.push_back("<A:hA:>");
noise.push_back("<E>");
noise.push_back("<I>");
noise.push_back("<Ik>");
noise.push_back("<OI>");
noise.push_back("<Q>");
noise.push_back("<Uf>");
noise.push_back("<Vf>");
noise.push_back("<Vg>");
noise.push_back("<Vh>");
noise.push_back("<VhV>");
noise.push_back("<VhVh>");
noise.push_back("<Vm>");
noise.push_back("<h@mf>");
noise.push_back("<hV>");
noise.push_back("<hVmf>");
noise.push_back("<hm>");
noise.push_back("<i:k>");
noise.push_back("<mhm>");
noise.push_back("<mm>");
noise.push_back("<sh>");
noise.push_back("<u:>");
noise.push_back("<u:f>");
noise.push_back("<w@U>");
noise.push_back("<waU>");
noise.push_back("<wi:>");
noise.push_back("<wu:>");
noise.push_back("#PAUSE#");
noise.push_back("<dZu:lIA:>");

cout << "updating screen info" << endl;
mainEngine.handleConcepts(displayReader.readDisplayContent(timestamp));

subConList = NULL;
while (true)
{
   cout << "reading conceptgraph" << endl;
   cerr << "waiting for user input..." << endl;
   while(true)
   {
     //cerr << "waiting... " << endl;
     // if (!in.eof()) //wait until able to read from file
     //{
        getline(cin,tmp);
        //cerr << tmp << endl;

        if (tmp.find("BEGIN_LATTICE") != string::npos)
        {
          cout << tmp << endl;
          break;
        }
        //    }
   }
   cerr << "user input received..." << endl;
   while(true)
   {
     //if (!in.eof()) //wait until able to read from file
     //{
      getline(cin, tmp);
     for(int i=0; i < noise.size(); i++)
     {
       while (tmp.find(noise[i]+" ")!=string::npos)
       {
         tmp.erase(tmp.find(noise[i]+" "), noise[i].size());
```

```
          }
          while (tmp.find(noise[i])!=string::npos)
          {
            tmp.erase(tmp.find(noise[i]), noise[i].size());
          }
        }
        cout << "done filtering noise" << endl;
        for (int i=0; i < tmp.size(); i++) // remove all spaces in front
        {
          if (tmp[i]!=' ' && tmp[i]!='\t')
          {
            tmp = tmp.substr(i);
            break;
          }
        }
        for (int i=tmp.size()-1; i>0; i--) //remove all spaces at the end
        {
          if (tmp[i]!=' ' && tmp[i]!='\t')
          {
            tmp = tmp.substr(0,i+1);
            break;
          }
        }
        while (tmp.find("  ")!=string::npos) //remove all double spaces
        {
          tmp.erase (tmp.find("  "), 1);
        }
        cout << tmp << endl;
        if (tmp.find("@")!=string::npos) // concept type read
        {
           if (conNr != -1) // not first concept
           {
              if (!ignore)
              {
               if (type == "contents")
                {
                  for (int k=0; k < newcontentvalue.size(); k++)
                  {
                    if (value.find(newcontentvalue[k])!=string::npos)
                    {
                      if (value.find("and")!=string::npos)
                      {
                       for (int l=k+1; l < newcontentvalue.size(); l++)
                       {
                         if (value.find(newcontentvalue[l])!=string::npos)
                         {
                           value = newcontentvalue[k] + " and " + newcontentvalue[l];
                           type = newcontenttype[k];
                           break;
                         }
                       }
                      }
                      else
                      {
                       value = newcontentvalue[k];
                       type = newcontenttype[k];
                       break;
                      }
                    }
                  }
                }
                cout << conNr << ": adding concept type:(" << type << ") value:(" <<
value << ") timestamp:(" << timestamp << ")" << endl;
                conList.push_back(DSConcept(type,value,timestamp));
                timestamp++;
                conList[conNr].setText(text);
                conList[conNr].setInputOrigin("user");
                cout << conNr << ": " << conList[conNr].getInputOrigin() << endl;
                conList[conNr].setSubConcepts(subConList);
                subConList = NULL;
                if (superconcept!="")
```

```
                {
                  conList[conNr].setSuperConcept(superconcept);
                  superconcept = "";
                }
                if (thisconcept!="")
                {
                  conList[conNr].setConcept(thisconcept);
                  thisconcept = "";
                }
                conNr++;
                if (deixis)
                {
                   for (int d=0; d < deixisvalue.size(); d++)
                   {
                      if(deixistype[d]!=deixistype[0]&&deixisvalue[d]!=deixisvalue[0])
                      {

conList.push_back(DSConcept(deixistype[d],deixisvalue[d],timestamp));
                       conList[conNr].setInputOrigin("deixis");
                       conNr++;
                      }
                      if(deixisvalue[d]==deixisvalue[0]&&deixistype[0]=="")
                      {
                        conList[0].setType(deixistype[d]);
                      }
                   }
                }
             }
             ignore = false;
             deixis = false;
             deixistype.clear();
             deixisvalue.clear();
          }
          else
          {
             cout << "first concept read" << endl;
             conNr++;
          }
          text = tmp + "\n";
          tmp = tmp.substr(tmp.find("@")+1);
          tmp = tmp.substr(0, tmp.find(" "));
          cout << "concept type '" << tmp << "' read" << endl;
          if (ignoreList.count(tmp)==0) // do not ignore the concept type
          {
             type = tmp;
          }
          else
          {
             cout << "ignore concept" << endl;
             ignore = true;
          }
       }
       else if (tmp.find("END_LATTICE") != string::npos)
       {
          if (!ignore && conNr!=-1)
          {
            if (type == "contents")
                {
                  for (int k=0; k < newcontentvalue.size(); k++)
                  {
                    if (value.find(newcontentvalue[k])!=string::npos)
                    {
                      if (value.find("and")!=string::npos)
                      {
                       for (int l=k+1; l < newcontentvalue.size(); l++)
                       {
                         if (value.find(newcontentvalue[l])!=string::npos)
                         {
                           value = newcontentvalue[k] + " and " + newcontentvalue[l];
                           type = newcontenttype[k];
                           break;
```

```
                        }
                      }
                    }
                    else
                    {
                      value = newcontentvalue[k];
                      type = newcontenttype[k];
                      break;
                    }
                  }
                }
              }
            cout << conNr << ": adding concept type:(" << type << ") value:(" << value
<< ") timestamp:(" << timestamp << ")" << endl;
            conList.push_back(DSConcept(type,value,timestamp));
            conList[conNr].setText(text);
            conList[conNr].setInputOrigin("user");
            cout << conNr << ": " << conList[conNr].getInputOrigin() << endl;
            conList[conNr].setSubConcepts(subConList);
            subConList = NULL;
            if (superconcept!="")
            {
              conList[conNr].setSuperConcept(superconcept);
              superconcept = "";
            }
            if (thisconcept!="")
            {
              conList[conNr].setConcept(thisconcept);
              thisconcept = "";
            }
            if (deixis)
            {
               for (int d=0; d < deixisvalue.size(); d++)
               {
                  conNr++;
                  conList.push_back(DSConcept(deixistype[d],deixisvalue[d],timestamp));
                  conList[conNr].setInputOrigin("deixis");
               }
               deixistype.clear();
               deixisvalue.clear();
            }
          }
          timestamp++;
          conNr = -1;
          ignore = false;
          deixis = false;
          cout << "starting main engine" << endl;
          if (conList.size() == 0)
          {
            conList.push_back(DSConcept("dummy","dummy",timestamp)); //create dummy
concept.
            conList[0].setInputOrigin("user");
          }
          mainEngine.handleConcepts(conList);
          cout << "updating screen info" << endl;
          mainEngine.handleConcepts(displayReader.readDisplayContent(timestamp));
          conList.clear();
          break;
        }
        else if (!ignore)
        {
           text = text + tmp +"\n";
           for(int i=0; i < tmp.size(); i++)
           {
              if (tmp[i] != ' ' && tmp[i]!='\t') //remove all spaces add the beginning of
the line
              {
                tmp = tmp.substr(i);
                cout << tmp << endl;
                break;
              }
```

```
            }
            tag = tmp.substr(0,tmp.find(" "));
            if (tag == "title")
            {
               tag = "programme";
            }
            cout << "tag is: " << tag << endl;
            tmp = tmp.substr(tmp.find(" ")+1);
            for(int i=0; i < tmp.size(); i++)
            {
              if (tmp[i]!=' ' && tmp[i]!='\t') //remove all spaces add the beginning of the
line
              {
                tmp = tmp.substr(i);
                break;
              }
            }
            if (subConceptList.count(tag) != 0)
            {
               cout << "subconcept tag" << endl;
               bool subconceptfound = false;
               //             tmp = tmp.substr(tmp.find(" ")+1);
               if (tmp!="-none-")
                 subconceptfound = true;
               if (subconceptfound)
               {
                  cout << "subconcept found" << endl;
                  subconceptType = tmp.substr(0,tmp.find(","));
                  cout << "subconcept type: " << subconceptType << endl;
                  tmp = tmp.substr(tmp.find(",")+2);
                  subconceptValue = tmp.substr(0,tmp.find(","));
                  cout << "subconcept value: " << subconceptValue << endl;
                  if (subconceptType == "start time" || subconceptType == "end time")
                  {
                    subconceptValue =
util.int2Str(util.str2Int(subconceptValue)/60)+":"+((util.str2Int(subconceptValue)%60)<10
?"0"+util.int2Str(util.str2Int(subconceptValue)%60):util.int2Str(util.str2Int(subconceptV
alue)%60));
                  }
                  if (subConList == NULL)
                  {
                     subConList = new vector<DSConcept>;
                  }
                  subConList->push_back(DSConcept(subconceptType, subconceptValue,
timestamp));
                  cout << "added subconcept to the list" << endl;
                  it = subConList->end();
                  it--;
                  it->setInputOrigin("user");
                  cout << "set subconcept input origin" << endl;
               }
            }
            else if (listEntriesList.count(tag) != 0)
            {
              // create a listentry for the concept. Currently not implemented in grammar.
            }
            else if (superConceptList.count(tag)!= 0)
            {
              if (tmp != "-none-")
                superconcept = tmp;
            }
            else if (thisConceptList.count(tag)!= 0)
            {
              if (tmp != "-none-")
                thisconcept = tmp;
            }
            else if (valueList.count(tag) != 0)
            {
               if (tmp.find("[")!=string::npos) // if deictic input
               {
                  cout << "add deixis value" << endl;
```

```
                deixisvalue.push_back(tmp.substr(tmp.find("[")+1,
(tmp.find(":")!=string::npos?tmp.find(":"):tmp.find("]")) - tmp.find("[") -1));
                cout << "add deixis type" << endl;
                deixistype.push_back((tmp.find(":")!=string::npos ?
tmp.substr(tmp.find(":")+1,tmp.find("]")-tmp.find(":")-1):""));
                if (deixistype[deixistype.size()-1] == "title")
                {
                   deixistype[deixistype.size()-1] = "program";
                }
                deixis = true;
                pointingvalue = deixisvalue[deixisvalue.size()-1];
                cout << "deixis type: " << deixistype[deixistype.size()-1] << " deixis
value: " << deixisvalue[deixisvalue.size()-1] << endl;
                tmp = tmp.substr(0,tmp.find("[")-1);
             }
             /*tmp = tmp.substr(tmp.find(" ")+1);
             for(int i=0; i < tmp.size(); i++)
             {
                if (tmp[i]!=' ' && tmp[i]!='\t') //remove all spaces add the beginning of
the line
                {
                   tmp = tmp.substr(i);
                   break;
                }
             }*/
             cout << "concept value: " << tmp << endl;
             value = tmp;
             //filter out 'on channel 5'/'on channel 4'/'on discovery channel'
             filterAndReplace(value, "on channel 5", "on channel5");
             filterAndReplace(value, "on channel 4", "on channel4");
             filterAndReplace(value, "on discovery channel", "on discoverychannel");
          }
          else if (deixis)
          {
            /*tmp = tmp.substr(tmp.find(" ")+1);
             for(int i=0; i < tmp.size(); i++)
             {
                if (tmp[i]!=' ' && tmp[i]!='\t') //remove all spaces add the beginning of
the line
                {
                   tmp = tmp.substr(i);
                   break;
                }
                }*/
             cout << "concept value: " << tmp << endl;
             if (pointingvalue==tmp)
             {
                for (int i=0; i < deixisvalue.size(); i++)
                {
                   if (deixisvalue[i]==pointingvalue)
                     if (tag != deixistype[i])
                       deixistype[i]=tag;
                }
             }
             else
             {
                deixisvalue.push_back(tmp);
                deixistype.push_back(tag);
             }
          }
       }
       //       }
    }
  }
}

int main()
{
  init();

  processData();
```

```
}
```

Display Reader

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: displread.h
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This module reads the display information
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DISPLAYREADER_H
#define DISPLAYREADER_H

#include <set>
#include <string>
#include <vector>
#include "concept.h"
#include <fstream>
#include <iostream>

class DSDisplayReader
{
public:

   /**
    * Constructor.
    */

   DSDisplayReader();

   /**
    * Destructor.
    */

   ~DSDisplayReader();

   /**
    * Creates a concept from a list of concepts, which can be referred to pronominally.
    * @param conlist Concept list, which contains the list of concepts, which can be
pronominally referred to.
    * @return A concept list, expanded with the concept, which can be pronominally
referred to.
    */

   vector<DSConcept> readDisplayContent(int &timestamp);

protected:
```

```
   // none

private:
   int timestamp;
   ifstream in;
};

#endif //DISPLAYREADER_H
```

Implementation File

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: displread.cc
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This module reads the display information
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "displread.h"
#include "myUtils.h"

//public:

   /**
    * Constructor.
    */

DSDisplayReader::DSDisplayReader()
{

  in.open("/tmp/philips/dispcont.log", ios::in);
  if (!in)
  {
     cerr << "Cannot open  file" << endl;
     exit;
  }
}

   /**
    * Destructor.
    */

DSDisplayReader::~DSDisplayReader()
{
  in.close();
}

   /**
    * Creates a concept from a list of concepts, which can be referred to pronominally.
    * @param conlist Concept list, which contains the list of concepts, which can be
pronominally referred to.
```

```
    * @return A concept list, expanded with the concept, which can be pronominally
referred to.
    */

vector<DSConcept> DSDisplayReader::readDisplayContent(int &timestamp)
{
  string tmp, substring;
  vector<DSConcept> res, *dateEntries, *startEntries, *endEntries, *channelEntries,
*titleEntries, *categoryEntries, *subConcepts;
  DSConcept *date, *start_time, *end_time, *channel, *title, *category, list, datelist,
startlist, endlist, channellist, categorylist;

  myUtils util;
  dateEntries = NULL;
  startEntries = NULL;
  endEntries = NULL;
  channelEntries = NULL;
  titleEntries = NULL;
  categoryEntries = NULL;
  subConcepts = new vector<DSConcept>;
  date = NULL;
  start_time = NULL;
  end_time = NULL;
  channel = NULL;
  title = NULL;
  category = NULL;

  std::streampos readpos = in.tellg();

  if (timestamp != 0) // no best when timestamp==0, because there is nothing to save
  {
    res.push_back(DSConcept("best" , "1" ,timestamp));
    res[0].setInputOrigin("system");
  }
  cerr << "waiting for screen info..." << endl;
  while (true)
  {
    //cerr << "waiting..." << endl;
     if (in.beg == in.end)
     {
     }
     else if (in.eof())
     {
       in.close();
       in.open("/tmp/philips/dispcont.log", ios::in);
       if (!in)
       {
        cerr << "Cannot open  file" << endl;
        exit;
       }
       //in.clear(in.rdstate() & ~std::ios::eofbit); //clear the eofbit
       //if (!in.eof())
       //       cerr << "eofbit cleared" << endl;
       in.seekg(readpos);
     }
     else
     {
       readpos = in.tellg();
       getline(in, tmp);
       if (tmp=="") // comment read.
       {
          //cout << "comment: " << tmp << endl;
       }
       else
       {
          if (tmp.find("<ITEMLIST")!=string::npos) //start of itemlist
          {
             tmp = tmp.substr(tmp.find(" name=") +7);
             string name = tmp.substr(0,tmp.find("\""));
             cout << "   name of list: " << name << endl;
             list = DSConcept("list", name+" " + util.int2Str(timestamp), timestamp);
```

```
            list.setInputOrigin("system");
            datelist = DSConcept("date list", "date list " + util.int2Str(timestamp),
timestamp);
            datelist.setInputOrigin("system");
            startlist = DSConcept("start time list", "start time list " +
util.int2Str(timestamp), timestamp);
            startlist.setInputOrigin("system");
            endlist = DSConcept("end time list", "end time list " +
util.int2Str(timestamp), timestamp);
            endlist.setInputOrigin("system");
            channellist = DSConcept("channel list", "channel list " +
util.int2Str(timestamp), timestamp);
            channellist.setInputOrigin("system");
            categorylist = DSConcept("category list", "category list " +
util.int2Str(timestamp), timestamp);
            categorylist.setInputOrigin("system");
            //cout << "done processing itemlist" << endl;
         }
         if (tmp.find("<PROGITEM>")!=string::npos) //start of a program item
         {
           subConcepts = new vector<DSConcept>;
           date = NULL;
           start_time = NULL;
           end_time = NULL;
           channel = NULL;
           title = NULL;
           category = NULL;
         }
         if (tmp.find("<DATE>")!=string::npos)
         {
            substring = tmp.substr(tmp.find("<DATE>")+6);
            substring = substring.substr(0,substring.find("</DATE>"));
            date = new DSConcept("date", substring, timestamp);
            date->setInputOrigin("system");
            //cout << "done processing date" << endl;
         }
         if (tmp.find("<START_TIME>")!=string::npos)
         {
            substring = tmp.substr(tmp.find("<START_TIME>")+12);
            substring = substring.substr(0,substring.find("</START_TIME>"));
            start_time = new DSConcept("start time", substring, timestamp);
            start_time->setInputOrigin("system");
            //cout << "done processing start time" << endl;
         }
         if (tmp.find("<END_TIME>")!=string::npos)
         {
            substring = tmp.substr(tmp.find("<END_TIME>")+10);
            substring = substring.substr(0,substring.find("</END_TIME>"));
            end_time = new DSConcept("end time", substring, timestamp);
            end_time->setInputOrigin("system");
            //cout << "done processing end time" << endl;
         }
         if (tmp.find("<CHANNEL>")!=string::npos)
         {
            substring = tmp.substr(tmp.find("<CHANNEL>")+9);
            substring = substring.substr(0,substring.find("</CHANNEL>"));
            channel = new DSConcept("channel", substring, timestamp);
            channel->setInputOrigin("system");
            //cout << "done processing channel" << endl;
         }
         if (tmp.find("<TITLE>")!=string::npos)
         {
            substring = tmp.substr(tmp.find("<TITLE>")+7);
            substring = substring.substr(0,substring.find("</TITLE>"));
            title = new DSConcept("programme", substring, timestamp);
            title->setInputOrigin("system");
            //cout << "done processing title" << endl;
         }
         if (tmp.find("<CATEGORY>")!=string::npos)
         {
            substring = tmp.substr(tmp.find("<CATEGORY>")+10);
```

```
      substring = substring.substr(0,substring.find("</CATEGORY>"));
      category = new DSConcept("category", substring, timestamp);
      category->setInputOrigin("system");
      //cout << "done processing category" << endl;
   }
   if (tmp.find("</PROGITEM>")!=string::npos) //end of program item
   {
      //cout << "/progitem tag found" << endl;
      if (date != NULL)
      {
        // cout << "date != NULL" << endl;
        if (dateEntries == NULL)
        {
           //cout << "dateEntries == NULL" << endl;
           dateEntries = new vector<DSConcept>;
        }
        //cout << "date going to be pushed back" << endl;
        dateEntries->push_back(*date);
        //cout << "date pushed back" << endl;
        if (date->getValue()!= "--")
          subConcepts->push_back(*date);
        //cout << "done updating date" << endl;
      }
      if (start_time != NULL)
      {
        if (startEntries == NULL)
        {
           startEntries = new vector<DSConcept>;
        }
        startEntries->push_back(*start_time);
        if (start_time->getValue()!= "--")
          subConcepts->push_back(*start_time);
        //cout << "done updating start time" << endl;
      }
      if (end_time != NULL)
      {
        if (endEntries == NULL)
        {
           endEntries = new vector<DSConcept>;
        }
        endEntries->push_back(*end_time);
        if (end_time->getValue()!= "--")
          subConcepts->push_back(*end_time);
        //cout << "done updating end time" << endl;
      }
      if (channel != NULL)
      {
        if (channelEntries == NULL)
        {
           channelEntries = new vector<DSConcept>;
        }
        channelEntries->push_back(*channel);
        if (channel->getValue()!= "--")
          subConcepts->push_back(*channel);
        //cout << "done updating channel" << endl;
      }
      if (category != NULL)
      {
        if (categoryEntries == NULL)
        {
           categoryEntries = new vector<DSConcept>;
        }
        categoryEntries->push_back(*start_time);
        if (category->getValue()!= "--")
          subConcepts->push_back(*start_time);
        //cout << "done updating category" << endl;
      }
      if (title != NULL)
      {
        if (titleEntries == NULL)
        {
```

```
                titleEntries = new vector<DSConcept>;
              }
              title->setSubConcepts(subConcepts);
              titleEntries->push_back(*title);
              //cout << "done updating title" << endl;
            }
          }
          if (tmp.find("</ITEMLIST>")!=string::npos) //end of itemlist
          {
             if (titleEntries != NULL)
               list.setListEntries(titleEntries);
             if (dateEntries != NULL)
               datelist.setListEntries(dateEntries);
             if (startEntries != NULL)
               startlist.setListEntries(startEntries);
             if (endEntries != NULL)
               endlist.setListEntries(endEntries);
             if (channelEntries != NULL)
               channellist.setListEntries(channelEntries);
             if (categoryEntries != NULL)
               categorylist.setListEntries(categoryEntries);
             //cout << "end of itemlist" << endl;
             break;
          }
       }
    }
  }
  res.push_back(list);
  res.push_back(datelist);
  res.push_back(startlist);
  res.push_back(endlist);
  res.push_back(channellist);
  res.push_back(categorylist);
  timestamp++;
  cerr << "screen info read" << endl;
  return res;
}

//protected:
   // none

//private:
   // none
```

Main Engine

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: maineng.h
//  Last changed by:
//  Last changed on:
//
```

```
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the main engine where the commands are given to detect and solve
the
//               different types of references and update the different data lists, which
are used
//               to solve them.
//               The following steps are taken depending on whether the data is from the
user or
//               from the system.
//               User input:
//               - Detect deictic input
//               - Temporarily update the data lists with the information from the
deictic input
//               - Detect and Classify each concept for referential properties
//               - Detect constraints
//               - Match referent with reference
//               - Temporarily update the datalists
//               - Generate output
//               System generated output:
//               - Save the correct data lists
//               - Update the datalists with presented system output
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef MAINENGINE_H
#define MAINENGINE_H

#include "concept.h"
#include "constr.h"

#include "update.h"
#include "deixdet.h"
#include "elldet.h"
#include "ellres.h"
#include "refdet.h"
#include "condet.h"
#include "pronres.h"
#include "demres.h"
#include "defres.h"
#include "oneres.h"
#include "dateres.h"
#include "listproc.h"

#include <set>
#include <iostream>
#include <fstream>

/**
 * MainEngine updates data lists, and resolves references.
 * Here the commands are given to detect and solve the different types of references and
update the
 * different data lists, which are used to solve them.
 * The following steps are taken depending on whether the data is from the user or from
the system.
 * User input:
 * - Detect deictic input.
 * - Temporarily update the data lists with the information from the deictic input.
 * - Detect ellipsis.
 * - Solve ellipsis.
 * - Detect and Classify each concept for referential properties.
 * - Detect constraints.
 * - Match referent with reference.
 * - Temporarily update the datalists.
 * - Generate output.
 * System generated output:
 * - Save the correct data lists.
 * - Update the datalists with presented system output.
 */
```

```
class DSMainEngine
{
   public:

   /**
    * Constructor.
    */

   DSMainEngine();

   ~DSMainEngine();

   /**
    * Detects and classifies the concept, using the info of the concept and the
conceptlist.
    * @param concept The concept.
    * @param concepts The concept list.
    */

   void detectAndResolve(DSConcept &concept, DSConcept *superconcept, vector<DSConcept>
&concepts);

   /**
    * handleConcpets divides input into user and system input. User input is handled
further by handleUserInput, system input by
    * handleSystemInput.
    * @param concepts List of concepts to be handled.
    */

     void handleConcepts (vector<DSConcept> concepts);

    /**
     * Handle system generated output.
     * - Save the correct data lists.
     * - Update the datalists with presented system output.
     * @param concepts List of concepts representing system generated output.
     */

     void handleSystemInput(vector<DSConcept> concepts);

    /**
     * Handle user input.
     * - Detect deictic input.
     * - Temporarily update the data lists with the information from the deictic input.
     * - Detect and Classify each concept for referential properties.
     * - Detect constraints.
     * - Match referent with reference.
     * - Temporarily update the datalists.
     * - Generate output.
     * @param concepts List of concepts representing user input.
     */

     void handleUserInput(vector<DSConcept> concepts);

  protected:
   //none
  private:

     // declare the different modules
   DSUpdate updateModule;
   DSDeixisDetection deixisDetectionModule;
   DSReferenceDetectionAndClassification referenceDetectionAndClassificationModule;
   DSPronounResolution pronounResolutionModule;
   DSDemonstrativeResolution demonstrativeResolutionModule;
   DSDefiniteDescriptionResolution definiteDescriptionResolutionModule;
   DSOneAnaphoraResolution oneAnaphoraResolutionModule;
   DSDateResolution dateResolutionModule;
   DSConstraintDetection constraintDetectionModule;
   DSSystemListProcessor systemListProcessor;
```

```
   // declare the lists
   DS_SList sList;
   DSHistoryList historyList;
   DSTypeHisList typeHisList;

   // declare sets containing specific information
   set<string> best;
   set<string> systemInput;

   /**
    * Determines whether the concepts are derived from user input or system output. The
concept which contains this information
    * is removed from the list during this test.
    * @param concepts List of concepts, one concept contains information about the input.
    * @return 0 = user, 1 = system
    */

     int determineInput(vector<DSConcept> concepts);

    /**
     * Determines the best hypothesis. This is derived from a concept in the concept
list. This concept is removed from the
     * list during the test.
     * @param concepts List of concepts, one concept contains information about the
input.
     * @return The best hypothesis.
     */

     int getBest(vector<DSConcept> &concepts);
     ofstream out;
};

#endif //MAINENGINE_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: maineng.cc
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the main engine where the commands are given to detect and solve
the
//               different types of references and update the different data lists, which
are used
//               to solve them.
//               The following steps are taken depending on whether the data is from the
user or
//               from the system.
//               User input:
//               - Detect deictic input
//               - Temporarily update the data lists with the information from the
deictic input
```

```
//               - Detect and Classify each concept for referential properties
//               - Detect constraints
//               - Match referent with reference
//               - Temporarily update the datalists
//               - Generate output
//               System generated output:
//               - Save the correct data lists
//               - Update the datalists with presented system output
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "maineng.h"

#include <stdlib.h>
#include <stddef.h>
// #include <math.h>


//   public:

/**
 * Constructor.
 */

DSMainEngine::DSMainEngine()
{
  // set the history lists for all the modules.
  updateModule.setLists(&historyList, &typeHisList, &sList);
  ellipsisResolutionModule.setList(&historyList);
  pronounResolutionModule.setList(&sList);
  definiteDescriptionResolutionModule.setList(&typeHisList);
  // set the resolution modules to access from other modules.
  demonstrativeResolutionModule.setModules(&pronounResolutionModule,
&definiteDescriptionResolutionModule);
  oneAnaphoraResolutionModule.setModules(&demonstrativeResolutionModule,
&definiteDescriptionResolutionModule);
  definiteDescriptionResolutionModule.setModules(&pronounResolutionModule,
&constraintDetectionModule);
  pronounResolutionModule.setModules(&constraintDetectionModule);
  // initialize the lookup sets used to look up specific information
  best.insert("best");
  systemInput.insert("system");
  systemInput.insert("slist");
  out.open("resolved.txt");
  if (!out)
  {
    cerr << "Cannot open resolved references output file" << endl;
    exit;
  }
}

DSMainEngine::~DSMainEngine()
{
  out.close();
}
    /**
     * Handle system generated output.
     * - Save the correct data lists.
     * - Update the datalists with presented system output.
     * @param concepts List of concepts representing system generated output.
     */

void DSMainEngine::handleSystemInput(vector<DSConcept> concepts)
{
       int best;

       cout << "handle system input" << endl;

       //cout << "set next sentence" << endl;
```

```
       //sList.nextSentence();

       best = getBest(concepts); // identify the best hypothesis, used by the system to
generate the current system output. The concept containing best information is removed
from the list.
       if (best != 0)
       {
         updateModule.save(best); // save the updated data lists from that hypothesis,
discard the rest
         cout << "best is saved" << endl;
       }

       cout << "starting system list processor" << endl;
       concepts = systemListProcessor.processList(concepts);
       updateModule.update(concepts); // update the data lists with the concepts of the
current system output
       updateModule.finalize();
       cout << "lists are updated" << endl;
}// end handleSystemInput

    /**
     * Handle user input.
     * - Detect deictic input.
     * - Temporarily update the data lists with the information from the deictic input.
     * - Detect and Classify each concept for referential properties.
     * - Detect constraints.
     * - Match referent with reference.
     * - Temporarily update the datalists.
     * - Generate output.
     * @param concepts List of concepts representing user input.
     */

void DSMainEngine::handleUserInput(vector<DSConcept> concepts)
{
       vector<DSConcept> deixisList;

       DSConcept referent;

       cout << "handle user input" << endl;

       sList.nextSentence();

       cout << "set next sentence" << endl;

       deixisList = deixisDetectionModule.extractDeixis(concepts); // extract concepts
derived from deictic input and remove the deictic concepts from the list.
       if (deixisList.size() != 0)
       {
         cout << "deixis present, updated" << endl;

         updateModule.tempUpdate(deixisList); // temporarily update the data lists with
the deictic input
       }
        for (int i=0; i < concepts.size(); i++) // for each concept in the list of
concepts
       {
         if (concepts[i].getConcept()!="" && concepts[i].getSuperConcept()!="")
         {
            DSConcept tmp1 = concepts[i];
            string value = concepts[i].getValue();
            tmp1.setValue(tmp1.getSuperConcept());
            concepts[i].setValue(concepts[i].getConcept());
            detectAndResolve(tmp1, NULL, concepts);
            detectAndResolve(concepts[i], tmp1.getReferent(), concepts);
            concepts[i].setValue(value);
         }
         else
           detectAndResolve(concepts[i], NULL, concepts);
       }// end for
       cout << "finalize temp" << endl;
       updateModule.finalizeTemp();
```

```
       for (int i = 0; i < concepts.size(); i++)
       {
          out << concepts[i].getType() << " (" << concepts[i].getValue() << "): "
              << (concepts[i].getReferent()!=NULL? concepts[i].getReferent()->getType() :
"NULL")
              << " (" << (concepts[i].getReferent()!=NULL? concepts[i].getReferent()-
>getValue() : "NULL")
              << ")" << endl;
          cerr << concepts[i].getType() << " (" << concepts[i].getValue() << "): "
              << (concepts[i].getReferent()!=NULL? concepts[i].getReferent()->getType() :
"NULL")
              << " (" << (concepts[i].getReferent()!=NULL? concepts[i].getReferent()-
>getValue() : "NULL")
              << ")" << endl;
       }
       // DSOutputTranslator.translate(concepts); // translate and send the concepts as
output
}// end handleUserInput

   /**
    * handleConcpets divides input into user and system input. User input is handled
further by handleUserInput, system input by
    * handleSystemInput.
    * @param concepts List of concepts to be handled.
    */

void DSMainEngine::handleConcepts (vector<DSConcept> concepts)
{
       int input; // used to indicate whether the input is from the user (0) or system
(1)
       const int system = 1; // system is valued 1

       input = determineInput(concepts); // determine from the concepts whether the input
is from the user (0) or the system (1)
       if (input == system)
       {
          handleSystemInput(concepts); // if the concepts are derived from system
generated output, then save the correct
       }                        //data lists and update the data lists with the presented
system output.
       else
       {
          handleUserInput(concepts); // if the concepts are derived from user input, then
detect and handle deictic input,
       }                       // ellipsis, and references.
}// end handleConcepts

void DSMainEngine::detectAndResolve(DSConcept &concept, DSConcept *superconcept,
vector<DSConcept> &concepts)
{
  referenceType refType;
  vector<DSConstraint> conList;

  cout << "\nCONCEPT:" << concept.getType() << " (" << concept.getValue() << ") " <<
endl;

  refType = referenceDetectionAndClassificationModule.detectAndClassify(concept); //
determine the referential property
  switch (refType)
  {
   case none: // if no referential property
     cout << "no referential property detected" << endl;

     updateModule.tempUpdate(concept); // simply temporarily update the data lists
     break;
   case pronoun: // if pronoun
     cout << "pronoun detected" << endl;

     conList = constraintDetectionModule.detectConstraints(concept, superconcept,
concepts); // detect the constraints
```

```
     cout << "constraints detected" << endl;
     pronounResolutionModule.resolve(&concept, conList); // resolve the pronoun

     if (concept.getReferent() != NULL)
       cout << "pronoun resolved to: " << concept.getReferent()->getValue() << endl;

     updateModule.tempUpdate(concept); // temporarily update the data lists
     break;
   case demonstrative: // if demonstrative
     cout << "demonstrative detected" << endl;

     conList = constraintDetectionModule.detectConstraints(concept, superconcept,
concepts); // detect the constraints
     demonstrativeResolutionModule.resolve(&concept, conList); // resolve the
demonstrative
     updateModule.tempUpdate(concept); // temporarily update the data lists
     break;
   case definite: // if definite description
     cout << "definite description detected" << endl;

     conList = constraintDetectionModule.detectConstraints(concept, superconcept,
concepts); // detect the constraints
     definiteDescriptionResolutionModule.resolve(&concept, conList); // resolve the
definite description
     updateModule.tempUpdate(concept); // temporarily update the data lists
     break;
   case one: // if one anaphora
     cout << "one anaphora detected" << endl;

     conList = constraintDetectionModule.detectConstraints(concept, superconcept,
concepts); // detect the constraints
     oneAnaphoraResolutionModule.resolve(&concept, conList); // resolve one anaphora
     updateModule.tempUpdate(concept); // temporarily update the data lists
     break;
   case date: // if date
     cout << "date detected" << endl;

     conList = constraintDetectionModule.detectConstraints(concept, superconcept,
concepts); // detect the constraints
     dateResolutionModule.resolve(&concept, conList); // resolve the date
     updateModule.tempUpdate(concept); // temporarily update the data lists
     break;
  }
}


//  protected:
   // none

//  private:

   /**
    * Determines whether the concepts are derived from user input or system output. The
concept which contains this information
    * is removed from the list during this test.
    * @param concepts List of concepts, one concept contains information about the input.
    * @return 0 = user, 1 = system
    */

int DSMainEngine::determineInput(vector<DSConcept> concepts)
{
  cout << "determining input :" << concepts[0].getInputOrigin() << endl;
   if (systemInput.count(concepts[0].getInputOrigin()) !=0)
     return 1; // input is from system.
   return 0; // input is from user.
}

    /**
     * Determines the best hypothesis. This is derived from a concept in the concept
list. This concept is removed from the
     * list during the test.
```

```
     * @param concepts List of concepts, one concept contains information about the
input.
     * @return The best hypothesis.
     */

int DSMainEngine::getBest(vector<DSConcept> &concepts)
{
  vector<DSConcept>::iterator pos = concepts.begin();
  for (int i=0; i < concepts.size(); i++)
  {
     if (best.count(concepts[i].getType()) != 0) // if the concept specifies which
previous hypothesis is used.
     {
        string tmp = concepts[i].getValue(); // get the value.
        int val = 0;
        for (int j=0; j < tmp.size(); j++) // transform it from string to integer.
        {
           val = 10*val + tmp[j]-(int)'0';
        }
        concepts.erase(pos);

        cout << "best is: " << val << endl;

        return val;
        break;
     }
     pos++;
  }
  return 0; //default value.
}
```

Update Module

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: update.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the update class, which is used to update the different data
lists. The data lists
//               currently included are:
//               - History of each concept, which is a history list of the values of each
concept, used to solve definite
//                 descriptions.
//               - Salience List, which is a list of concepts ordered by salience, used
to solve pronominal references.
//
```

```
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSUPDATE_H
#define DSUPDATE_H

#include "concept.h"
#include "typelist.h"
#include "slist.h"

/**
 * The update module is used to update the different data lists. The data lists currently
included are:
 * - History of concepts
 * - History of each concept
 * - Salience list
 */

class DSUpdate
{
  public:

   /**
    * Sets the lists for the module.
    * @param typehislist The type history list for the module.
    * @param slist The s-list for the module.
    */

   void setLists(DSTypeHisList *typehislist, DS_SList *slist);

   /**
    * Saves the temporary data lists from the best hypothesis. The other temporary data
lists are discarded.
    * @param best An integer, which indicates the best hypothesis to be saved.
    */

   void save(int best);

   /**
    * Updates the data lists with the data from the concepts.
    * @param concepts List of concepts, to be added to the data lists.
    */

   void update(vector<DSConcept> concepts);

   /**
    * Temporarily updates the data lists with the data from the concepts.
    * @param concepts List of concepts, to be added temporarily to the data lists.
    */

   void tempUpdate(vector<DSConcept> concepts);

   /**
    * Temporarily updates the data lists with the data from the concept.
    * @param currentConcept Current concept to be added temporarily to the data list.
    */

   void tempUpdate(DSConcept currentConcept);

   /**
    * Finishes the temporary data lists. No further concepts can be added.
    */

   void finalizeTemp();

   /**
    * Finalizes the s-list. No further concepts can be added.
    */

   void finalize();
```

```
  private:

   DSTypeHisList *typeHisList;
   DS_SList *sList;
   vector<DSConcept> tempList;

};

#endif // DSUPDATE_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: update.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the update class, which is used to update the different data
lists. The data lists
//               currently included are:
//               - History of concepts, which is a history of the concepts used each
turn, used to solve ellipsis.
//               - History of each concept, which is a history list of the values of each
concept, used to solve definite
//                 descriptions.
//               - Salience List, which is a list of concepts ordered by salience, used
to solve pronominal references.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "update.h"

/**
 * The update module is used to update the different data lists. The data lists currently
included are:
 * - History of concepts
 * - History of each concept
 * - Salience list
 */


//  public:

   /**
    * Sets the lists for the module.
    * @param hislist The history list for the module. (for ellipsis resolution)
    * @param typehislist The type history list for the module. (for definite description
resolution)
    * @param slist The s-list for the module. (for pronoun resolution)
    */
```

```
void DSUpdate::setLists(DSHistoryList *hislist, DSTypeHisList *typehislist, DS_SList
*slist)
{
  //hisList = hislist;
  typeHisList = typehislist;
  sList = slist;
}

   /**
    * Saves the temporary data lists from the best hypothesis. The other temporary data
lists are discarded.
    * @param best An integer, which indicates the best hypothesis to be saved.
    */

void DSUpdate::save(int best)
{
  //cout << "save hislist" << endl;
  //hisList->save(best);
  cout << "save typehislist" << endl;
  typeHisList->save(best-1);
  cout << "save slist" << endl;
  sList->save(best-1);
}

   /**
    * Updates the data lists with the data from the concepts.
    * @param concepts List of concepts, to be added to the data lists.
    */

void DSUpdate::update(vector<DSConcept> concepts)
{
  cout << "update" << endl;

  // store the concepts in the history list. commented because I'm unsure whether it
should be added to
  // the history list, because system output may possibly not be used for ellipsis.
  // hisList->saveAdd(concepts);
  for (int i=0; i<concepts.size(); i++)
  {
     // store the concepts in the type history lists.
     typeHisList->add(concepts[i]);
     cout << concepts[i].getType() << " (" << concepts[i].getValue() << ") added to
concept list" << endl;
     // store the concepts in the s-list.
     if (concepts[i].getInputOrigin() == "slist")
     {
        cout << concepts[i].getType() << " (" << concepts[i].getValue() << ") added to
slist" << endl;
        sList->add(concepts[i]);
     }
  }
}

   /**
    * Temporarily updates the data lists with the data from the concepts.
    * @param concepts List of concepts, to be added temporarily to the data lists.
    */

void DSUpdate::tempUpdate(vector<DSConcept> concepts)
{
  cout << "temp update" << endl;

  for (int i=0; i<concepts.size(); i++)
  {
     // store the concepts in the history list.
     // check for deictic input if deixis is not to be stored in the history list
     //tempList.push_back(concepts[i]);
     // store the concepts in the type history lists.

     typeHisList->tmpAdd(concepts[i]);
     // store the concepts in the s-list.
```

```
     sList->tempAdd(concepts[i]);
     cout << "added " << concepts[i].getValue() << " to s-list, size is now: " << sList-
>tempSize() << endl;
  }
}

   /**
    * Temporarily updates the data lists with the data from the concept.
    * @param currentConcept Current concept to be added temporarily to the data list.
    */

void DSUpdate::tempUpdate(DSConcept currentConcept)
{
  // store the concepts in the history list.
  // check for deictic input if deixis is not to be stored in the history list
  //tempList.push_back(currentConcept);
  // store the concepts in the type history lists.

  typeHisList->tmpAdd(currentConcept);
  // store the concepts in the s-list.
  sList->tempAdd(currentConcept);
  cout << "added " << currentConcept.getValue() << " to s-list, size is now: " << sList-
>tempSize() << endl;
}

   /**
    * Finishes the temporary data lists. No further concepts can be added.
    */

void DSUpdate::finalizeTemp()
{
  //hisList->tempAdd(tempList);
  cout << "hislist finalized" << endl;
  typeHisList->finalize();
  cout << "type history list finalized" << endl;
  sList->finalizeTemp();
  cout << "slist temp finalized" << endl;
}
   /**
    * Finalizes the s-list. No further concepts can be added.
    */

void DSUpdate::finalize()
{
     sList->finalize();
     cout << "slist finalized" << endl;
}
//  protected:
   // none

//  private:
   // none
```

Salience List

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//              Copyright (C) 2001 Philips GmbH Dialog Systems
//
//              All rights reserved
//
```

```
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: slist.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the salience list, used to order the concepts by discourse
old/new
//               and hearer old/new.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DS_SLIST_H
#define DS_SLIST_H

#include "listent.h"
#include "concept.h"
#include <deque>
#include <vector>
#include <set>
#include <list>

/**
 * This is the salience list, used to order the concepts by discourse old/new and hearer
old/new.
 */

class DS_SList
{
public:

   /**
    * Constructor.
    */

   DS_SList();

  /**
   * Returns the size of the s-list.
   * @return The size of the s-list.
   */

   int size();

     /**
   * Returns the size of the temporarily saved list.
   * @return The size of the temporarily saved list.
   */

   int tempSize();

  /**
   * Saves temporary s-list i. This overwrites the previous s-list.
   * The temporary list is cleared.
   * @param i The number of the list to be saved.
   */

  void save(int i);

   /**
    * Adds a concept to the temporary s-list.
    * @param concept The concept to be added to the temporary s-list.
```

```
     */

    void tempAdd(DSConcept concept);

    /**
     * Adds a concept to the s-list.
     * @param concept The concept to be added to the s-list.
     */

    void add(DSConcept concept);

    /**
     * Finishes the s-list by removing the concepts not used this turn.
     */

    void finalize();

    /**
     * Finishes the temporary s-list by removing the concepts not used this turn and
adding it to the temporary s-lists.
     */

    void finalizeTemp();

    /**
     * Returns the concept at position i in the s-list.
     * @param i The index of the concept to be returned.
     * @return The concept at position i in the s-list.
     */

    DSConcept get(int i);

    /**
     * Returns the concept at position i in the temporary s-list.
     * @param i The index of the concept to be returned.
     * @return The concept at position i in the temporary s-list.
     */

    DSConcept getTemp(int i);

    /**
     * Returns the sentence number of the concept at position i in the s-list.
     * @param i The index of the sentence number of the concept to be returned.
     * @return The sentence number of the concept at position i in the s-list.
     */

    int getSentenceNr(int i);


    /**
     * Returns the sentence number of the concept at position i in the temporary s-list.
     * @param i The index of the sentence number of the concept to be returned.
     * @return The sentence number of the concept at position i in the temporary s-list.
     */

    int getTempSentenceNr(int i);


    /**
     * Increases the sentence number.
     */

    void nextSentence();

       /**
     * Returns the sentence number.
     * @return Rhe sentence number.
     */

    int getSentence();
```

```
protected:
  //none

private:
  int sentenceNr; //number of the current sentence, used for intrasentential resolution.
  list<DSListEnt> sList; //the salience-list.
  list<DSListEnt> temp; //the temporary s-list
  vector<list<DSListEnt> > tempList; //list of temporary salience-lists.
  vector<DSListEnt> used; //list of used entities this turn.

  set<string> deixisTypes; //types which indicates deixis.
  set<string> namesAndTitles; //set of names and titles, used to determine 'unused'
concepts;
  set<string> inferrableInd; //indicators for definite descriptions that might have been
resolved. If not resolved than inferrable.
  vector<string> infConInd; // indicators for inferrable contained.
  set<string> bnA_Ind; //set of indicators for bnA.
  set<string> bnInd; //set of brand new indicators.
  bool on; //indicates wether a sentence increase finalizes an utterance.
  bool dialog; //indicates whether the interaction is a dialog or a monologue.
  vector<string> posMod; //set of possesive modifiers like 's.'
  entTag slistTag; // predefined tag for concepts from system forced on the slist.

  /**
   * Tags a concept with old-new information.
   * @param concept The concept to be tagged.
   * @param list The list according to which the concept must be tagged.
   * @return A list entity, containing the concept and the tag.
   */

   DSListEnt tag(DSConcept concept, list<DSListEnt> &lst);

  /**
   * inserts an entity into an s-list.
   * @param entity The entity to be inserted.
   * @param list The s-list in which the entity must be inserted.
   */

   void insertEnt(DSListEnt entity, list<DSListEnt> &lst);

  /**
   * Sends a representation of the list to std out.
   * @param lst The list to be presented.
   */

   void printList(list<DSListEnt> lst);
};

#endif

////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: listent.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
```

```
//  Created on: January 17, 2001
//
//  Description: This is an entity of the salience list, it consists of a concept and a
tag
//               which indicates whether it is new or old.
//
////////////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSLISTENT_H
#define DSLISTENT_H

#include "concept.h"


enum entTag {untagged, deixis, evoked, unused, inferrable, infcont, bna, bn};

/**
 * This is an entity of the salience list, it consists of a concept and a tag which
indactes whether it is new or old.
 */

class DSListEnt
{
public:

   /**
    * Constructor.
    */

   DSListEnt();

   /**
    * Constructor.
    * @param concept The concept.
    * @param tag Tag information about how new the concept is to the 'hearer'.
    */

   DSListEnt(DSConcept concept, entTag tag);

   /**
    * Sets the concept of the list entity.
    * @param concept The new concept.
    */

   /**
    * Constructor.
    * @param concept The concept.
    * @param tag Tag information about how new the concept is to the 'hearer'.
    * @param sentence The number of the sentence, needed for intrasentential constraints.
    */

   DSListEnt(DSConcept concept, entTag tag, int sentence);

   void setConcept(DSConcept concept);

   /**
    * Sets the tag of the list entity.
    * @param tag The new tag.
    */

   void setTag(entTag tag);

   /**
    * Sets the sentence number, needed for intrasentential constraints.
    * @param nr The sentence number.
    */

   void setSentence(int nr);

   /**
```

```
    * Returns the concept.
    * @return The concept.
    */

   DSConcept getConcept();

   /**
    * Returns the tag.
    * @return The tag.
    */

   entTag getTag();

   /**
    * Gets the sentence number, needed for intrasentential constraints.
    * @return The sentence number.
    */

   int getSentence();

   /**
    * == operator for DSListEnt.
    */

   bool operator==(DSListEnt);

protected:
  //none

private:
 DSConcept concept;
 entTag tag;
 int sentence;
};

#endif
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: slist.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the salience list, used to order the concepts by discourse
old/new
//               and hearer old/new.
//
////////////////////////////////////////////////////////////////////////////////
////////
```

```
#include "slist.h"
#include <fstream>

//public:

/**
 * Constructor.
 */

DS_SList::DS_SList()
{
  sentenceNr = 0;

  string tmp;
  cout << "loading tag indicators" << endl;
  ifstream in;
  in.open("names.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open names and titles data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        namesAndTitles.insert(tmp);
     }
  }
  in.close();
  in.open("inferrable.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open inferrable data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        inferrableInd.insert(tmp);
     }
  }
  in.close();
  in.open("infCon.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open inferrable contained data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        infConInd.push_back(tmp);
     }
  }
  in.close();
  in.open("bn.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open brand new data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
```

```
   {
      bnInd.insert(tmp);
   }
}
in.close();
in.open("bna.txt", ios::in);
if (!in)
{
   cerr << "Cannot open brand new anchored data file" << endl;
   exit;
}
while (!in.eof())
{
   getline(in, tmp);
  if (tmp.find("#") == string::npos && tmp!="")
   {
      bnA_Ind.insert(tmp);
   }
}
in.close();
in.open("deixisTypes.txt", ios::in);
if (!in)
{
   cerr << "Cannot open deixis types data file" << endl;
   exit;
}
while (!in.eof())
{
   getline(in, tmp);
   if (tmp.find("#") == string::npos && tmp!="")
   {
      deixisTypes.insert(tmp);
   }
}
in.close();
in.open("slist.txt", ios::in);
if (!in)
{
   cerr << "Cannot open slist parameter file" << endl;
   exit;
}
on=false;
dialog=false;
while (!in.eof())
{
   getline(in, tmp);
   if (tmp.find("#") == string::npos && tmp!="")
   {
      if (tmp == "on")
        on=true;
      if (tmp == "dialog")
        dialog=true;
      else if (tmp == "untagged")
        slistTag=untagged;
      else if (tmp == "deixis")
        slistTag=deixis;
      else if (tmp == "evoked")
        slistTag=evoked;
      else if (tmp == "unused")
        slistTag=unused;
      else if (tmp == "inferrable")
        slistTag=inferrable;
      else if (tmp == "infcont")
        slistTag=infcont;
      else if (tmp == "bna")
        slistTag=bna;
      else if (tmp == "bn")
        slistTag=bn;
   }
}
in.close();
```

```
  in.open("posmod.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open possesive modifier data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        if (tmp != "")
        {
           posMod.push_back(tmp);
        }
     }
  }
  in.close();
}

  /**
   * Returns the size of the temporarily s-list.
   * @return The size of the temporarily s-list.
   */

int DS_SList::tempSize()
{
  return temp.size();
}
  /**
   * Returns the size of the s-list.
   * @return The size of the s-list.
   */

int DS_SList::size()
{
  return sList.size();
}

  /**
   * Saves temporary s-list i. This overwrites the previous s-list.
   * The temporary list is cleared.
   * @param i The number of the list to be saved.
   */

void DS_SList::save(int i)
{
  sList = tempList[i];
  tempList.clear();
  temp = sList;
  printList(temp);
}

   /**
    * Adds a concept to the temporary s-list.
    * @param concept The concept to be added to the temporary s-list.
    */

void DS_SList::tempAdd(DSConcept concept)
{
  DSListEnt tmpEnt;
  cout << "now tagging" << endl;
  tmpEnt = tag(concept, temp); // tag the concept according to the temporary s-list.
  tmpEnt.setSentence(sentenceNr); // sets the sentence nr of the concept.

  string tmp;
  switch (tmpEnt.getTag())
  {
   case untagged: tmp = " untagged"; break;
   case deixis: tmp = " deixis"; break;
   case evoked: tmp = " evoked"; break;
```

```
   case unused: tmp = " unused"; break;
   case inferrable: tmp = " inferrable"; break;
   case infcont: tmp = " infcont"; break;
   case bna: tmp = " bna"; break;
   case bn: tmp = " bn"; break;
  }
  cout << concept.getValue() << tmp << endl;

  insertEnt(tmpEnt, temp); // insert the tagged concept into the temporary s-list.
  used.push_back(tmpEnt); // add the tagged concept to the set of used concepts.
  cout << "used size is now: " << used.size() << endl;
  printList(temp);
}

   /**
    * Adds a concept to the s-list.
    * @param concept The concept to be added to the s-list.
    */

void DS_SList::add(DSConcept concept)
{
  DSListEnt tmpEnt;
  tmpEnt = tag(concept, sList); // tag the concept accordig to the s-list
  tmpEnt.setSentence(sentenceNr); // sets the sentence nr of the concept.

  string tmp;
  switch (tmpEnt.getTag())
  {
   case untagged: tmp = " untagged"; break;
   case deixis: tmp = " deixis"; break;
   case evoked: tmp = " evoked"; break;
   case unused: tmp = " unused"; break;
   case inferrable: tmp = " inferrable"; break;
   case infcont: tmp = " infcont"; break;
   case bna: tmp = " bna"; break;
   case bn: tmp = " bn"; break;
  }
  //cout << concept.getValue() << tmp << endl;

  if (tmpEnt.getTag()!=untagged)
  {
     insertEnt(tmpEnt, sList); // insert the tagged concept into the s-list
     used.push_back(tmpEnt); // add the tagged concept to the set of used concepts.
  }
  cout << "used size is now: " << used.size() << endl;
  printList(temp);
}

   /**
    * Finishes the s-list by removing the concepts not used this turn.
    */

void DS_SList::finalize()
{
  int size, usedSize;
  list<DSListEnt> tmp;

  if (dialog)
  {
  usedSize = used.size();
  for (list<DSListEnt>::iterator i=sList.begin(); i!=sList.end() ; i++) // for every
element in the s-list
    for (int j=0; j<usedSize; j++)
    {
       if (used[j] == *i) // check whether it is used
         tmp.push_back(*i); // create a list of only used entities
    }
  sList = tmp; // the s-list consists only of used entities.
  used.clear(); // clear the information of all used entities.
  }
  temp = sList;
```

```
  printList(temp);
}

   /**
    * Finishes the temporary s-list by removing the concepts not used this turn and
adding it to the temporary s-lists.
    */

void DS_SList::finalizeTemp()
{
  int size, usedSize;
  list<DSListEnt> tmp;

  usedSize = used.size();
  cout << "usedSize = " << usedSize << endl;
  for (list<DSListEnt>::iterator i=temp.begin(); i!=temp.end(); i++) // for every element
in the temporary s-list
    for (int j=0; j<usedSize; j++)
    {
       if (used[j] == *i) // check whether it is used
         tmp.push_back(*i); // create a list of only used entities
    }
  cout << "tempList size = " << tempList.size() << endl;
  tempList.push_back(tmp); // save the temporary s-list consisting only of used entities.
  used.clear(); // clear the information of all used entities.
  cout << "tempList size after update = " << tempList.size() << endl;
  cout << "size of last entry in tempList = " << tempList[tempList.size()-1].size() <<
endl;
  temp = sList; // clear the temporary s-list.
  printList(temp);
}

  /**
    * Returns the concept at position i in the s-list.
    * @param i The index of the concept to be returned.
    * @return The concept at position i in the s-list.
    */

DSConcept DS_SList::get(int i)
{
  list<DSListEnt>::iterator pos;

  pos = sList.begin();
  int j=0;
  while(j < i)
  {
     pos++;
     j++;
  }

  return pos->getConcept();
}

   /**
    * Returns the concept at position i in the temporary s-list.
    * @param i The index of the concept to be returned.
    * @return The concept at position i in the temporary s-list.
    */

DSConcept DS_SList::getTemp(int i)
{
  list<DSListEnt>::iterator pos;

  pos = temp.begin();
  int j=0;
  while(j < i)
  {
     pos++;
     j++;
  }
```

```
  return pos->getConcept();
}


   /**
    * Returns the sentence number of the concept at position i in the s-list.
    * @param i The index of the sentence number of the concept to be returned.
    * @return The sentence number of the concept at position i in the s-list.
    */

int DS_SList::getSentenceNr(int i)
{
  list<DSListEnt>::iterator pos;

  pos = sList.begin();
  int j=0;
  while(j < i)
  {
     pos++;
     j++;
  }

  return pos->getSentence();
}

   /**
    * Returns the sentence number of the concept at position i in the temporary s-list.
    * @param i The index of the sentence number of the concept to be returned.
    * @return The sentence number of the concept at position i in the temporary s-list.
    */

int DS_SList::getTempSentenceNr(int i)
{
  list<DSListEnt>::iterator pos;

  pos = temp.begin();
  int j=0;
  while(j < i)
  {
     pos++;
     j++;
  }

  return pos->getSentence();
}
   /**
    * Increases the sentence number.
    */

void DS_SList::nextSentence()
{
  // first check whether there are entities with the current sentence number added to the
slist this sentence not equal to deixis.
  // if not, only update sentence number
  cout << "increasing sentence number" << endl;
  if (used.size() != 0 && on)
  {
     cout << "removing not used entities from list, new size will be: " << used.size() <<
endl;
     for (int h=0; h<used.size(); h++)
     {
        if (used[h].getTag() != deixis)
        {
           int usedSize;
           list<DSListEnt> tmp;

           cout << "removing not used entities from list, new size will be: " <<
used.size() << endl;
           usedSize = used.size();
           for (list<DSListEnt>::iterator i=temp.begin(); i!=temp.end() ; i++) // for
every element in the temporary s-list
```

```
            for (int j=0; j<usedSize; j++)
            {
               if (used[j] == *i) // check whether it is used
                 tmp.push_back(*i); // create a list of only used entities
            }
          temp = tmp; // the temporary s-list consists only of used entities.
          used.clear(); // clear the information of all used entities.
          break;
        }
     }
  }
  sentenceNr++;
  printList(temp);
}

   /**
    * Returns the sentence number.
    * @return Rhe sentence number.
    */

int DS_SList::getSentence()
{
  return sentenceNr;
}

//protected:
  //none

//private:

  /**
   * Tags a concept with old-med-new information. If the concept is already in the list,
it will become evoked
   * and removed from the list (it is expected that the concept will be inserted later
with the new tag) unless it is already new.
   * @param concept The concept to be tagged.
   * @param list The list according to which the concept must be tagged.
   * @return A list entity, containing the concept and the tag.
   */

DSListEnt DS_SList::tag(DSConcept concept, list<DSListEnt> &lst)
{
  // if input origin is from system and it is forced on the slist, then tag it according
to user set Tag.
  if (concept.getInputOrigin()=="slist")
    return DSListEnt(concept, slistTag);

  cout << "can it be tagged as deixis?" << endl;
  // tag deixis
  if (deixisTypes.count(concept.getInputOrigin()) != 0) //check whether the input origin
of the concept is a deixis type
    return DSListEnt(concept, deixis); // tag it as deixis

  // a concept is evoked if it is co-referring with a concept already in the list.
Proniminal and nominal anaphora,
  // previous mentioned proper names, relative pronouns, appositives.

  cout << "does it has a referent?" << endl;
  list<DSListEnt>::iterator i;
  if(concept.getReferent() != NULL) // if the concept has a referent, evoke the referent.
  {
     cout << "referent = " << concept.getReferent()->getValue() << endl;
     if (lst.size() > 0)
     {
        i=lst.begin();
        while (i != lst.end())
        {
           if (i->getConcept() == (*concept.getReferent()))
           {
              cout << "referent found in list" << endl;
```

```
             if ((i->getTag() != evoked) && (i->getTag() != unused) && (i->getTag() !=
deixis)) // if not already old, remove it to be replaced
             {
                string tmp;
                switch (i->getTag())
                {
                 case untagged: tmp = "untagged"; break;
                 case deixis: tmp = "deixis"; break;
                 case evoked: tmp = "evoked"; break;
                 case unused: tmp = "unused"; break;
                 case inferrable: tmp = "inferrable"; break;
                 case infcont: tmp = "infcont"; break;
                 case bna: tmp = "bna"; break;
                 case bn: tmp = "bn"; break;
                }
                cout << "the referent isn't old, it is tagged as: " << tmp << endl;
                lst.erase(i);
                DSConcept referent = *concept.getReferent();
                referent.setTimestamp(concept.getTimestamp()); // update the timestamp of
the referent.
                return DSListEnt(referent, evoked, sentenceNr);
             }
             else // else tag it as evoked if unused.
             {
                if (i->getTag() == unused)
                  i->setTag(evoked);
                DSConcept tmp = i->getConcept();
                tmp.setTimestamp(concept.getTimestamp()); // update the timestamp of the
concept.
                i->setConcept(tmp);
                 cout << "concept has been evoked, new timestamp: " <<
concept.getTimestamp() << endl;
                i->setSentence(sentenceNr); //update the sentence number of the entity.
                used.push_back(*i);
                return DSListEnt(concept, untagged, sentenceNr);
             }
          }// endif
          i++;
       }// end while
       cout << "referent not found in the list" << endl;
       DSConcept referent = *concept.getReferent();
       referent.setTimestamp(concept.getTimestamp()); // update the timestamp of the
referent.
       return DSListEnt(referent, evoked, sentenceNr);
    } // end if list not empty
    else
    {
       cout << "referent not found in the list, because list was empty" << endl;
       DSConcept referent = *concept.getReferent();
       referent.setTimestamp(concept.getTimestamp()); // update the timestamp of the
referent.
       return DSListEnt(referent, evoked, sentenceNr);
    }
  } // end if has referent

  cout << "is it an inferrable?" << endl;

  // brand-new proper names are usually accompanied by a relative clause or an appositive
which relates them to the
  // hearer's knowledge.

  string conceptVal = concept.getValue();

  //check for 's.
  //commented out because of ambiguity with abbreviation of "is".
  /*for (int i=0; i < posMod.size(); i++)
  {
     if (conceptVal.find(posMod[i]) != string::npos)
       return DSListEnt(concept, inferrable, sentenceNr);
       }*/
  for (int i=0; i< infConInd.size(); i++)
```

```
  {
     if ( conceptVal.find(infConInd[i]) != string::npos)
       return DSListEnt(concept, infcont, sentenceNr);
  }

  cout << "check for indicators" << endl;

  //check whether indicators exist.
  entTag conceptTag = untagged; // initialize conceptTag
  int nextpos;

  while(conceptVal.find(" ") != string::npos)
  {
     nextpos = conceptVal.find(" ");
     //cout << conceptVal.substr(0,nextpos) << endl;
     if (inferrableInd.count(conceptVal.substr(0,nextpos))!=0)
       return DSListEnt(concept, ((conceptTag == bn) ? bna : inferrable), sentenceNr);
     if (bnInd.count(conceptVal.substr(0,nextpos))!=0)
       conceptTag = bn;
     if (bnA_Ind.count(conceptVal.substr(0,nextpos))!=0)
       return DSListEnt(concept, bna, sentenceNr); //perhaps check whether it is bn
first, though no problems are expected that can be solved with this.
     // current word is not an indicator, so try next one.
     conceptVal = conceptVal.substr(nextpos+1);
  }
  // check whether last word is an indicator.
  nextpos=conceptVal.size();
  if (bnA_Ind.count(conceptVal.substr(0,nextpos))!=0)
    return DSListEnt(concept, bna, sentenceNr); //perhaps check whether it is bn first,
though no problems are expected that can be solved with this.if
(oneInd.count(concept.substr(0,nextpos-1))!=0)
  if (conceptTag != untagged)
    return DSListEnt(concept, conceptTag, sentenceNr);
  if (lst.size() > 0)
  {
     // check whether the concept is already in the list.

     cout << "is the concept already in the list?" << endl;

     i=lst.begin();
     while (i != lst.end())
     {
        if ((i->getConcept() == concept)&&(i-
>getConcept().getTimestamp()!=concept.getTimestamp())) // if so, then remove the concept
from the list and tag as evoked
        {
           if ((i->getTag() != evoked) && (i->getTag() != unused) && (i->getTag() !=
deixis)) // not old
           {
              lst.erase(i); // remove the concept from the list
              return DSListEnt(concept,evoked, sentenceNr); // so it can be inserted at
the right position
           }
           else // old
           {
              if (i->getTag() == unused)
                i->setTag(evoked); // update the tag
              i->setSentence(sentenceNr); // update the sentence Nr.
              return DSListEnt(concept, untagged);
           }
        }
        i++;
     }

     cout << "is the concept value already in the list?" << endl;

     // check whether the concept value is already in the list.

     i=lst.begin();
     while (i != lst.end())
     {
```

```
       if ((i->getConcept().getValue() == concept.getValue())&&(i-
>getConcept().getTimestamp()!=concept.getTimestamp())) // if so, then remove the concept
from the list and tag as evoked
       {
          if ((i->getTag() != evoked) && (i->getTag() != unused) && (i->getTag() !=
deixis))
          {
             lst.erase(i);
             return DSListEnt(concept,evoked, sentenceNr);
          }
          else
          {
             if (i->getTag() == unused)
               i->setTag(evoked);
             i->setSentence(sentenceNr); // update the sentence Nr.
             return DSListEnt(concept, untagged);
          }
       }
       i++;
     }

     cout << "is a substring already in the list, or is it a substring of a value already
in the list?" << endl;

     // check whether the concept value is a substring already in the list.

     i=lst.begin();
     while (i != lst.end())
     {
       if ((i->getConcept().getValue().find(concept.getValue()) != string::npos)&&(i-
>getConcept().getTimestamp()!=concept.getTimestamp())) // if so, then remove the concept
from the list and tag as evoked
       {
          if ((i->getTag() != evoked) && (i->getTag() != unused) && (i->getTag() !=
deixis))
          {
             lst.erase(i);
             return DSListEnt(concept,evoked, sentenceNr);
          }
          else
          {
             if (i->getTag() == unused)
               i->setTag(evoked);
             i->setSentence(sentenceNr); // update the sentence Nr.
             return DSListEnt(concept, untagged);
          }
       } // or vice versa
       else if ((concept.getValue().find(i->getConcept().getValue()) !=
string::npos)&&(i->getConcept().getTimestamp()!=concept.getTimestamp())) // hopefully
doesn't lead to big/strange errors
       {                                              // possible additional constraint:
type must be the same.
          if ((i->getTag() != evoked) && (i->getTag() != unused) && (i->getTag() !=
deixis))
          {
             lst.erase(i);
             return DSListEnt(concept,evoked, sentenceNr);
          }
          else
          {
             if (i->getTag() == unused)
               i->setTag(evoked);
             i->setSentence(sentenceNr); // update the sentence Nr.
             return DSListEnt(concept, untagged);
          }
       }
       i++;
     } // end while
  } // end if (lst.size > 0)

  // a concept is unused if it is a proper name or a title.
```

```
  cout << "is it a name or a title?" << endl;

  if (namesAndTitles.count(concept.getValue()) != 0 ) // check whether the concept value
is a name or title
  {
     return DSListEnt(concept, unused, sentenceNr);
  }

  return DSListEnt(concept,conceptTag, sentenceNr);
}

  /**
   * inserts an entity into an s-list.
   * @param entity The entity to be inserted.
   * @param list The s-list in which the entity must be inserted.
   */

void DS_SList::insertEnt(DSListEnt entity, list<DSListEnt> &lst)
{
  if (entity.getTag()==bn) // new
  {
     cout << entity.getConcept().getValue() << " put at the end of the list, before bn's
from previous sentence" << endl;

     if (lst.size() == 0)
     {
       lst.push_back(entity); // place it at the end of the list
       cout << entity.getConcept().getValue() << " put at end of the list"  << endl;
     }
     else
     {
        list<DSListEnt>::iterator pos = lst.end();
        pos--;
        for (int i=lst.size()-1; i>=0; i--)
        {
           if (!((pos->getTag()==bn)&&(pos->getSentence() != sentenceNr)))// put at the
end of the list, before bn's from previous sentence
           {
              pos++;
              lst.insert(pos, entity);

              cout << entity.getConcept().getValue() << " put at position" << i+1 << endl;
              break;
           }
           if (i==0) //
           {
              lst.push_front(entity);
              cout << entity.getConcept().getValue() << " put in front of the list" <<
endl;
           }
           pos--;
        }
     }
  }
  else if ((entity.getTag()==bna) || (entity.getTag()==infcont) ||
(entity.getTag()==inferrable)) //med
  {
     if (lst.size() == 0)
       lst.push_back(entity); // place it at the end of the list
     else
     {
        list<DSListEnt>::iterator pos = lst.end();
        pos--;
        for (int i=lst.size()-1; i>=0; i--) //place it after the med of the same sentence
or any evoked entities
        {
           if ((((pos->getTag()==bna) || (pos->getTag()==infcont) || (pos-
>getTag()==inferrable))
                &&(pos->getSentence()==sentenceNr))
               || (pos->getTag()==evoked) || (pos->getTag()==deixis))
```

```
          {
             pos++;
             lst.insert(pos, entity);

             cout << entity.getConcept().getValue() << " put at position" << i+1 << endl;
             break;
          }
          if (i==0) //
          {
             lst.push_front(entity);
             cout << entity.getConcept().getValue() << " put in front of the list" <<
endl;
          }
          pos--;
       }
    }
  }
  else if (entity.getTag()==untagged) // already set as evoked
  { // do nothing
  }
  else  // old
  {
     cout << "list size = " << lst.size() << endl;
     if (lst.size() == 0)
     {
        lst.push_back(entity);
        cout << entity.getConcept().getValue() << " put at the end of the list" << endl;
     }
     else
     {
       list<DSListEnt>::iterator pos = lst.begin();
       for (int i=0; i <= lst.size(); i++) // place it after the old entities
       {
/*        string tmp;
          switch (pos->getTag())
          {
           case untagged: tmp = "untagged"; break;
           case deixis: tmp = "deixis"; break;
           case evoked: tmp = "evoked"; break;
           case unused: tmp = "unused"; break;
           case inferrable: tmp = "inferrable"; break;
           case infcont: tmp = "infcont"; break;
           case bna: tmp = "bna"; break;
           case bn: tmp = "bn"; break;
          }
          cout << "currently at position " << i << " tag is: " << tmp << endl; */
          if (i == lst.size())
          {
             lst.push_back(entity);
             cout << entity.getConcept().getValue() << " put at the end of the list" <<
endl;
             break;
          }
          if (!(((pos->getTag()==evoked)||(pos->getTag()==deixis)||(pos-
>getTag()==unused))&&(pos->getSentence()==sentenceNr)))
          {
             lst.insert(pos,entity);
             cout << entity.getConcept().getValue() << " put at position" << i << endl;
             break;
          }
          if ((pos->getTag()==bna) || (pos->getTag()==infcont) || (pos-
>getTag()==inferrable)
              || (pos->getTag()==bn))
          {
             lst.insert(pos, entity);
             cout << entity.getConcept().getValue() << " put at position" << i << endl;
             break;
          }
          pos++;
       }
     }
```

```
  }
  lst.unique(); // remove all duplicates
  printList(lst);
}

/**
 * Sends a representation of the list to std out.
 * @param lst The list to be presented.
 */

void DS_SList::printList(list<DSListEnt> lst)
{
  cout << "S-list (" << lst.size() << "): ";
  list<DSListEnt>::iterator pos = lst.begin();
  for (int i = 0; i < lst.size(); i++)
  {
     string tmp;
     switch (pos->getTag())
     {
      case untagged: tmp = "untagged"; break;
      case deixis: tmp = "deixis"; break;
      case evoked: tmp = "evoked"; break;
      case unused: tmp = "unused"; break;
      case inferrable: tmp = "inferrable"; break;
      case infcont: tmp = "infcont"; break;
      case bna: tmp = "bna"; break;
      case bn: tmp = "bn"; break;
     }
     cout << pos->getConcept().getValue() << " (" << tmp << "), ";
     pos++;
  }
  cout << endl;
}

////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: listent.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is an entity of the salience list, it consists of a concept and a
tag
//               which indicates whether it is new or old.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "listent.h"


//public:

   /**
    * Constructor.
    */
```

```
DSListEnt::DSListEnt()
{
  //none
}
   /**
    * Constructor.
    * @param concept The concept.
    * @param tag Tag information about how new the concept is to the 'hearer'.
    */

DSListEnt::DSListEnt(DSConcept concept, entTag tag)
{
  this->concept = concept;
  this->tag = tag;
}

   /**
    * Constructor.
    * @param concept The concept.
    * @param tag Tag information about how new the concept is to the 'hearer'.
    * @param sentence The number of the sentence, needed for intrasentential constraints.
    */

DSListEnt::DSListEnt(DSConcept concept, entTag tag, int sentence)
{
  this->concept = concept;
  this->tag = tag;
  this->sentence = sentence;
}

   /**
    * Sets the concept of the list entity.
    * @param concept The new concept.
    */

void DSListEnt::setConcept(DSConcept concept)
{
  this->concept = concept;
}

   /**
    * Sets the tag of the list entity.
    * @param tag The new tag.
    */

void DSListEnt::setTag(entTag tag)
{
  this->tag = tag;
}

   /**
    * Sets the sentence number, needed for intrasentential constraints.
    * @param nr The sentence number.
    */

void DSListEnt::setSentence(int nr)
{
  sentence = nr;
}

   /**
    * Returns the concept.
    * @return The concept.
    */

DSConcept DSListEnt::getConcept()
{
  return concept;
}
```

```
   /**
    * Returns the tag.
    * @return The tag.
    */

entTag DSListEnt::getTag()
{
  return tag;
}

   /**
    * Gets the sentence number, needed for intrasentential constraints.
    * @return The sentence number.
    */

int DSListEnt::getSentence()
{
  return sentence;
}

/**
    * == operator for DSListEnt.
    */

bool DSListEnt::operator==(DSListEnt a)
{
  return (a.tag == tag)&&(a.concept == concept);
}

//protected:
  //none

//private:
   //none
```

History List

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: typelist.h
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: Data structure which contains type histories.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSTYPEHISLIST_H
```

```
#define DSTYPEHISLIST_H

#include "typehis.h"

#include <map>

/**
 * Data structure which contains type histories..
 */

class DSTypeHisList
{
  public:

   /**
    * Constructor.
    */

   DSTypeHisList();

   /**
    * Adds a concept to the oppropriate temporary list.
    * If the concept isn't used before, a new history list is created.
    * @param concept Concept to be added.
    */

   void tmpAdd(DSConcept concept);

   /**
    * Adds a concept to the oppropriate list. If the concept isn't used before, a new
history list is created.
    * @param concept Concept to be added.
    */

   void add(DSConcept concept);

   /**
    * Saves the temporary history lists of hypothesis i of each concept.
    * @param i The index of the temporary history lists to be saved.
    */

   void save(int i);

   /**
    * Finalizes the history lists. Each temporary history list is added to the list of
temporary history lists.
    * The temporary history list is reseted to the saved list.
    */

   void finalize();

   /**
    * Returns the size of the temporary history list of a certain type.
    * @param type The type of the history list, of which the size is requested.
    * @return The size of the temporary history list of the specified type.
    */

   int tempSize(string type);

   /**
    * Returns the size of the history list of a certain type.
    * @param type The type of the history list, of which the size is requested.
    * @return The size of the history list of the specified type.
    */

   int size(string type);

   /**
    * Returns the concept at a certain position from the temporary history list of a
certain type.
    * @param type The type of the concept.
```

```
    * @param i The index of the concept.
    * @return The concept at position i of the specified type.
    */

   DSConcept getTemp(string type, int i);


   /**
    * Returns the concept at a certain position from the history list of a certain type.
    * @param type The type of the concept.
    * @param i The index of the concept.
    * @return The concept at position i of the specified type.
    */

   DSConcept get(string type, int i);

   /**
    * Returns the list of keys.
    * @return The list of keys.
    */

   vector<string> getKeyList();


  protected:
   // none

  private:
   map<string, DSTypeHistory> typehislist; //the type history list.
   vector<string> keylist; // list of existing keys.
   int hypothesis; // number of the hypothesis
};



#endif // DSTYPEHISLIST_H

//////////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//////////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: typehis.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the history list of a concept type, used to determine the
referent of a definite description
//
//////////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSTYPEHISTORY_H
#define DSTYPEHISTORY_H

#include "concept.h"
#include <deque>

/**
```

```
 * This is the history list of a concept type, used to determine the referent of a
definite description
 */

class DSTypeHistory
{

public:

   /**
    * Constructor.
    */

   DSTypeHistory();

   /**
    * Constructor.
    */

   DSTypeHistory(string type);

  /**
   * Returns the size of the type history.
   * @return The size of the type history.
   */

   int size();

     /**
   * Returns the size of the temporarily type history.
   * @return The size of the temporarily type history.
   */

   int tempSize();

  /**
   * Saves temporary type history i.
   * The temporary list is cleared.
   * @param i The number of the list to be saved.
   */

  void save(int i);

   /**
    * Adds a concept to the temporary history. If the number of concepts exceeds maxSize
then the oldest is discarded.
    * maxSize is defined in "typehis.cfg".
    * @param concept The concept to be added to the temporary list.
    */

   void tempAdd(DSConcept concept);

   /**
    * Adds a concept list to the saved history. If the number of concepts exceeds maxSize
then the oldest is discarde.
    * maxSize is defined in "typehis.cfg".
    * @param clist The concept to be added to the saved list.
    */

   void add(DSConcept concept);

   /**
    * Returns the saved concept i.
    * @param i The index of the saved concept.
    * @return The last saved concept i.
    */

   DSConcept get(int i);

   /**
    * Returns the temporarily saved concept i.
```

```
    * @param i The index of the temporarily saved concept.
    * @return The temporarily saved concept i.
    */

   DSConcept getTemp(int i);

   /**
    * Finalizes the temporary type history. This pushes the temporary type history on the
list and resets the history.
    */

   void finalize(int hypothesis);

   /**
    * Sets the type of the type history.
    * @param type The type of the type history.
    */

   void setType(string type);

   /**
    * Returns the type of the type history.
    * @return The type of the type history.
    */

   string getType();

protected:
  //none

private:
  deque<DSConcept> saved; //saved type history.
  deque<DSConcept> temp; //temporary type history
  vector<deque<DSConcept> > tempList; //list of all temporary type history.
  int maxSize; //the maxSize of the saved conceptlists.
  int maxduration; // the timewindow for saved conceptlists.
  string type; //The type of the history.
};

#endif
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: typelist.h
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: Data structure which contains type histories.
//
////////////////////////////////////////////////////////////////////////////////
////////
```

```
#include "typelist.h"

//  public:

/**
 * Constructor.
 */

DSTypeHisList::DSTypeHisList()
{
  hypothesis = 0;
}

   /**
    * Adds a concept to the oppropriate temporary list.
    * If the concept isn't used before, a new history list is created.
    * @param concept Concept to be added.
    */

void DSTypeHisList::tmpAdd(DSConcept concept)
{
  DSConcept referent;
  if (concept.getReferent() != NULL)
  {
     referent = *concept.getReferent();
     referent.setTimestamp(concept.getTimestamp());
     concept = referent;
  }
  cout << "temp adding type: " << concept.getType() << ", value: " << concept.getValue()
<< endl;
  if (typehislist.count(concept.getType()) == 0) // if there isn't a history list of the
type of the concept
  {
     DSTypeHistory tmp(concept.getType());
     tmp.tempAdd(concept); // create a new history list of the type of the concept and
add the concept.
     typehislist[concept.getType()] = tmp; //add the new history list to the set of
history lists.
     keylist.push_back(concept.getType()); //add the key to the list of keys.
  }
  else
  {
     typehislist[concept.getType()].tempAdd(concept);  // add the concept to the history
list of the type of the concept
  }
}


   /**
    * Adds a concept to the oppropriate list. If the concept isn't used before, a new
history list is created.
    * @param concept Concept to be added.
    */

void DSTypeHisList::add(DSConcept concept)
{
  DSConcept referent;
  if (concept.getReferent() != NULL)
  {
     referent = *concept.getReferent();
     referent.setTimestamp(concept.getTimestamp());
     concept = referent;
  }
  cout << "adding: " << concept.getType() << " (" << concept.getValue() << ") to list" <<
endl;
  if (typehislist.count(concept.getType()) == 0) // if there isn't a history list of the
type of the concept
  {
     DSTypeHistory tmp(concept.getType());
```

```
     tmp.add(concept); // create a new history list of the type of the concept and add
the concept.
     typehislist[concept.getType()] = tmp; // add the new history list to the set of
history lists
     keylist.push_back(concept.getType()); // add the key to the list of keys.
  }
  else
    typehislist[concept.getType()].add(concept); // add the concept to the history list
of the type of the concept.
}

   /**
    * Saves the temporary history lists of hypothesis i of each concept.
    * @param i The index of the temporary history lists to be saved.
    */

void DSTypeHisList::save(int i)
{

  int size = keylist.size();
  for(int j=0; j<size; j++) // for every concept type history
  {
     typehislist[keylist[j]].save(i); // save hypothesis i
  }
  hypothesis = 0; // reset the hypothesis number in order for the new round to start with
0 again.
}

   /**
    * Finalizes the history lists. Each temporary history list is added to the list of
temporary history lists.
    * The temporary history list is reseted to the saved list.
    */

void DSTypeHisList::finalize()
{
  int size = keylist.size();
  for(int j=0; j<size; j++) // for every concept type history (read from keylist)
  {
    //   cout << "going to finalize type history of " << keylist[j] << ", which is in
the list? " << typehislist.count(keylist[j]) << endl;
     typehislist[keylist[j]].finalize(hypothesis); // finalize
  }
  hypothesis++;
}

   /**
    * Returns the size of the temporary history list of a certain type.
    * @param type The type of the history list, of which the size is requested.
    * @return The size of the temporary history list of the specified type.
    */

int DSTypeHisList::tempSize(string type)
{
  if (typehislist.count(type) != 0) //check whether the type is in the map.
    return typehislist[type].tempSize();
  else
    return 0;
}

   /**
    * Returns the size of the history list of a certain type.
    * @param type The type of the history list, of which the size is requested.
    * @return The size of the history list of the specified type.
    */

int DSTypeHisList::size(string type)
{
  if (typehislist.count(type) != 0) //check whether the type is in the map.
    return typehislist[type].size();
  else
```

```
    return 0;
}

   /**
    * Returns the concept at a certain position from the temporary history list of a
certain type.
    * @param type The type of the concept.
    * @param i The index of the concept.
    * @return The concept at position i of the specified type.
    */

DSConcept DSTypeHisList::getTemp(string type, int i)
{
  if (typehislist.count(type) != 0) //check whether the type is in the map.
    return typehislist[type].getTemp(i);
  else
  {
     DSConcept nothing;
     return nothing;
  }
}


   /**
    * Returns the concept at a certain position from the history list of a certain type.
    * @param type The type of the concept.
    * @param i The index of the concept.
    * @return The concept at position i of the specified type.
    */

DSConcept DSTypeHisList::get(string type, int i)
{
  if (typehislist.count(type) != 0) //check whether the type is in the map.
    return typehislist[type].get(i);
  else
  {
     DSConcept nothing;
     return nothing; // return nothing
  }
}


/**
    * Returns the list of keys.
    * @return The list of keys.
    */

vector<string> DSTypeHisList::getKeyList()
{
  return keylist;
}

//  protected:
   // none

//  private:
  // none

//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: typehis.cc
//  Revision:
//
//  Last changed by:
```

```
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the history list of a concept type, used to determine the
referent of a definite description
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "typehis.h"

//public:

/**
 * Constructor.
 */

DSTypeHistory::DSTypeHistory()
{
  maxSize = 5000;
  maxduration = 1000;
}

/**
 * Constructor.
 */

DSTypeHistory::DSTypeHistory(string type)
{
  this->type = type;
  maxSize = 5000;
  maxduration = 1000;
}

  /**
   * Returns the size of the type history.
   * @return The size of the type history.
   */

int DSTypeHistory::size()
{
  return saved.size();
}

     /**
   * Returns the size of the temporarily type history.
   * @return The size of the temporarily type history.
   */

int DSTypeHistory::tempSize()
{
  return temp.size();
}

  /**
   * Saves temporary type history i.
   * The temporary list is cleared.
   * @param i The number of the list to be saved.
   */

void DSTypeHistory::save(int i)
{
  saved = tempList[i];
  tempList.clear();
  temp = saved;
}
   /**
    * Adds a concept to the temporary history. If the number of concepts exceeds maxSize
then the oldest is discarded.
```

```
    * maxSize is defined in "typehis.cfg".
    * @param concept The concept to be added to the temporary list.
    */

void DSTypeHistory::tempAdd(DSConcept concept)
{
  if (temp.size() != 0)
  {
     while ((concept.getTimestamp() - temp.front().getTimestamp()) > maxduration)
     {
        temp.pop_front();
     }
  }
  if (concept.getReferent() != NULL)
  {
    temp.push_back(*(concept.getReferent()));
    temp[temp.size()-1].setTimestamp(concept.getTimestamp());
  }
  else
  {
     temp.push_back(concept);
  }
  if (temp.size() > maxSize)
    temp.pop_front();
}

   /**
    * Adds a concept list to the saved history. If the number of concepts exceeds maxSize
then the oldest is discarde.
    * maxSize is defined in "typehis.cfg".
    * @param clist The concept to be added to the saved list.
    */

void DSTypeHistory:: add(DSConcept concept)
{
  if (temp.size() != 0)
  {
     while ((concept.getTimestamp() - saved.front().getTimestamp()) > maxduration)
     {
        saved.pop_front();
     }
  }
  if (concept.getReferent() != NULL)
  {
    saved.push_back(*(concept.getReferent()));
  }
  else
  {
     saved.push_back(concept);
  }
  if (saved.size() > maxSize)
    saved.pop_front();
  temp = saved;
}
   /**
    * Returns the saved concept i.
    * @param i The index of the saved concept.
    * @return The last saved concept i.
    */

DSConcept  DSTypeHistory::get(int i)
{
  return saved[i];
}

   /**
    * Returns the temporarily saved concept i.
    * @param i The index of the temporarily saved concept.
    * @return The temporarily saved concept i.
    */
```

```
DSConcept  DSTypeHistory::getTemp(int i)
{
  return temp[i];
}

   /**
    * Finalizes the temporary type history. This pushes the temporary type history on the
list and resets the history.
    * @param hypothesis The number of the hypothesis to be finalized.
    */

void  DSTypeHistory::finalize(int hypothesis)
{
  if (tempList.size() == hypothesis)
  {
     tempList.push_back(temp);
  }
  else
  {
     for (int i = tempList.size(); i < hypothesis; i++)
     {
        tempList.push_back(saved);
     }
     tempList.push_back(temp);
  }
//  cout << "updated templist" << endl;
  temp = saved;
//  cout << "reassigned temp" << endl;
}

   /**
    * Sets the type of the type history.
    * @param type The type of the type history.
    */

void DSTypeHistory::setType(string type)
{
  this->type = type;
}

   /**
    * Returns the type of the type history.
    * @return The type of the type history.
    */

string DSTypeHistory::getType()
{
  return type;
}


//protected:
  //none

//private:
  // none
```

Grouping Module

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
```

```
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: listproc.h
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This module, creates from a list a concept, which can be referred to
pronominally.
//               For instance when a list of movies is shown, the user say something
like:
//               "ok, record them". In this case them would refer to all movies presented
on the display.
//               So a grouping concept must be made which can be referred to
pronominally.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef LISTPROC_H
#define LISTPROC_H

#include "concept.h"
#include <set>
#include <string>

class DSSystemListProcessor
{
public:

   /**
    * Constructor.
    */

   DSSystemListProcessor();

   /**
    * Creates a concept from a list of concepts, which can be referred to pronominally.
    * @param conlist Concept list, which contains the list of concepts, which can be
pronominally referred to.
    * @return A concept list, expanded with the concept, which can be pronominally
referred to.
    */

   vector<DSConcept> processList(vector<DSConcept> conlist);

private:
   bool on; //switch this module on or off.
   int number; //number indicating the number of the processed list.
   set<string> slistTypes; //set of types which must be processed.
};

#endif //LISTPROC_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
```

```
//              All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: listproc.cc
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This module, creates from a list a concept, which can be referred to
pronominally.
//               For instance when a list of movies is shown, the user say something
like:
//               "ok, record them". In this case them would refer to all movies presented
on the display.
//               So a grouping concept must be made which can be referred to
pronominally.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "listproc.h"
#include "myUtils.h"

#include <iostream>
#include <fstream>
#include <string>
#include <vector>

//public:

   /**
    * Constructor.
    */

DSSystemListProcessor::DSSystemListProcessor()
{
  ifstream in;
  string tmp;

  in.open("listproc.txt");
  if (!in)
  {
     cerr << "Cannot open SystemListProcessor initiation file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in,tmp);
     if ((tmp.find("#")!=string::npos)||(tmp==""))
     {
        // do nothing;
     }
     else if (tmp=="0")
     {
        cout << "System List Processor is OFF" << endl;
        on = false;
     }
     else if (tmp=="1")
     {
        cout << "System List Processor is ON" << endl;
        on = true;
     }
     else
     {
        cout << "inserted: " << tmp << endl;
```

```
        slistTypes.insert(tmp);
      }
  }
  number = 0;
  cout << "Done initiating SystemListProcessor" << endl;

}

   /**
    * Creates a concept from a list of concepts, which can be referred to pronominally.
    * @param conlist Concept list, which contains the list of concepts, which can be
pronominally referred to.
    * @return A concept list, expanded with the concept, which can be pronominally
referred to.
    */

vector<DSConcept> DSSystemListProcessor::processList(vector<DSConcept> conlist)
{
  myUtils util;

  vector<DSConcept> listEntries, res;
  if (!on)
  {
     cout << "System List Processor is OFF" << endl;
     return conlist; // module is not on, so do nothing.
  }
  cout << "System List Processor is ON" << endl;
  for (int i=0; i < conlist.size(); i++)
  {
     res.push_back(conlist[i]);
     if (conlist[i].getListEntries()!= NULL)
     {
       vector <DSConcept> concepts = *conlist[i].getListEntries();
        vector <DSConcept> subconcepts;
        set <string> groups;
        map <string, int> groupindex;
        vector <DSConcept> conceptgroups;
        vector <DSConcept> *listentries;
        vector <DSConcept> *subconceptentries;
        for (x=0; x < conlist.size(); x++)
        {
           if  (concepts.getSubConcepts() != NULL)
             subconcepts = *concepts[x].getSubConcepts();
           else
              subconcepts.clear();
           for (y=0; y < subconcepts.size(); y++)
           {
               if (groups.count(subconcepts[y].getValue()) == 0)
               {
                     groups.insert(subconcepts[y].getValue());
                     groupindex[subconcepts[y].getValue()] =
        conceptgroups.size();
                     conceptgroups.push_back(DSConcept("programs",
        subconcepts.getValue() + "s" , subconcepts[y].getTimestamp()));
                     listentries = new vector <DSConcept>;
                     listentries->push_back(concepts[x]);
                     *subconceptentries = *subconcepts;
                     conceptgroups[conceptgroups.size()-
        1].setListEntries(listentries);
                     conceptgroups[conceptgroups.size()-
        1].setSubConcepts(subconceptentries);
               }
               else
```

```
                {
                    listentries = conceptgroups[ groupindex[
        subconcepts[y].getValue()]].getListEntries();
                    conceptgroups[groupindex[subconcepts[y].getValue()]]
        .setListEntries(listentries);
                    subconceptentries =
        conceptgroups[groupindex[subconcepts[y].getValue()]].getSubConce
        pts ();
                    for (z=0; z <subconcepts.size(); z++)
                    {
                        subconceptentries->push_back(subconcepts[z]);
                    }

        conceptgroups[groupindex[subconcepts[y].getValue()]].setSubConce
        pt (subconceptentries);
                }
           }
        }
        for (x=0; x < conceptgroups.size(); x++) //remove groups with
        only 1 entry.
        {
           if (conceptgroups[x].getListEntries()->size() > 1)
           {
               res.push_back(conceptgroups[x]);
           }
        }
       if (slistTypes.count(conlist[i].getType())!=0)
       {
          cout << "start processing" << endl;
          listEntries = *conlist[i].getListEntries();
          if (listEntries.size()>1)
          {
             res.push_back(conlist[i]);
             res[res.size()-1].setValue(listEntries[0].getType()+"s
"+util.int2Str(number));
             cout << "added: " << res[res.size()-1].getValue() << endl;
             res[res.size()-1].setType(listEntries[0].getType());
             res[res.size()-1].setInputOrigin("slist");
             number ++;
             for (int j=listEntries.size()-1; j >=0; j--) // add the list entries as
separate items, backwards, so that items higher in the list gain priority.
             {
                res.push_back(listEntries[j]);
             }
          }
          else
          {
             res.push_back(listEntries[0]); // only one.
             res[res.size()-1].setInputOrigin("slist"); // mark that it must be added to
the slist.
          }
       }
    }
  }
  return res;
}
```

Deixis filter

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: deixdet.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the deixis detection module. It looks up the concept type
string. e.g. deixis.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSDEIXISDETECTION_H
#define DSDEIXISDETECTION_H

#include "concept.h"
#include <set>
#include <vector>

/**
 * This class is used to find the concepts which are derived from deixis.
 */

class DSDeixisDetection
{
  public:

   /**
    * Constructor.
    */

   DSDeixisDetection();

   /**
    * Extract the concepts which are derived from deixis. These are removed from the
list.
    * @param concepts List of concepts from which the deictic input must be extracted.
    * @return List of concepts of deictic input.
    */

   vector<DSConcept> extractDeixis(vector<DSConcept> &concepts);

protected:
   // none
private:
   set<string> deixisTypes; //types which indicates deixis.
};
```

```
#endif // DSDEIXISDETECTION_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: deixdet.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the deixis detection module. It looks up the concept type
string. e.g. deixis.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "deixdet.h"

//  public:

/**
 * Constructor.
 */

DSDeixisDetection::DSDeixisDetection()
{
  deixisTypes.insert("deixis"); // load the set of deixis types
}

   /**
    * Extract the concepts which are derived from deixis. These are removed from the
list.
    * @param concepts List of concepts from which the deictic input must be extracted.
    * @return List of concepts of deictic input.
    */

vector<DSConcept> DSDeixisDetection::extractDeixis(vector<DSConcept> &concepts)
{
  vector<DSConcept> res;
  vector<DSConcept> tmp;

  cout << "detect deixis, size = " << concepts.size() << endl;

  for (int i=0; i < concepts.size(); i++)
  {
     if (deixisTypes.count(concepts[i].getInputOrigin()) != 0) // if the input origin of
the concept is a form of deixis.
     {
        cout << "deixis detected: " << concepts[i].getValue() << endl;

        res.push_back(concepts[i]); // add it to the list of deictic concepts
     }
```

```
     else // create a list without deictic concepts
     {
        tmp.push_back(concepts[i]);
     }
  }
  concepts = tmp; // concepts is the list without deictic concepts.
  return res;
}

//  protected:
   // none

//  private:
   //none
```

Reference Detection & Classification Module

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: refdet.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the Reference Detection and Classification Module. It determines
the referential property
//               and classifies them in one of these categories:
//               - pronoun
//               - definite description
//               - date
//               - demonstrative
//               - one-anaphora
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSREFERENCECLASSIFICATIONANDDETECTION_H
#define DSREFERENCECLASSIFICATIONANDDETECTION_H

#include "conlist.h"
#include "concept.h"
#include <set>
#include "string"

   /**
    * Enumeration type to indicate the referential property:
    * - none,
    * - a pronoun,
    * - a demonstrative,
```

```
    * - a definite description,
    * - one anaphora, or
    * - a date.
    */

   enum referenceType {none, pronoun, demonstrative, definite, one, date};

/**
 * This class is used to determine the referential property of a concept and classify
them in the proper category.
 */

class DSReferenceDetectionAndClassification
{

  public:

   /**
    * Constructor.
    */

   DSReferenceDetectionAndClassification();

   /**
    * This method is used to determine the referential property of a concept and classify
them in the proper category.
    * @param currentConcept The concept to be classified.
    * @return The classification.
    */

   referenceType detectAndClassify (DSConcept currentConcept);

private:

   set<string> pronouns; // types indicating a pronoun.
   set<string> demonstratives; // types indicating a demonstrative.
   set<string> definites; // types indicating a definite description.
   set<string> ones; // types indicating one anaphora.
   set<string> dates; // types indicating a date.

   set<string> pronounInd; // set of words that indicate a pronoun.
   set<string> demonstrativeInd; // set of words that indicate a demonstrative.
   set<string> definiteInd; // set of words that indicate a definite description.
   set<string> oneInd; // set of words that indicate one anaphora.
   set<string> dateInd; // set of words that indicate a date.
   vector<string> posMod; //set of possesive modifiers like 's.

};

#endif // DSREFERENCECLASSIFICATIONANDDETECTION_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: refdet.cc
//  Revision:
//
//  Last changed by:
```

```
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the Reference Detection and Classification Module. It determines
the referential property
//               and classifies them in one of these categories:
//               - pronoun
//               - definite description
//               - date
//               - demonstrative
//               - one-anaphora
//
////////////////////////////////////////////////////////////////////////////////////////
////////

#include "refdet.h"
#include <fstream>

//  public:

/**
 * Constructor.
 */

DSReferenceDetectionAndClassification::DSReferenceDetectionAndClassification()
{
  pronouns.insert("Pronoun"); // initialize types indicating a pronoun.
  demonstratives.insert("Demonstrative"); // initialize types indicating a demonstrative.
  definites.insert("Definite"); //initialize types indicating a definite description.
  ones.insert("One"); //initialize types indicating one anaphora.
  dates.insert("Date"); //initialize types indicating a date.

  string tmp;
  cout << "loading reference indicators" << endl;
  ifstream in;
  in.open("pronouns.txt", ios::in);//initialize set of words that indicate a pronoun.
These words aren't accompanied by other words.
  if (!in)
  {
     cerr << "Cannot open pronoun data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        pronounInd.insert(tmp);
     }
  }
  in.close();
  in.open("demonstratives.txt", ios::in); //initialize set of words that indicate a
demonstrative. Usually referring to a deictic input.
  if (!in)
  {
     cerr << "Cannot open names and demonstratives data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        demonstrativeInd.insert(tmp);
     }
  }
  in.close();
  in.open("definites.txt", ios::in);//initialize set of words that indicate a definite
description. These words occur in conjunction with other words
```

```
  if (!in)
  {
     cerr << "Cannot open definite description data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        definiteInd.insert(tmp);
     }
  }
  in.close();
  in.open("one.txt", ios::in);//initialize set of words that indicate one anaphora. These
will occur in conjunction with a demonstrative indicator, or a definite description
indicator.
  if (!in)
  {
     cerr << "Cannot open one anaphora data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        oneInd.insert(tmp);
     }
  }
  in.close();
 in.open("date.txt", ios::in); //initialize set of words that indicate a date. These
words may occur in conjunction with other words.
  if (!in)
  {
     cerr << "Cannot open dates data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        dateInd.insert(tmp);
     }
  }
  in.close();
  in.open("posmod.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open possesive modifier data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        if (tmp != "")
        {
           posMod.push_back(tmp);
        }
     }
  }
  in.close();
}
   /**
    * This method is used to determine the referential property of a concept and classify
them in the proper category.
    * @param currentConcept The concept to be classified.
    * @return The classification.
```

```
    */

referenceType DSReferenceDetectionAndClassification::detectAndClassify (DSConcept
currentConcept)
{
  int nextpos;

  cout << "detect and classify" << endl;

  if (currentConcept.getType()=="title" || currentConcept.getType()=="contents" ||
currentConcept.getType()=="info_command_title")
  {
    return none;
  }
  if (currentConcept.getType()=="given_date" ||
currentConcept.getType()=="time_and_time_duration")
    return date;

  string concept = currentConcept.getValue();
  //check for pronouns.
  if (pronounInd.count(concept)!=0)
    return pronoun;
  //check for demonstratives.
  if (demonstrativeInd.count(concept)!=0)
    return demonstrative;
  //check for simple dates.
  if (dateInd.count(concept)!=0)
    return date;
  //check for 's.
  //commented because of ambiguity with abbreviation if "is".
  /*for (int i=0; i < posMod.size(); i++)
  {
     if (concept.find(posMod[i]) != string::npos)
       return definite;
       }*/

  //check whether indicators exist.

  referenceType res = none;

  while(concept.find(" ") != string::npos)
  {
     nextpos = concept.find(" ");

     //cout << concept.substr(0,nextpos) << endl;

     if (oneInd.count(concept.substr(0,nextpos))!=0)
       return one;
     if (definiteInd.count(concept.substr(0,nextpos))!=0)
       res = definite;
     if (dateInd.count(concept.substr(0,nextpos))!=0)
       return date;
     if (demonstrativeInd.count(concept.substr(0,nextpos))!=0)
       res = demonstrative;
     // current word is not an indicator, so try next one.

     concept = concept.substr(nextpos+1);
  }
  //cout << concept << endl;
  // check whether last word is an indicator.
  if (oneInd.count(concept)!=0)
    return one;
  if (definiteInd.count(concept)!=0)
    return definite;
  if (dateInd.count(concept)!=0)
    return date;
  if (demonstrativeInd.count(concept)!=0)
    return demonstrative;
  // cout << "return res" << endl;
  return res;
}
```

```
// protected
   // none
// private
   // none
```

Constraint Detection Module

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: condet.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the Constraint Detection Module. It is used to detect
constraints for a reference,
//               to narrow down the scope of possible referents.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSCONSTRAINTDETECTION
#define DSCONSTRAINTDETECTION

#include "concept.h"
#include "constr.h"
#include <vector>
#include <map>
#include <set>
#include "typcons.h"


/**
 * This class is used to detect constraints for a reference, to narrow down the scope of
possible referents.
 */

class DSConstraintDetection
{
  public:

   /**
    * Constructor.
    */

   DSConstraintDetection();

   /**
```

```
    * Detect the constraints from a list of concepts for a reference.
    * @param reference The reference for which the constraints must be detected.
    * @param concepts List of concepts, from which the constraints must be derived.
    * @return List of constraints for the reference.
    */

   vector<DSConstraint> detectConstraints (DSConcept reference, DSConcept *superconcept,
vector<DSConcept> concepts);

  protected:
   // none

  private:

   map<string, int> valueConstraintIndex; // index for the value - constraint map.
   vector<vector<DSConstraint> > valueConstraintMap; // the value - constraint map.

   map<string, int> modifierConstraintIndex; // index for the modifier - constraint
premisses index.
   vector<vector<int> > modConstrPreIndex; // index for modifier constraint (premisses)
map.

   vector<vector<DSConstraint> > modifierConstraintMap; // the modifier - constraint map.
   vector<vector<DSConstraint> > modifierConstraintPreMap; // premisses for the modifier
constraints.

   set<string> hasValueTypeConstraint; // are there constraints linked to a type for this
value?
   vector<map<string,int> > valueTypeConstraintIndex; // constraint index for value
linked to a type
   vector<vector<DSConstraint> > valueTypeConstraint; // constraint for value linked to a
type.
   set<string> listTypes; //set of list types.
   vector<string> subConModVal; // string value of the modifier.
   vector<int> subConModArg; // number of the argument (1,2) that is the superconcept.
   vector<string> subConModPos; // position of the modifier: F(ront), M(middle), B(ack).

   void findSubValConstraints(DSConcept reference, string subValue, vector<DSConstraint>
&res, map<string, int> &foundTypes, string &conType);

   DSTypeConstraints typeConstraints;

};

#endif // DSCONSTRAINTDETECTION
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: condet.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
```

```
//  Created on: January 17, 2001
//
//  Description: This is the Constraint Detection Module. It is used to detect
constraints for a reference,
//               to narrow down the scope of possible referents.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "condet.h"
#include <fstream>
#include "myUtils.h"

// public:

   /**
    * Constructor.
    */

DSConstraintDetection::DSConstraintDetection()
{
  string conceptValue, tmp, type, value, premtype, premvalue, conceptType, typConValue,
typConType;
  int priority, typConPriority;
  vector<DSConstraint> tmpConstraints, tmpTypeConstraints;
  vector<DSConstraint> tmpPremisses;
  vector<int> tmpIndex;
  map<string,int> tmpValueTypeConstraintIndex;
  myUtils util;

  // load file which contains constraint data.
  cout << "loading constraint data, condet.cc" << endl;
  ifstream in;
  in.open("constraints.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open constraint data file" << endl;
     exit;
  }
  cout << "loading constraints" << endl;
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#")!=string::npos || tmp == "") // comment read.
     {
        // do nothing.
        // cout << "comment: " << tmp << endl;
     }
     else if (tmp.find(">;")!=string::npos) // end of constraint linked to concepttype
read.
     {
        hasValueTypeConstraint.insert(conceptValue); // there are constraints for this
value linked to the concept type
        valueTypeConstraint.push_back(tmpTypeConstraints); // add the constraints linked
to the concept type
        tmpValueTypeConstraintIndex[conceptType]=valueTypeConstraint.size()-1; // note the
index of the constraints
        tmpTypeConstraints.clear();
     }
     else if (tmp.find(">")!=string::npos) // concept type to link constraint to read.
     {
        conceptType=tmp.substr(tmp.find(">")+1);
     }
     else if (tmp.find(")")!=string::npos) // constraint linked to concepttype read.
     {
        typConType = tmp.substr(1,tmp.find(",")-1);
        tmp = tmp.substr(tmp.find(",")+2);
        typConValue = tmp.substr(0,tmp.find(","));
        typConPriority = util.str2Int(tmp.substr(tmp.find(",")+2));
        tmpTypeConstraints.push_back(DSConstraint(typConType,typConValue,
typConPriority));
```

```
     }
     else if (tmp.find(":")!=string::npos) // concept value read.
     {
        conceptValue = tmp.substr(0,tmp.find(":")); // save concept value.
        //cout << "concept value: " << conceptValue << endl;
     }
     else if (tmp.find(";")!=string::npos) //end of constraints for a conceptvalue read.
     {
        valueConstraintMap.push_back(tmpConstraints); // save the constraints.
        valueConstraintIndex[conceptValue] = valueConstraintMap.size()-1;
        tmpConstraints.clear();
        valueTypeConstraintIndex.push_back(tmpValueTypeConstraintIndex); // save the index
for the constraints linked to a type.
        tmpValueTypeConstraintIndex.clear();

        /*cout << "done adding constraints for concept value " << conceptValue << " at
index: " << valueConstraintIndex[conceptValue] << endl;
        cout << "number of constraints added: " <<
valueConstraintMap[valueConstraintIndex[conceptValue]].size() << endl;
        for (int i=0; i < valueConstraintMap[valueConstraintIndex[conceptValue]].size();
i++)
        {
            cout << "constraint value: " <<
valueConstraintMap[valueConstraintIndex[conceptValue]][i].getValue() << " type: " <<
valueConstraintMap[valueConstraintIndex[conceptValue]][i].getType() << endl;
        }*/
     }
     else if (tmp.find(",")!=string::npos) // constraint type value pair.
     {
        type = tmp.substr(0,tmp.find(","));
        tmp = tmp.substr(tmp.find(",")+2);
        value = tmp.substr(0,tmp.find(","));
        priority = util.str2Int(tmp.substr(tmp.find(",")+2));
        tmpConstraints.push_back(DSConstraint(type,value, priority));
     }
  }
  in.close();
  // load file containing modifier data.
  in.open("modifiers.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open modifier constraint data file" << endl;
     exit;
  }
  cout << "loading modifiers" << endl;
  while (!in.eof())
  {
     getline(in, tmp);
     //in >> tmp;
     if (tmp.find("#")!=string::npos || tmp == "") // comment read.
     {
        //do nothing.
        //cout << "comment: " << tmp << endl;
     }
     else if (tmp.find(":")!=string::npos) // concept value read.
     {
        conceptValue = tmp.substr(0,tmp.find(":")); // save concept value.
        //cout << "modifier value: " << conceptValue << endl;
     }
     else if (tmp.find(">;")!=string::npos) //end of premisses for a modifier value read.
     {
        modifierConstraintPreMap.push_back(tmpPremisses); // save the premisses.
        tmpIndex.push_back(modifierConstraintPreMap.size()-1); // save the index of the
premisses
        //cout << "premisses constraints size: " <<
modifierConstraintPreMap[tmpIndex.size()-1].size() << ", " << tmpPremisses.size() <<
endl;
        tmpPremisses.clear();
        //cout << "done adding premisses for modifier value " << conceptValue << endl;
     }
```

```
     else if ((tmp.find(">")!=string::npos)&&(tmp.find(",")!=string::npos)) //premisses
for a conceptvalue read.
     {
        premtype=tmp.substr(tmp.find(">")+1,tmp.find(",")-tmp.find(">")-1);
        //cout << "premisses type: " << premtype << endl;
        premvalue=tmp.substr(tmp.find(",")+2);
        //cout << "premisses value: " << premvalue << endl;
        tmpPremisses.push_back(DSConstraint(premtype,premvalue));
     }
     else if (tmp.find(");")!=string::npos) //end of constraints for a modifier value
read.
     {
        modifierConstraintMap.push_back(tmpConstraints); // save the constraints.
        // tmpIndex isn't used, since it should already be updated with the premisses
        //cout << "modifier constraints size: " << modifierConstraintMap[tmpIndex.size()-
1].size() << ", " << tmpConstraints.size() << endl;
        tmpConstraints.clear();
        //cout << "done adding constraints for modifier value " << conceptValue << endl;
     }
     else if ((tmp.find(")")!=string::npos)&&(tmp.find(",")!=string::npos)) //premisses
for a conceptvalue read.
     {
        type=tmp.substr(tmp.find(")")+1,tmp.find(",")-tmp.find(")")-1);
        //cout << "constraint type: " << type << endl;
        tmp=tmp.substr(tmp.find(",")+2);
        value = tmp.substr(0,tmp.find(","));
        priority = util.str2Int(tmp.substr(tmp.find(",")+2));
        tmpConstraints.push_back(DSConstraint(type,value, priority));
        //cout << "constraint value: " << value << endl;
     }
     else if (tmp.find(";")!=string::npos) //end of conceptvalue read.
     {
        modConstrPreIndex.push_back(tmpIndex); // save the index of premisses and
constraints.
        modifierConstraintIndex[conceptValue] = modConstrPreIndex.size()-1;
        tmpIndex.clear();
        //cout << "done with concept value " << conceptValue << endl;
     }
  }
  in.close();
  in.open("listTypes.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open list types data file" << endl;
     exit;
  }
  cout << "loading list types" << endl;
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#")!=string::npos || tmp == "") // comment read.
     {
        // do nothing.
        // cout << "comment: " << tmp << endl;
     }
     else
     {
        listTypes.insert(tmp);
     }
  }
  in.close();
  in.open("subConMod.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open subconcept modifier data file" << endl;
     exit;
  }
  cout << "Loading subconcept modifier data..." << endl;
  while (!in.eof())
  {
     getline(in, tmp);
```

```
     if (tmp.find("#")!=string::npos || tmp == "") // comment read.
     {
        // do nothing.
        // cout << "comment: " << tmp << endl;
     }
     else
     {
        subConModVal.push_back(tmp.substr(0,tmp.find(":")));
        cout << "sub concept modifier: " << tmp.substr(0,tmp.find(":")) << " ";
        subConModArg.push_back(util.str2Int(tmp.substr(tmp.find(":")+2,1)));
        cout << util.str2Int(tmp.substr(tmp.find(":")+2,1)) << " ";
         subConModPos.push_back(tmp.substr(tmp.find(":")+5,1));
         cout << tmp.substr(tmp.find(":")+5,1) << endl;
     }
  }
  in.close();
  in.open("typeconstraints.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open type constraint data file" << endl;
     exit;
  }
  cout << "loading type constraints" << endl;
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#")!=string::npos || tmp == "") // comment read.
     {
        // do nothing.
        //cout << "comment: " << tmp << endl;
     }
     else if (tmp.find(":")!=string::npos) // concept type read.
     {
        conceptType = tmp.substr(0,tmp.find(":")); // save concept value.
        //cout << "concept type: " << conceptType << endl;
     }
     else if (tmp.find(",")!=string::npos) // constraint type value pair.
     {
        type = tmp.substr(0,tmp.find(","));
        //cout << "constraint type: " << type << endl;
        value = tmp.substr(tmp.find(",")+2);
        //cout << "constraint value: " << value << endl;
        typeConstraints.set(conceptType, type, value); // save the concept type -
constraints
     }
  }
  in.close();
}
   /**
    * Detect the constraints from a list of concepts for a reference.
    * @param reference The reference for which the constraints must be detected.
    * @param concepts List of concepts, from which the constraints must be derived.
    * @return List of constraints for the reference.
    */

vector<DSConstraint> DSConstraintDetection::detectConstraints (DSConcept reference,
DSConcept *superconcept, vector<DSConcept> concepts)
{
  vector<DSConstraint> res;
  string subValue, tmpValue, conType;
  map<string, int> foundTypes;

  // look for constraints within the concept.
  cout << "looking for constraints within the concept" << endl;
  // first check whether there are constraints for the complete value.
  if (valueConstraintIndex.count(reference.getValue())!=0) // check whether the value
exists in the map.
  {
     if (hasValueTypeConstraint.count(reference.getValue())!=0) // check whether there
are constraints linked to a type.
     {
```

```
       if
(valueTypeConstraintIndex[valueConstraintIndex[reference.getValue()]].count(reference.get
Type())!=0)
       {
          res =
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[reference.getValue()]][
reference.getType()]];
          cout << "constraints linked to type added" << endl;
       }
       else
       {
          res = valueConstraintMap[valueConstraintIndex[reference.getValue()]];
       }
    }
   res = valueConstraintMap[valueConstraintIndex[reference.getValue()]];
  }
  cout << "constraints within the concept as a whole " << ((res.size()==0)?"not
found":"found") << endl;
  bool conLinkedToTypeAdded = false;
  if (res.size() == 0)
  {
     if (superconcept != NULL)
     {
        if (superconcept->getReferent()!= NULL)
        {
           cout << "constraint added: listvalue, " << superconcept->getReferent()-
>getValue() << endl;
           res.push_back(DSConstraint("listvalue", superconcept->getReferent()-
>getValue()+":"+superconcept->getReferent()->getType()));
        }
        else
        {
           cout << "constraint added: listvalue, " << superconcept->getValue() << endl;
           res.push_back(DSConstraint("listvalue", superconcept-
>getValue()+":"+superconcept->getType()));
        }
     }

     // look for each word in the string for constraints.
     cout << "look for each word in the string for constraints" << endl;
     if (reference.getValue().find(" ")!= string::npos) // then for parts of the value.
     {
        tmpValue = reference.getValue();
        //cout << "tmpvalue: " << tmpValue << endl;
        while (tmpValue.find(" ")!= string::npos)
        {
           subValue = tmpValue.substr(0,tmpValue.find(" ")); // take first subvalue
           tmpValue = tmpValue.substr(tmpValue.find(" ")+1); // rest value

           // cout << "subvalue: " << subValue << ", tmpvalue: " << tmpValue << endl;

           findSubValConstraints(reference, subValue, res, foundTypes, conType);

        } // end while
         //cout << "tmpValue: " <<  tmpValue << endl;
         findSubValConstraints(reference, tmpValue, res, foundTypes, conType);
     } // end if
  } // end if
  else // constraints found for concept value as a whole, now printing them
  {
    /*for (int i=0; i < res.size(); i++)
    {
       cout << "constraint type: " << res[i].getType() << " constraint value: " <<
res[i].getValue() << endl;
       foundTypes[res[i].getType()]=i;
    }*/
  }
  if (res.size()==0) // no constraints found in concept value, search in concept type
  {
    cout << "no constraints found in concept value, search in concept type" << endl;
    res = typeConstraints.get(reference.getType());
```

```
  }
  if (reference.getSubConcepts() != NULL) // look for constraints in the subconcept list.
  {
     cout << "looking for constraints in the subconcept list" << endl;
     vector<DSConcept> subConcepts = *reference.getSubConcepts();
     for (int i=0; i < subConcepts.size(); i++)
     {
        // subconcept type and value become constraint type and value for the concept.
        res.push_back(DSConstraint(subConcepts[i].getType(), subConcepts[i].getValue()));
        foundTypes[subConcepts[i].getType()]=res.size()-1;
     }
  }
  else
  {
    cout << "no subconcepts to look constraints for" << endl;
  }
  cout << "looking for constraints in the concept list" << endl;
  bool prem = true;
  vector<int> index;
  int ressize = res.size();
  // look for constraints in the concept list.
  for( int i=0; i < concepts.size(); i++) // for each concept in the concept list.
  {
     if (!(reference == concepts[i]))
     {
        cout << "working on concept: " << concepts[i].getValue() << endl;
        string conItype;
        if (concepts[i].getReferent()!=NULL)
        {
           conItype = concepts[i].getReferent()->getType();
           cout << "referent Type = " << conItype << endl;
        }
        else
          conItype = concepts[i].getType();

        /*if (listTypes.count(conItype) != 0)
        {
           if (concepts[i].getReferent()!=NULL)
           {
              cout << "constraint added: listvalue, " << concepts[i].getReferent()-
>getValue() << endl;
              res.push_back(DSConstraint("listvalue", concepts[i].getReferent()-
>getValue()+":"+superconcept->getReferent()->getType()));
           }
           else
           {
              cout << "constraint added: listvalue, " << concepts[i].getValue() << endl;
              res.push_back(DSConstraint("listvalue",
concepts[i].getValue()+":"+superconcept->getType()));
           }
        }
        else */ if (modifierConstraintIndex.count(concepts[i].getValue()) != 0) // check
whether the value exists in the map
        {
           for (int j=0; j <
modConstrPreIndex[modifierConstraintIndex[concepts[i].getValue()]].size(); j++)// for
each of the premisses - constraint pair.
           {
//            cout << "checking premisses constraint pair " << j << endl;
              index = modConstrPreIndex[modifierConstraintIndex[concepts[i].getValue()]];
              for (int k=0; k < modifierConstraintPreMap[index[j]].size(); k++) // check
each of the premisses
              {
                 // cout << "checking premisses : "
<<modifierConstraintPreMap[index[j]][k].getType() << ", " <<
modifierConstraintPreMap[index[j]][k].getValue() << endl;
                 for (int l=0; l < res.size(); l++) // with the constraints already in the
constraintlist.
                 {
                    // cout << "still working at l=" << l << ", size = " << res.size() <<
endl;
```

```
                    if (res[l].getType() ==
modifierConstraintPreMap[index[j]][k].getType()) // if the types are the same
                    {
                       // cout << "types are the same" << endl;
                       if ((res[l].getValue() != "none") &&
(res[l].getValue()!=modifierConstraintPreMap[index[j]][k].getValue()))
                       {
                          prem = false; // but the values differ, then the additional
constraints can't be assigned.
                          break; // there's no need to check further.
                       }
                       else // the premisses and constraints are from the same type and
value
                       {
                          break; // no need to look further for the constraint with the
same type.
                       }
                    }

                 } // end going through constraints already in the constraint list.

                 //cout << "still working at k=" << k << ", size= " <<
modifierConstraintPreMap[index[j]].size() << endl;

                 if (!prem) // one of the premisses don't hold
                 {
                    break; // no need to look further
                 }
              } // end checking each of the premisses

              cout << "done checking each of the premisses" << endl;
              if (prem) // the premisses hold
              {
                 cout << "premisses hold" << endl;
                 //cout << "size of modifier constraint map: "  <<
modifierConstraintMap[index[j]].size() << endl;
                 for(int k=0; k < modifierConstraintMap[index[j]].size(); k++) // add the
additional constraints.
                 {
                    //cout << "adding additional constraints, currently at position " << k
<< endl;
                    cout << "  adding constraint type: " <<
modifierConstraintMap[index[j]][k].getType() << ", " <<
modifierConstraintMap[index[j]][k].getValue() << endl;
                    res.push_back(modifierConstraintMap[index[j]][k]);
                 }
                 break; // no need to look further
              }
           } // end checking each of the modifier - constraint pair
        } // end if
     } // end if
  } // end checking each concept in the list
  if (res.size() == ressize)
  {
    cout << "constraints in the concept list not found..." << endl;
  }
  cout << "the following constraints were determined for " << reference.getType() << " ("
<< reference.getValue() << ") :" << endl;
  for (int i = 0; i < res.size(); i++)
  {
    cout << "   contraint: " << res[i].getType() << " (" << res[i].getValue() << ")" <<
endl;
  }
  cout << "end of constraints" << endl;
  return res;
}


void DSConstraintDetection::findSubValConstraints(DSConcept reference, string subValue,
vector<DSConstraint> &res, map<string, int> &foundTypes, string &conType)
{
```

```
   bool conLinkedToTypeAdded = false;

   if (valueConstraintIndex.count(subValue)!=0) // check whether the subValue exists in
the map.
   {
      if (hasValueTypeConstraint.count(subValue)!=0) // check whether there are
constraints linked to a type.
      {
        if
(valueTypeConstraintIndex[valueConstraintIndex[subValue]].count(reference.getType())!=0)
// is the type in the list?
        {
           for (int j=0; j <
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]].size(); j++)
           {
              conType =
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j].getType();
              if (foundTypes.count(conType)==0)
              {

res.push_back(valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]
][reference.getType()]][j]);// add the constraint.
                 cout << "  constraint type added: " << res[res.size()-1].getType() << ",
" << res[res.size()-1].getValue() << endl;
                 foundTypes[conType]=res.size()-1; //add constraint type to list of found
constraint types.
              }
              else if (res[foundTypes[conType]].getValue() !=
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j].getValue()) // values conflict
              {
                 if (res[foundTypes[conType]].getPriority() ==
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j].getPriority())
                 {
                    res[foundTypes[conType]].setValue("mixed"); // priorities are the
same, set as mixed.
                  }
                 else
                 {
                    if (res[foundTypes[conType]].getPriority() <
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j].getPriority())
                    {
                       res[foundTypes[conType]] =
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j]; // new priority is higher, so replace
                     }
                  }
              }// end else if (values conflict)
           }// end for
           cout << "constraints linked to type added" << endl;
           conLinkedToTypeAdded = true;
        }// end is the type in the list?
      } // end are there type related constraints?
      if (!conLinkedToTypeAdded)
      {
        cout << subValue << " has constraints to add, index = " <<
valueConstraintIndex[subValue] << endl;
        for (int j=0; j < valueConstraintMap[valueConstraintIndex[subValue]].size(); j++)
        {
           //cout << "adding constraint " << j << " of " <<
valueConstraintMap[valueConstraintIndex[subValue]].size() << endl;
           // maybe add rules on assigning constraints???
           conType = valueConstraintMap[valueConstraintIndex[subValue]][j].getType();
           if (foundTypes.count(conType)==0)
           {
              res.push_back(valueConstraintMap[valueConstraintIndex[subValue]][j]);// add
the constraint.
```

```
               cout << "constraint type added: " << res[res.size()-1].getType() << ", " <<
res[res.size()-1].getValue() << endl;
               foundTypes[conType]=res.size()-1; //add constraint type to list of found
constraint types.
            }
           else if (res[foundTypes[conType]].getValue() !=
valueConstraintMap[valueConstraintIndex[subValue]][j].getValue()) // values conflict
           {
              if (res[foundTypes[conType]].getPriority() ==
valueConstraintMap[valueConstraintIndex[subValue]][j].getPriority())
              {
                 res[foundTypes[conType]].setValue("mixed"); // priorities are the same,
set as mixed.
              }
              else
              {
                 if (res[foundTypes[conType]].getPriority() <
valueConstraintMap[valueConstraintIndex[subValue]][j].getPriority())
                 {
                    res[foundTypes[conType]] =
valueConstraintMap[valueConstraintIndex[subValue]][j];
                 }
              }
           }
        } // end for
     } // end if not type linked constraints added
  } // end if subvalue is in the map
}
```

Pronoun Resolution Module

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: pronres.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the Pronoun Resolution Module. It is used to resolve pronominal
references.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSPRONOUNRESOLUTION_H
#define DSPRONOUNRESOLUTION_H

#include "concept.h"
#include "constr.h"
```

```
#include "slist.h"
#include "typcons.h"
#include "condet.h"

#include <vector>
#include <string>

/**
 * The pronoun resolution class, used to resolve pronominal references.
 */

class DSPronounResolution
{
  public:

   /**
    * Constructor.
    */

   DSPronounResolution();

   /**
    * Sets the constraintdetection module
    * @param condet The constraint detection module.
    */

   void setModules(DSConstraintDetection *condet);

   /**
    * Sets the s-list for the module.
    * @param list The s-list for the module.
    */

   void setList(DS_SList *list);

   /**
    * Resolve the pronoun, using strubes algorithm.
    * @param reference The pronoun to be resolved.
    * @param conList List of constraints to narrow the scope of possible referents.
    */

   void resolve (DSConcept *reference, vector<DSConstraint> &conList);

  private:

   /**
    * Determine whether a concept value is compatible with the list of constraints.
    * @param type The concept value.
    * @param constraints The list of constraints.
    * @return The concept type is compatible with the list of constraints.
    */

   bool isCompatible(DSConcept concept, vector<DSConstraint> constraints);

   DS_SList *sList;
   vector<string> reflexives; //list of reflexive pronoun patterns.
   set<string> possesives; // list of possesives.
   set<string> personals; //list of personals.
   DSConstraintDetection *constraintDetectionModule;

   //DSTypeConstraints valueConstraints;

};

#endif // DSPRONOUNRESOLUTION_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: pronres.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the Pronoun Resolution Module. It is used to resolve pronominal
references.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "pronres.h"
#include <fstream>
#include <string>

//  public:

   /**
    * Constructor.
    */

DSPronounResolution::DSPronounResolution()
{
  string conceptValue, tmp, type, value;
  vector<DSConstraint> tmpConstraints;

  cout << "loading personals data" << endl;
  ifstream in;
  in.open("personal.txt");
  if(!in)
  {
     cerr << "Cannot open personals data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in,tmp);
     if (tmp != "")
       personals.insert(tmp); // add personal.
  }
  in.close();
  cout << "loading possessives data" << endl;
  in.open("possessives.txt");
  if(!in)
  {
     cerr << "Cannot open possessives data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in,tmp);
     if (tmp != "")
       possesives.insert(tmp); // add possesive.
  }
```

```
  in.close();
  cout << "loading reflexives data" << endl;
  in.open("reflexives.txt");
  if(!in)
  {
     cerr << "Cannot open reflexives data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in,tmp);
     if (tmp != "")
       reflexives.push_back(tmp); // add reflexive pronoun patterns
  }
  in.close();
}

void DSPronounResolution::setModules(DSConstraintDetection *condet)
{
  constraintDetectionModule = condet;
}

   /**
    * Sets the s-list for the module.
    * @param list The s-list for the module.
    */

void DSPronounResolution::setList(DS_SList *list)
{
  sList = list;
}

   /**
    * Resolve the pronoun, using strubes algorithm.
    * @param reference The pronoun to be resolved.
    * @param conList List of constraints to narrow the scope of possible referents.
    */

void DSPronounResolution::resolve (DSConcept *reference, vector<DSConstraint> &conList)
{
  vector<string> conceptTypes;
  DSConcept *referent = NULL;
  int type, concept;
  int mostrecent = 0;
  bool reflexive = 0;
  bool possesive = 0;

  cout << "constraints found, start resolving pronouns..." << endl;

  for (int i=0; i < reflexives.size(); i++)
  {
     if (reference->getValue().find(reflexives[i]) != string::npos)
     {
        cout << "pronoun is reflexive because it contains " << reflexives[i] << endl;

        reflexive = 1; // check whether the reference is a reflexive pronoun.
     }
  }
  if (personals.count(reference->getValue()) != 0)
  {
     cout << "pronoun is personal" << endl;

     sList->nextSentence(); // a personal pronoun indicates a next sentence.
  }
  if (possesives.count(reference->getValue()) != 0)
  {
     cout << "pronoun is possesive" << endl;

     possesive = 1; // check whether the reference is a possesive.
  }
//  cout << "still working" << endl;
```

```
  for (int i=0; i < sList->tempSize(); i++)
  {
//     cout << "getting most recent timestamp, currently at pos " << i << endl;
     sList->getTemp(i);
     if (sList->getTemp(i).getTimestamp() > mostrecent)
     {
       if (sList->getTemp(i).getTimestamp() < reference->getTimestamp()) // additional
constraint for cases of deixis.
       {
//        cout << "timestamp " << i << " more recent than previous" << endl;
          mostrecent = sList->getTemp(i).getTimestamp(); // assign the most recent
timestamp to mostrecent.
       }
     }
  }
//  cout << "most recent timestamp = " << mostrecent << endl;
  cout << "look up first compatible entry. size of s-list:" << sList->tempSize() << endl;
  for (int i=0; i < sList->tempSize(); i++) // look up the first compatible entry from
the s-list.
  {
     cout << "s-list is at position " << i << ", " << sList->getTemp(i).getValue() <<
endl;
//     cout << "s-list is at concept: " << sList->getTemp(i).getValue() << endl;

     if (isCompatible (sList->getTemp(i), conList)) // if compatible
     {
       cout << "concept is compatible according to constraints" << endl;

       // if the reference is not a reflexive or possesive pronoun, it is unlikely that
it is the most recent entry.
       if (((reflexive || possesive) && (sList->getTemp(i).getTimestamp() == mostrecent))
|| (sList->getTemp(i).getTimestamp() != mostrecent) || (sList->getSentence()!=sList-
>getSentenceNr(i)) ) // also if not in the same sentence then most recent entry is
possible.
       {
          cout << "concept is compatible according to position" << endl;

          referent = new DSConcept; //(DSConcept *) malloc (sizeof(DSConcept));
//allocate memory
//        cout << "memory allocated" << endl;
          (*referent) = sList->getTemp(i); //assign the concept as referent
//        cout << "values of the referent are assigned" << endl;
          break;
       }
     }
     // if none is found, may want to check for the most recent entry. Give probability
according to this fact.
  }
  //cout << "still working" << endl;
  if (referent != NULL)
  {
    if (referent->getReferent() != NULL)
    {
      referent = referent->getReferent();
    }
    reference->setReferent(referent);
    cout << "referent = " << referent->getValue() << endl;
  }
  else
  {
     cout << "no referent found" << endl;
  }
}

//  protected:
    // none

//  private:

   /**
    * Determine whether a concept value is compatible with the list of constraints.
```

```
    * @param type The concept value.
    * @param constraints The list of constraints.
    * @return The concept type is compatible with the list of constraints.
    */

bool DSPronounResolution::isCompatible(DSConcept concept, vector<DSConstraint>
constraints)
{
  bool result = 1;
  DSTypeConstraints conceptConstraints; // = valueConstraints;
  vector<DSConstraint> constrList;
  vector<DSConcept> emptyList;

  constrList = constraintDetectionModule->detectConstraints(concept, NULL, emptyList);
  for (int i=0; i < constrList.size(); i++)
  {
     conceptConstraints.set(concept.getValue(), constrList[i].getType(),
constrList[i].getValue());
  }
  cout << "checking for compatibility, size of constraints is " << constraints.size() <<
endl;
  for (int i = 0; i < constraints.size(); i++)
  {
     //cout << "still working at iteration " << i << endl;
     cout << "constraint type: " << constraints[i].getType() << endl;
     if ((conceptConstraints.get(concept.getValue(), constraints[i].getType()) !=
constraints[i].getValue())
         && conceptConstraints.get(concept.getValue(), constraints[i].getType()) !=
"none")
     {
       result = 0; // if the concept type it's constraint type's value isn't the same as
the one from the constraint list
       break; // and it isn't "none" then the concept type is not compatible and false
must be returned.
     }
  }
  return result;
}
```

Definite Description Resolution Module

Header file

```
//////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
//////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: defres.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
```

```
//  Description: This is the Definite Description Resolution Module. It is used to
resolve pronominal references.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSDEFINITEDESCRIPTIONRESOLUTION_H
#define DSDEFINITEDESCRIPTIONRESOLUTION_H

#include "concept.h"
#include "constr.h"
#include "concdet.h"
#include "condet.h"

#include "typelist.h"
#include "pronres.h"

/**
 * The definite description resolution class, used to resolve definite descriptions.
 */

class DSDefiniteDescriptionResolution
{

  public:

   /**
    * Constructor.
    */

   DSDefiniteDescriptionResolution();

   /**
    * Sets the type history list for the module.
    * @param list The type history list for the module.
    */

   void setList(DSTypeHisList *list);

   /**
    * Sets the pronoun resolution module, for use when words like 'his', 'her', etc. are
encountered.
    * @param pronres The pronoun resolution module.
    * @param condet The constraint detection resolution module.
    */

   void setModules(DSPronounResolution *pronres, DSConstraintDetection *condet);

   /**
    * Resolve the definite description, looking up the most recent compatible concept
value of the concept type.
    * @param reference The definite description to be resolved.
    * @param conList List of constraints to narrow the scope of possible referents.
    */

   void resolve (DSConcept *reference, vector<DSConstraint> &conList);

protected:
   // none

private:

   DSConceptDeterminer conceptDeterminerModule;
   DSTypeHisList *typeHisList;
   //DSTypeConstraints valueConstraints;
   DSPronounResolution *pronounResolutionModule;
   DSConstraintDetection *constraintDetectionModule;
   int recency;

   /**
    * Determine whether a concept value is compatible with the list of constraints.
```

```
    * @param type The concept value.
    * @param constraints The list of constraints.
    * @return The concept type is compatible with the list of constraints.
    */

    bool isCompatible(DSConcept concept, vector<DSConstraint> constraints);

   /**
    * Determine whether a concept value is compatible with the list of constraints.
    * @param type The concept value.
    * @param posCons Positional constraints for the value.
    * @param constraints The list of constraints.
    * @return The concept type is compatible with the list of constraints.
    */

     bool isCompatible(DSConstraint posCons, vector<DSConstraint> constraints);

};

#endif // DSDEFINITEDESCRIPTIONRESOLUTION_H
```

Implementation file

```
//////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
//////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: defres.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the Definite Description Resolution Module. It is used to
resolve pronominal references.
//
//////////////////////////////////////////////////////////////////////////////////
////////

#include "defres.h"
#include "myUtils.h"
#include <fstream>

//  public:

/**
 * Constructor
 */

DSDefiniteDescriptionResolution::DSDefiniteDescriptionResolution()
{

}

/**
 * Sets the type history list for the module.
```

```
 * @param list The type history list for the module.
 */

void DSDefiniteDescriptionResolution::setList(DSTypeHisList *list)
{
  typeHisList = list;
}

/**
 * Sets the pronoun resolution module, for use when words like 'his', 'her', etc. are
encountered.
 * @param pronres The pronoun resolution module.
 * @param condet The constraint detection resolution module.
 */

void DSDefiniteDescriptionResolution::setModules(DSPronounResolution *pronres,
DSConstraintDetection *condet)
{
  pronounResolutionModule = pronres;
  constraintDetectionModule = condet;
}


   /**
    * Resolve the definite description, looking up the most recent compatible concept
value of the concept type.
    * @param reference The definite description to be resolved.
    * @param conList List of constraints to narrow the scope of possible referents.
    */

void DSDefiniteDescriptionResolution::resolve (DSConcept *reference, vector<DSConstraint>
&conList)
{
  vector<string> conceptTypes;
  DSConcept *referent = NULL;
  int type, concept;

  // Todo: check for pronouns like 'his' 'her', etc.
  // solve them.

  cout << "constraints found, start resolving definite description..." << endl;

  conceptTypes = conceptDeterminerModule.determineConceptType(*reference, conList,
typeHisList->getKeyList());  // determine the concept types which history has to be
accessed.

  cout << "concept types determined: ";
  for (int i=0; i < conceptTypes.size(); i++)
  {
     cout << conceptTypes[i] << " ";
  }
  cout << endl;

  for (int i=0; i < conceptTypes.size(); i++) // for each possible concept type
  {
//     cout << "concept type " << i << ": " << conceptTypes[i] << endl;

     recency = 0; // set the recency value of the concept. When should it be
incremented???? every concept encoutered or when a compatible concept is
encountered?????? currently set to compatible...
     if (conceptTypes[i].find(".relativetimeposition") != string::npos) // look at the
listentries for a concept at a relative time position, ie earlier, later, etc.
     {
       int substring = conceptTypes[i].find(".relativetimeposition"); // find the index
of the substring to be removed.
       conceptTypes[i] = conceptTypes[i].erase(substring); // get the conceptType of
which the subconcepts should be accessed.
       cout << "looking at the listentries of : " << conceptTypes[i] << "for a relative
time position" << endl;
```

```
       int conceptPos = typeHisList->tempSize(conceptTypes[i])-1; // default last concept
type to search in
       // before getting the listEntries, check for list value constraint. Which
indicates the appropriate list to search in.
       for (int j=0; j < conList.size(); j++) // look for constraint indicating the
concept to search for
       {
          if (conList[j].getType()=="listvalue")
          {
             string listValue = conList[j].getValue(); // get the value of the concept to
search in.
             for (int k=conceptPos; k >=0; k--) // find the position of the appropriate
list to search in.
             {
               if (typeHisList->getTemp(conceptTypes[i],k).getValue() == listValue) //if
the concept is the one to be searched in
               {
                  conceptPos = k; // remember the position of the concept in the list
                  break; // done searching
               }
             }
             break; // done searching constraints
          }
       }
       string relativetimeposition = "";
       for (int j=0; j < conList.size(); j++) // find the relative timeposition
constraint
       {
          if (conList[j].getType()=="relativetimeposition")
          {
             relativetimeposition = conList[j].getValue(); // get the relative
timeposition constraint
             break; // done searching
          }
       }
       vector<DSConcept> listEntries;// get the listEntries of the last entry of the
appropriate type history list

       if (typeHisList->getTemp(conceptTypes[i],conceptPos).getListEntries() != NULL) //
get the list entries
       {
          listEntries=*(typeHisList-
>getTemp(conceptTypes[i],conceptPos).getListEntries());
       }
       else
       {
          cerr << "warning listEntries not found!!!!" << endl;
          exit;
       }
       for (int j=0; j < listEntries.size(); j++) // search for compatible list entries.
       {
          if (isCompatible (listEntries[j], conList)) // if compatible
          {
             cout << listEntries[j].getValue() << "is compatible" << endl;
             if (referent == NULL) // if there is no previous referent, assign it
             {
               referent = new DSConcept;
               *referent = listEntries[i];
             }
             else // there's a previous referent
             {
               myUtils util;
               vector<DSConcept> newsubconcepts;
               vector<DSConcept> refsubconcepts;
               if (listEntries[j].getSubConcepts() == NULL || referent->getSubConcepts()
== NULL)
               {
                  cerr << "no subconcepts found while looking for relative time
position!!!" << endl;
                  return;
               }
```

```
                else
                {
                   newsubconcepts = *listEntries[j].getSubConcepts(); // look at the
subconcepts for the time concept
                   refsubconcepts = *(referent->getSubConcepts());
                }
                int reftime, newtime;

                for (int k=0; k < newsubconcepts.size(); k++)
                {
                   if (newsubconcepts[k].getType() == "time")
                   {
                      newtime = util.str2Int(newsubconcepts[k].getValue());
                      break;
                   }
                }
                for (int k=0; k < refsubconcepts.size(); k++) // and again for the
existing referent
                {
                   if (refsubconcepts[k].getType() == "time")
                   {
                      reftime = util.str2Int(refsubconcepts[k].getValue());
                      break;
                   }
                }
                if(newtime > reftime) // compare to get the earlier/later one.
                {
                   if (relativetimeposition == "max") // if looking for highest value
then change referent
                   {
                      *referent = listEntries[j];
                   } // else keep current referent
                }
             } // end else (there's a previous referent)
          }// end if
       } // end for
     } // end if relative time position
     else if (conceptTypes[i].find(".subconcept") != string::npos) //if needed look at
the subconcept of the concept type
     {
        int substring = conceptTypes[i].find(".subconcept");
        conceptTypes[i] = conceptTypes[i].erase(substring);
        cout << "looking at the subconcepts of : " << conceptTypes[i] << endl;

        int conceptPos = typeHisList->tempSize(conceptTypes[i])-1; // default last concept
type to search in
        for (int j=0; j < conList.size(); j++) // look for constraints indicating the
concept to search for
        {
           if (conList[j].getType()=="listvalue")
           {
              string listValue = conList[i].getValue(); // get the value of the concept to
search in.
              for (int k=conceptPos; k >=0; k--) // find the position of the appropriate
list to search in.
              {
                 if (typeHisList->getTemp(conceptTypes[i],k).getValue() == listValue) //
if the concept is the on to be searched in
                 {
                    cout << "listValue " << listValue << " found." << endl;
                    conceptPos = k; // remember the position of the concept in the list
                    break; // done searching
                 }
              }
              break; // done searching for constraints
           }
        }

        vector<DSConcept> subConcepts;// get the subConcepts of the required concept
```

```
       if (typeHisList->getTemp(conceptTypes[i],conceptPos).getSubConcepts() != NULL) //
get the subconcepts
       {
          subConcepts=*(typeHisList-
>getTemp(conceptTypes[i],conceptPos).getSubConcepts());
       }
       else
       {
          cerr << "warning subConcepts not found!!!!" << endl;
          exit;
       }

       // look for compatible subconcept
       for (int j=0; j < subConcepts.size(); j++)
       {
          // cout << "checking for compatibility" << endl;

          if (isCompatible (subConcepts[j], conList)) // if compatible
          {
             cout << subConcepts[j].getValue() << "is compatible" << endl;

             if (referent == NULL) // if there is no previous referent, assign it
             {
                referent = new DSConcept;
                *referent = subConcepts[j];
                break; // go on with the next typeHisList.
             }
             else // if the new referent is more recent than the previous referent,
assign it
             {
                if(subConcepts[j].getTimestamp() > referent->getTimestamp())
                {
                   *referent = subConcepts[j];
                   break; // go on with the next typeHisList.
                }
             }
          } // end if is compatible
       } // end for
    } // end if subConcepts
    else if(conceptTypes[i].find(".listentries") != string::npos) //if needed look at
the listentries of the concept type
    {
       int substring = conceptTypes[i].find(".listentries"); // find the index of the
substring to be removed
       conceptTypes[i] = conceptTypes[i].erase(substring); // get the conceptType of
which the subconcepts should be accessed.
       cout << "looking at the listentries of : " << conceptTypes[i] << endl;

       int conceptPos = typeHisList->tempSize(conceptTypes[i])-1; // default last concept
type to be accessed
        // before getting the listEntries, check for list value constraint.Which
indicates the appropriate list to search in.

       for (int j=0; j < conList.size(); j++) // look for constraints indicating the
concept to search for
       {
          if (conList[j].getType()=="listvalue") // get the value of the concept to
search in
          {
             string listValue = conList[i].getValue();
             for (int k=conceptPos; k >=0; k--) // find the position of the appropriate
list to search in.
             {
                if (typeHisList->getTemp(conceptTypes[i],k).getValue() == listValue) //
if the concept is the one to be searched in
                {
                   cout << "listValue " << listValue << " found." << endl;
                   conceptPos = k; // remember the position of the concept in the list
                   break; // done searching
                }
             }
```

```
              break; // done searching for constraints
           }
        }

        vector<DSConcept> listEntries;// get the listEntries of the appropriate concept
(last if none specified)

        if (typeHisList->getTemp(conceptTypes[i],conceptPos).getListEntries() != NULL) //
get the listentries
        {
           listEntries=*(typeHisList-
>getTemp(conceptTypes[i],conceptPos).getListEntries());
        }
        else
        {
           cerr << "warning listEntries not found!!!!" << endl;
           exit;
        }

        // maybe should check first whether a title is meant?

        myUtils util;
        string pos;
        DSConstraint posConstraint; //positional constraint
         // look the xth concept up in the list.
        int position = 1;
        vector <int> compatibles;
        bool done = false;
        for (int j=0; j < listEntries.size(); j++)
        {
           pos = util.int2Str(position);
//         cout << pos << endl;
           posConstraint = DSConstraint("listentry",pos); // add position counted from
above.

           cout << "checking for compatibility of: " << listEntries[j].getValue() << endl;

           if (isCompatible (listEntries[j], conList)) // if compatible
           {
             cout << listEntries[j].getValue() << " is compatible, now checking for
position" << endl;
             compatibles.push_back(j);
             if (isCompatible(posConstraint, conList))
             {
               cout << listEntries[j].getValue() << " IS COMPATIBLE" << endl;

               if (referent == NULL) // if there is no previous referent, assign it
               {
                 referent = new DSConcept;
                 *referent = listEntries[j];
                 done = true;
                 break; // go on with the next typeHisList.
               }
               else // if the new referent is more recent than the previous referent,
assign it
               {
                 if(listEntries[j].getTimestamp() > referent->getTimestamp())
                 {
                   *referent = listEntries[j];
                   done = true;
                   break; // go on with the next typeHisList.
                 }
               }
             }
             position++;
           } // end if is compatible
         } // end for
        if (!done)
        {
          position = 1;
          for (int j = compatibles.size()-1; j >0; j--)
```

```
          {
            cout << "checking for compatibility of: " << listEntries[j].getValue() <<
endl;
            pos = "-" +util.int2Str(position);
            posConstraint = DSConstraint ("listentry", pos);
            if (isCompatible(posConstraint, conList))
             {
               cout << listEntries[j].getValue() << "is compatible" << endl;

               if (referent == NULL) // if there is no previous referent, assign it
               {
                 referent = new DSConcept;
                 *referent = listEntries[j];
                 done = true;
                 break; // go on with the next typeHisList.
               }
               else // if the new referent is more recent than the previous referent,
assign it
               {
                 if(listEntries[j].getTimestamp() > referent->getTimestamp())
                 {
                   *referent = listEntries[j];
                   done = true;
                   break; // go on with the next typeHisList.
                 }
               }
             }
             position++;
          }
        }
     } //end if listentry
     else // not an listentry
     {
       // look at slist first
        pronounResolutionModule->resolve(reference,conList);
       if (reference->getReferent()!=NULL)
       {
         referent = reference->getReferent();
       }
       else
       {
         for (int j=typeHisList->tempSize(conceptTypes[i])-1; j >= 0;  j--) // look up
the most recent compatible entry from the compatible concept types.
         {
           // cout << "check compatibility" << endl;
           if (isCompatible (typeHisList->getTemp(conceptTypes[i],j), conList)) // if
compatible
           {
             cout << typeHisList->getTemp(conceptTypes[i],j).getValue() << "is
compatible" << endl;
             if (referent == NULL) // if there is no previous referent, assign it
             {
                referent = new DSConcept;
                *referent = typeHisList->getTemp(conceptTypes[i],j);
                break; // go on with the next typeHisList.
             }
             else // if the new referent is more recent than the previous referent,
assign it
             {
                if(typeHisList->getTemp(conceptTypes[i],j).getTimestamp() > referent-
>getTimestamp())
                {
                   *referent = typeHisList->getTemp(conceptTypes[i],j);
                   break; // go on with the next typeHisList.
                }
             }
           }// end if
         }// end for
       }// end else
     }// end for
  }
```

```
//  cout << "done searching for referent" << endl;
  if (referent != NULL)
  {
    if (referent->getReferent() != NULL)
    {
      referent = referent->getReferent();
    }
    cout << "referent value is: " << referent->getValue() << endl;
  }
  reference->setReferent(referent);
}

//  protected:
    // none

//  private:

   /**
    * Determine whether a concept value is compatible with the list of constraints.
    * @param type The concept value.
    * @param constraints The list of constraints.
    * @return The concept type is compatible with the list of constraints.
    */

bool DSDefiniteDescriptionResolution::isCompatible(DSConcept concept,
vector<DSConstraint> constraints)
{
  DSTypeConstraints conceptConstraints;
  vector<DSConstraint> conList;
  vector<DSConcept> emptyList;
  myUtils util;

  cout << "isCompatible: detect constraints for candidate referent :" <<
concept.getValue() << endl;
  conList = constraintDetectionModule->detectConstraints(concept, NULL, emptyList); //
find the constraints of the concept.
  for (int i=0; i < conList.size(); i++)
  {
     conceptConstraints.set(concept.getValue(), conList[i].getType(),
conList[i].getValue());
  }
  conceptConstraints.set(concept.getValue(), "recency", "-"+util.int2Str(recency)); //
add recency constraint of the concept.

  if (concept.getSubConcepts() != NULL) // add subconcepts as constraint.
  {
     cout << "subconcepts detected" << endl;
     vector<DSConcept> subConcepts = *concept.getSubConcepts();
     for (int i = 0; i < subConcepts.size(); i++)
     {
        conceptConstraints.set(concept.getValue(), subConcepts[i].getType(),
subConcepts[i].getValue());
        cout << "  constraint: " << subConcepts[i].getType() << " (" <<
subConcepts[i].getValue() << ") added" << endl;
     }
  }
  cout << "done checking constraints, constraint size = " << constraints.size() << endl;
  bool result = 1;
  for (int i = 0; i < constraints.size(); i++) // check for all constraints whether there
is a conflict or not.
  {
     if ((conceptConstraints.get(concept.getValue(), constraints[i].getType()) !=
constraints[i].getValue())
         && conceptConstraints.get(concept.getValue(), constraints[i].getType()) !=
"none")
     {
        cout << "not compatible because: " << constraints[i].getValue() << "!=" <<
conceptConstraints.get(concept.getValue(), constraints[i].getType()) << endl;
        if (constraints[i].getType() == "recency")
          recency++;
```

```
       result = 0; // if the concept type it's constraint type's value isn't the same as
the one from the constraint list
       break; // and it isn't "none" then the concept type is not compatible and false
must be returned.
     }
  }
  return result;
}

   /**
    * Determine whether a concept value is compatible with the list of constraints.
    * @param type The concept value.
    * @param posCons Positional constraints for the value.
    * @param constraints The list of constraints.
    * @return The concept type is compatible with the list of constraints.
    */

bool DSDefiniteDescriptionResolution::isCompatible(DSConstraint posCons,
vector<DSConstraint> constraints)
{

  bool result = 1;
  for (int i = 0; i < constraints.size(); i++)
  {
     if (constraints[i].getType() == "listentry") // check positional constraint
     {
       if (constraints[i].getValue() != posCons.getValue())
       {
//        cout << "not compatible because of position: " << constraints[i].getValue() <<
"!=" << posCons.getValue() << endl;
          result = 0; //positions don't match!!!  // initialize valueConstraints.
          break;
       }
       break;
     }
  }
  return result;
}
```

Demonstrative Resolution Module

Header file

```
//////////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
//////////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: demres.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
```

```
//  Description: This is the Demonstrative Resolution Module. It is used to resolve
demonstrative references.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSDEMONSTRATIVERESOLUTION_H
#define DSDEMONSTRATIVERESOLUTION_H

#include "concept.h"
#include "constr.h"

#include <vector>
#include <string>
#include <set>

#include "defres.h"
#include "pronres.h"

/**
 * The demonstrative resolution class, used to resolve demonstrative references.
 */

class DSDemonstrativeResolution
{
  public:

  /**
   * constructor.
   */

  DSDemonstrativeResolution();

   /**
    * Sets the resolution modules needed.
    * @param pronMod the pronoun resolution module.
    * @param defMod the definite description resolution module.
    */

   void setModules(DSPronounResolution *pronMod, DSDefiniteDescriptionResolution
*defMod);

   /**
    * Resolve the demonstrative, check for either pronominal or definite description
properties, and let the respective module
    * handle them. If needed, additional constraints can be added.
    * @param reference The demonstrative to be resolved.
    * @param conList List of constraints to narrow the scope of possible referents.
    */

   void resolve (DSConcept *reference, vector<DSConstraint> &conList);

  private:

   DSDefiniteDescriptionResolution *definiteDescriptionResolutionModule;
   DSPronounResolution *pronounResolutionModule;

   /**
    * Checks whether the demonstrative has pronominal properties or not.
    * @param reference The reference value.
    * @return The demonstrative has pronominal properties.
    */

   bool pronominalProp(string reference);
   set<string> demonstratives;
};

#endif // DSDEMONSTRATIVERESOLUTION_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: demres.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the Demonstrative Resolution Module. It is used to resolve
demonstrative references.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "demres.h"
#include <fstream>

/**
 * The demonstrative resolution class, used to resolve demonstrative references.
 */


//  public:

/**
 * Constructor
 */

DSDemonstrativeResolution::DSDemonstrativeResolution()
{
  ifstream in;
  string tmp;
  in.open("demonstratives.txt", ios::in); //initialize set of words that indicate a
demonstrative. Usually referring to a deictic input.
  if (!in)
  {
     cerr << "Cannot open names and demonstratives data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        demonstratives.insert(tmp);
     }
  }
  in.close();
}
   /**
    * Sets the resolution modules needed.
    * @param pronMod the pronoun resolution module.
    * @param defMod the definite description resolution module.
```

```
    */

void DSDemonstrativeResolution::setModules(DSPronounResolution *pronMod,
DSDefiniteDescriptionResolution *defMod)
{
  pronounResolutionModule = pronMod;
  definiteDescriptionResolutionModule = defMod;
}
   /**
    * Resolve the demonstrative, check for either pronominal or definite description
properties, and let the respective module
    * handle them. If needed, additional constraints can be added.
    * @param reference The demonstrative to be resolved.
    * @param conList List of constraints to narrow the scope of possible referents.
    */

void DSDemonstrativeResolution::resolve (DSConcept *reference, vector<DSConstraint>
&conList)
{
  if (pronominalProp(reference->getValue()))
  {
     pronounResolutionModule->resolve(reference, conList);
  }
  else
  {
     definiteDescriptionResolutionModule->resolve(reference, conList);
  }
}

//  protected:
   // none

//  private:

   /**
    * Checks whether the demonstrative has pronominal properties or not.
    * @param reference The reference value.
    * @return The demonstrative has pronominal properties.
    */

bool DSDemonstrativeResolution::pronominalProp(string reference)
{
  if (demonstratives.count(reference) != 0)
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

One Anaphora Resolution Module

Header file

```
//////////////////////////////////////////////////////////////////////////////////
////////
//
//
//              Copyright (C) 2001 Philips GmbH Dialog Systems
//
//              All rights reserved
//
//
//
```

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: oneres.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the One Anaphora Resolution Module. It is used to resolve one
anaphora.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSONEANAPHORARESOLUTION_H
#define DSONEANAPHORARESOLUTION_H

#include "concept.h"
#include "constr.h"
#include <vector>

#include "defres.h"
#include "demres.h"

/**
 * The one anaphora resolution class, used to resolve one anaphora.
 */

class DSOneAnaphoraResolution
{
  public:

   /**
    * Constructor.
    */

   DSOneAnaphoraResolution();

   /**
    * Sets the resolution modules needed.
    * @param pronMod the pronoun resolution module.
    * @param defMod the definite description resolution module.
    */

   void setModules(DSDemonstrativeResolution *demMod, DSDefiniteDescriptionResolution
*defMod);

   /**
    * Resolve one anaphora, check for either pronominal or definite description
properties, and let the respective module
    * handle them. If needed, additional constraints can be added.
    * @param reference The one anaphora to be resolved.
    * @param conList List of constraints to narrow the scope of possible referents.
    */

   void resolve (DSConcept *reference, vector<DSConstraint> &conList);

private:

   DSDefiniteDescriptionResolution *definiteDescriptionResolutionModule;
   DSDemonstrativeResolution *demonstrativeResolutionModule;

   /**
    * Determines whether the reference has pronominal properties or not.
    * @param reference The reference.
    * @return The reference has pronoominal properties.
    */
```

```
   bool demonstrativeProp (string reference);
   set<string> demonstrativeInd; // set of words that indicate a demonstrative.
};

#endif //  DSONEANAPHORARESOLUTION_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: oneres.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the One Anaphora Resolution Module. It is used to resolve one
anaphora.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "oneres.h"
#include <fstream>

//  public:

/**
 * Constructor.
 */

DSOneAnaphoraResolution::DSOneAnaphoraResolution()
{
  string tmp;
  ifstream in;

  in.open("demonstratives.txt", ios::in); //initialize set of words that indicate a
demonstrative. Usually referring to a deictic input.
  if (!in)
  {
     cerr << "Cannot open names and demonstratives data file" << endl;
     exit;
  }
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#") == string::npos && tmp!="")
     {
        demonstrativeInd.insert(tmp);
     }
  }
  in.close();
}
```

```
   /**
    * Sets the resolution modules needed.
    * @param pronMod the pronoun resolution module.
    * @param defMod the definite description resolution module.
    */

void DSOneAnaphoraResolution::setModules(DSDemonstrativeResolution *demMod,
DSDefiniteDescriptionResolution *defMod)
{
  demonstrativeResolutionModule = demMod;
  definiteDescriptionResolutionModule = defMod;
}

   /**
    * Resolve one anaphora, check for either pronominal or definite description
properties, and let the respective module
    * handle them. If needed, additional constraints can be added.
    * @param reference The one anaphora to be resolved.
    * @param conList List of constraints to narrow the scope of possible referents.
    */

void DSOneAnaphoraResolution::resolve (DSConcept *reference, vector<DSConstraint>
&conList)
{
  if (demonstrativeProp(reference->getValue()))
  {
     demonstrativeResolutionModule->resolve(reference, conList);
  }
  else
  {
     definiteDescriptionResolutionModule->resolve(reference, conList);
  }

}

//  protected:
   // none

//  private:

/**
 * Determines whether the reference has pronominal properties or not.
 * @param reference The reference.
 * @return The reference has pronoominal properties.
 */

bool DSOneAnaphoraResolution::demonstrativeProp(string reference)
{
  int nextpos;
  while(reference.find(" ") != string::npos)
  {
     nextpos = reference.find(" ");
     if (demonstrativeInd.count(reference.substr(0,nextpos))!=0)
       return true;
     // current word is not an indicator, so try next one.
     reference = reference.substr(nextpos+1);
  }
  // check whether last word is an indicator.
  if (demonstrativeInd.count(reference)!=0)
    return true;
  return false;
}
```

Concept Type Filter

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: concdet.h
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the Concept Determiner Module. It is used to determine the
compatible concept types
//               of the reference.
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSCONCEPTDETERMINER_H
#define DSCONCEPTDETERMINER_H

#include <vector>
#include <string>
#include <map>

#include "concept.h"
#include "constr.h"
#include "typcons.h"

/**
 * This class is used to determine the compatible concept types of the reference.
 */

class DSConceptDeterminer
{
  public:

   /**
    * Constructor.
    */

   DSConceptDeterminer();


   /**
    * Determine the compatible concept types of the reference.
    * @param reference The reference for which the concept types must be determined.
    * @param constraints List of constraints, from which the concept types may be
determined.
    * @param types List of possible types.
    * @return List of compatible types of the reference.
    */
```

```
   vector<string> determineConceptType (DSConcept reference, vector<DSConstraint>
&constraints, vector<string> types);

  protected:
   // none

  private:

    DSTypeConstraints typeConstraints;

   /**
    * Determine whether a concept type is compatible with the list of constraints.
    * @param type The concept type.
    * @param constraints The list of constraints.
    * @return The concept type is compatible with the list of constraints.
    */

    bool isCompatible(string type, vector<DSConstraint> constraints);

    /**
    * Look for a concept type containing a concept type which is compatible with the list
of constraints.
    * @param types List of possible concept type.
    * @param constraints The list of constraints.
    * @return A list of concept types, compatible with the list of constraints.
    */

    vector<string> getCompatibleLists(vector<string> types, vector<DSConstraint>
constraints);

};

#endif // DSCONCEPTDETERMINER_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialogue Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: condet.cc
//  Revision:
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: This is the Constraint Detection Module. It is used to detect
constraints for a reference,
//               to narrow down the scope of possible referents.
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "condet.h"
#include <fstream>
#include "myUtils.h"

// public:
```

```
   /**
    * Constructor.
    */

DSConstraintDetection::DSConstraintDetection()
{
  string conceptValue, tmp, type, value, premtype, premvalue, conceptType, typConValue,
typConType;
  int priority, typConPriority;
  vector<DSConstraint> tmpConstraints, tmpTypeConstraints;
  vector<DSConstraint> tmpPremisses;
  vector<int> tmpIndex;
  map<string,int> tmpValueTypeConstraintIndex;
  myUtils util;

  // load file which contains constraint data.
  cout << "loading constraint data, condet.cc" << endl;
  ifstream in;
  in.open("constraints.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open constraint data file" << endl;
     exit;
  }
  cout << "loading constraints" << endl;
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#")!=string::npos || tmp == "") // comment read.
     {
        // do nothing.
        // cout << "comment: " << tmp << endl;
     }
     else if (tmp.find(">;")!=string::npos) // end of constraint linked to concepttype
read.
     {
        hasValueTypeConstraint.insert(conceptValue); // there are constraints for this
value linked to the concept type
        valueTypeConstraint.push_back(tmpTypeConstraints); // add the constraints linked
to the concept type
        tmpValueTypeConstraintIndex[conceptType]=valueTypeConstraint.size()-1; // note the
index of the constraints
        tmpTypeConstraints.clear();
     }
     else if (tmp.find(">")!=string::npos) // concept type to link constraint to read.
     {
        conceptType=tmp.substr(tmp.find(">")+1);
     }
     else if (tmp.find(")")!=string::npos) // constraint linked to concepttype read.
     {
        typConType = tmp.substr(1,tmp.find(",")-1);
        tmp = tmp.substr(tmp.find(",")+2);
        typConValue = tmp.substr(0,tmp.find(","));
        typConPriority = util.str2Int(tmp.substr(tmp.find(",")+2));
        tmpTypeConstraints.push_back(DSConstraint(typConType,typConValue,
typConPriority));
     }
     else if (tmp.find(":")!=string::npos) // concept value read.
     {
        conceptValue = tmp.substr(0,tmp.find(":")); // save concept value.
        //cout << "concept value: " << conceptValue << endl;
     }
     else if (tmp.find(";")!=string::npos) //end of constraints for a conceptvalue read.
     {
        valueConstraintMap.push_back(tmpConstraints); // save the constraints.
        valueConstraintIndex[conceptValue] = valueConstraintMap.size()-1;
        tmpConstraints.clear();
        valueTypeConstraintIndex.push_back(tmpValueTypeConstraintIndex); // save the index
for the constraints linked to a type.
        tmpValueTypeConstraintIndex.clear();
```

```
        /*cout << "done adding constraints for concept value " << conceptValue << " at
index: " << valueConstraintIndex[conceptValue] << endl;
        cout << "number of constraints added: " <<
valueConstraintMap[valueConstraintIndex[conceptValue]].size() << endl;
        for (int i=0; i < valueConstraintMap[valueConstraintIndex[conceptValue]].size();
i++)
        {
            cout << "constraint value: " <<
valueConstraintMap[valueConstraintIndex[conceptValue]][i].getValue() << " type: " <<
valueConstraintMap[valueConstraintIndex[conceptValue]][i].getType() << endl;
        }*/
     }
     else if (tmp.find(",")!=string::npos) // constraint type value pair.
     {
        type = tmp.substr(0,tmp.find(","));
        tmp = tmp.substr(tmp.find(",")+2);
        value = tmp.substr(0,tmp.find(","));
        priority = util.str2Int(tmp.substr(tmp.find(",")+2));
        tmpConstraints.push_back(DSConstraint(type,value, priority));
     }
  }
  in.close();
  // load file containing modifier data.
  in.open("modifiers.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open modifier constraint data file" << endl;
     exit;
  }
  cout << "loading modifiers" << endl;
  while (!in.eof())
  {
     getline(in, tmp);
     //in >> tmp;
     if (tmp.find("#")!=string::npos || tmp == "") // comment read.
     {
        //do nothing.
        //cout << "comment: " << tmp << endl;
     }
     else if (tmp.find(":")!=string::npos) // concept value read.
     {
        conceptValue = tmp.substr(0,tmp.find(":")); // save concept value.
        //cout << "modifier value: " << conceptValue << endl;
     }
     else if (tmp.find(">;")!=string::npos) //end of premisses for a modifier value read.
     {
        modifierConstraintPreMap.push_back(tmpPremisses); // save the premisses.
        tmpIndex.push_back(modifierConstraintPreMap.size()-1); // save the index of the
premisses
        //cout << "premisses constraints size: " <<
modifierConstraintPreMap[tmpIndex.size()-1].size() << ", " << tmpPremisses.size() <<
endl;
        tmpPremisses.clear();
        //cout << "done adding premisses for modifier value " << conceptValue << endl;
     }
     else if ((tmp.find(">")!=string::npos)&&(tmp.find(",")!=string::npos)) //premisses
for a conceptvalue read.
     {
        premtype=tmp.substr(tmp.find(">")+1,tmp.find(",")-tmp.find(">")-1);
        //cout << "premisses type: " << premtype << endl;
        premvalue=tmp.substr(tmp.find(",")+2);
        //cout << "premisses value: " << premvalue << endl;
        tmpPremisses.push_back(DSConstraint(premtype,premvalue));
     }
     else if (tmp.find(");")!=string::npos) //end of constraints for a modifier value
read.
     {
        modifierConstraintMap.push_back(tmpConstraints); // save the constraints.
        // tmpIndex isn't used, since it should already be updated with the premisses
```

```
        //cout << "modifier constraints size: " << modifierConstraintMap[tmpIndex.size()-
1].size() << ", " << tmpConstraints.size() << endl;
        tmpConstraints.clear();
        //cout << "done adding constraints for modifier value " << conceptValue << endl;
      }
      else if ((tmp.find(")")!=string::npos)&&(tmp.find(",")!=string::npos)) //premisses
for a conceptvalue read.
      {
        type=tmp.substr(tmp.find(")")+1,tmp.find(",")-tmp.find(")")-1);
        //cout << "constraint type: " << type << endl;
        tmp=tmp.substr(tmp.find(",")+2);
        value = tmp.substr(0,tmp.find(","));
        priority = util.str2Int(tmp.substr(tmp.find(",")+2));
        tmpConstraints.push_back(DSConstraint(type,value, priority));
        //cout << "constraint value: " << value << endl;
      }
      else if (tmp.find(";")!=string::npos) //end of conceptvalue read.
      {
        modConstrPreIndex.push_back(tmpIndex); // save the index of premisses and
constraints.
        modifierConstraintIndex[conceptValue] = modConstrPreIndex.size()-1;
        tmpIndex.clear();
        //cout << "done with concept value " << conceptValue << endl;
      }
   }
   in.close();
   in.open("listTypes.txt", ios::in);
   if (!in)
   {
      cerr << "Cannot open list types data file" << endl;
      exit;
   }
   cout << "loading list types" << endl;
   while (!in.eof())
   {
      getline(in, tmp);
      if (tmp.find("#")!=string::npos || tmp == "") // comment read.
      {
        // do nothing.
        // cout << "comment: " << tmp << endl;
      }
      else
      {
        listTypes.insert(tmp);
      }
   }
   in.close();
   in.open("subConMod.txt", ios::in);
   if (!in)
   {
      cerr << "Cannot open subconcept modifier data file" << endl;
      exit;
   }
   cout << "Loading subconcept modifier data..." << endl;
   while (!in.eof())
   {
      getline(in, tmp);
      if (tmp.find("#")!=string::npos || tmp == "") // comment read.
      {
        // do nothing.
        // cout << "comment: " << tmp << endl;
      }
      else
      {
        subConModVal.push_back(tmp.substr(0,tmp.find(":")));
        cout << "sub concept modifier: " << tmp.substr(0,tmp.find(":")) << " ";
        subConModArg.push_back(util.str2Int(tmp.substr(tmp.find(":")+2,1)));
        cout << util.str2Int(tmp.substr(tmp.find(":")+2,1)) << " ";
         subConModPos.push_back(tmp.substr(tmp.find(":")+5,1));
         cout << tmp.substr(tmp.find(":")+5,1) << endl;
      }
```

```
  }
  in.close();
  in.open("typeconstraints.txt", ios::in);
  if (!in)
  {
     cerr << "Cannot open type constraint data file" << endl;
     exit;
  }
  cout << "loading type constraints" << endl;
  while (!in.eof())
  {
     getline(in, tmp);
     if (tmp.find("#")!=string::npos || tmp == "") // comment read.
     {
        // do nothing.
        //cout << "comment: " << tmp << endl;
     }
     else if (tmp.find(":")!=string::npos) // concept type read.
     {
        conceptType = tmp.substr(0,tmp.find(":")); // save concept value.
        //cout << "concept type: " << conceptType << endl;
     }
     else if (tmp.find(",")!=string::npos) // constraint type value pair.
     {
        type = tmp.substr(0,tmp.find(","));
        //cout << "constraint type: " << type << endl;
        value = tmp.substr(tmp.find(",")+2);
        //cout << "constraint value: " << value << endl;
        typeConstraints.set(conceptType, type, value); // save the concept type -
constraints
     }
  }
  in.close();
}
   /**
    * Detect the constraints from a list of concepts for a reference.
    * @param reference The reference for which the constraints must be detected.
    * @param concepts List of concepts, from which the constraints must be derived.
    * @return List of constraints for the reference.
    */

vector<DSConstraint> DSConstraintDetection::detectConstraints (DSConcept reference,
DSConcept *superconcept, vector<DSConcept> concepts)
{
  vector<DSConstraint> res;
  string subValue, tmpValue, conType;
  map<string, int> foundTypes;

  // look for constraints within the concept.
  cout << "looking for constraints within the concept" << endl;
  // first check whether there are constraints for the complete value.
  if (valueConstraintIndex.count(reference.getValue())!=0) // check whether the value
exists in the map.
  {
     if (hasValueTypeConstraint.count(reference.getValue())!=0) // check whether there
are constraints linked to a type.
     {
        if
(valueTypeConstraintIndex[valueConstraintIndex[reference.getValue()]].count(reference.get
Type())!=0)
        {
           res =
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[reference.getValue()]][
reference.getType()]];
           cout << "constraints linked to type added" << endl;
        }
        else
        {
           res = valueConstraintMap[valueConstraintIndex[reference.getValue()]];
        }
     }
```

```
    res = valueConstraintMap[valueConstraintIndex[reference.getValue()]];
  }
  cout << "constraints within the concept as a whole " << ((res.size()==0)?"not
found":"found") << endl;
  bool conLinkedToTypeAdded = false;
  if (res.size() == 0)
  {
     if (superconcept != NULL)
     {
        if (superconcept->getReferent()!= NULL)
        {
           cout << "constraint added: listvalue, " << superconcept->getReferent()-
>getValue() << endl;
           res.push_back(DSConstraint("listvalue", superconcept->getReferent()-
>getValue()+":"+superconcept->getReferent()->getType()));
        }
        else
        {
           cout << "constraint added: listvalue, " << superconcept->getValue() << endl;
           res.push_back(DSConstraint("listvalue", superconcept-
>getValue()+":"+superconcept->getType()));
        }
     }

     // look for each word in the string for constraints.
     cout << "look for each word in the string for constraints" << endl;
     if (reference.getValue().find(" ")!= string::npos) // then for parts of the value.
     {
        tmpValue = reference.getValue();
        //cout << "tmpvalue: " << tmpValue << endl;
        while (tmpValue.find(" ")!= string::npos)
        {
           subValue = tmpValue.substr(0,tmpValue.find(" ")); // take first subvalue
           tmpValue = tmpValue.substr(tmpValue.find(" ")+1); // rest value

           // cout << "subvalue: " << subValue << ", tmpvalue: " << tmpValue << endl;

           findSubValConstraints(reference, subValue, res, foundTypes, conType);

        } // end while
         //cout << "tmpValue: " <<  tmpValue << endl;
         findSubValConstraints(reference, tmpValue, res, foundTypes, conType);
     } // end if
  } // end if
  else // constraints found for concept value as a whole, now printing them
  {
    /*for (int i=0; i < res.size(); i++)
    {
       cout << "constraint type: " << res[i].getType() << " constraint value: " <<
res[i].getValue() << endl;
       foundTypes[res[i].getType()]=i;
    }*/
  }
  if (res.size()==0) // no constraints found in concept value, search in concept type
  {
    cout << "no constraints found in concept value, search in concept type" << endl;
    res = typeConstraints.get(reference.getType());
  }
  if (reference.getSubConcepts() != NULL) // look for constraints in the subconcept list.
  {
     cout << "looking for constraints in the subconcept list" << endl;
     vector<DSConcept> subConcepts = *reference.getSubConcepts();
     for (int i=0; i < subConcepts.size(); i++)
     {
        // subconcept type and value become constraint type and value for the concept.
        res.push_back(DSConstraint(subConcepts[i].getType(), subConcepts[i].getValue()));
        foundTypes[subConcepts[i].getType()]=res.size()-1;
     }
  }
  else
  {
```

```
    cout << "no subconcepts to look constraints for" << endl;
  }
  cout << "looking for constraints in the concept list" << endl;
  bool prem = true;
  vector<int> index;
  int ressize = res.size();
  // look for constraints in the concept list.
  for( int i=0; i < concepts.size(); i++) // for each concept in the concept list.
  {
     if (!(reference == concepts[i]))
     {
        cout << "working on concept: " << concepts[i].getValue() << endl;
        string conItype;
        if (concepts[i].getReferent()!=NULL)
        {
           conItype = concepts[i].getReferent()->getType();
           cout << "referent Type = " << conItype << endl;
        }
        else
          conItype = concepts[i].getType();

        /*if (listTypes.count(conItype) != 0)
        {
           if (concepts[i].getReferent()!=NULL)
           {
              cout << "constraint added: listvalue, " << concepts[i].getReferent()-
>getValue() << endl;
              res.push_back(DSConstraint("listvalue", concepts[i].getReferent()-
>getValue()+":"+superconcept->getReferent()->getType()));
           }
           else
           {
              cout << "constraint added: listvalue, " << concepts[i].getValue() << endl;
              res.push_back(DSConstraint("listvalue",
concepts[i].getValue()+":"+superconcept->getType()));
           }
        }
        else */ if (modifierConstraintIndex.count(concepts[i].getValue()) != 0) // check
whether the value exists in the map
        {
           for (int j=0; j <
modConstrPreIndex[modifierConstraintIndex[concepts[i].getValue()]].size(); j++)// for
each of the premisses - constraint pair.
           {
//            cout << "checking premisses constraint pair " << j << endl;
              index = modConstrPreIndex[modifierConstraintIndex[concepts[i].getValue()]];
              for (int k=0; k < modifierConstraintPreMap[index[j]].size(); k++) // check
each of the premisses
              {
                 // cout << "checking premisses : "
<<modifierConstraintPreMap[index[j]][k].getType() << ", " <<
modifierConstraintPreMap[index[j]][k].getValue() << endl;
                for (int l=0; l < res.size(); l++) // with the constraints already in the
constraintlist.
                {
                   // cout << "still working at l=" << l << ", size = " << res.size() <<
endl;
                   if (res[l].getType() ==
modifierConstraintPreMap[index[j]][k].getType()) // if the types are the same
                   {
                      // cout << "types are the same" << endl;
                      if ((res[l].getValue() != "none") &&
(res[l].getValue()!=modifierConstraintPreMap[index[j]][k].getValue()))
                      {
                         prem = false; // but the values differ, then the additional
constraints can't be assigned.
                         break; // there's no need to check further.
                      }
                      else // the premisses and constraints are from the same type and
value
                      {
```

```
                        break; // no need to look further for the constraint with the
same type.
                     }
                  }

               } // end going through constraints already in the constraint list.

               //cout << "still working at k=" << k << ", size= " <<
modifierConstraintPreMap[index[j]].size() << endl;

               if (!prem) // one of the premisses don't hold
               {
                  break; // no need to look further
               }
            } // end checking each of the premisses

            cout << "done checking each of the premisses" << endl;
            if (prem) // the premisses hold
            {
               cout << "premisses hold" << endl;
               //cout << "size of modifier constraint map: "  <<
modifierConstraintMap[index[j]].size() << endl;
               for(int k=0; k < modifierConstraintMap[index[j]].size(); k++) // add the
additional constraints.
               {
                  //cout << "adding additional constraints, currently at position " << k
<< endl;
                  cout << "  adding constraint type: " <<
modifierConstraintMap[index[j]][k].getType() << ", " <<
modifierConstraintMap[index[j]][k].getValue() << endl;
                  res.push_back(modifierConstraintMap[index[j]][k]);
               }
               break; // no need to look further
            }
         } // end checking each of the modifier - constraint pair
      } // end if
    } // end if
  } // end checking each concept in the list
  if (res.size() == ressize)
  {
    cout << "constraints in the concept list not found..." << endl;
  }
  cout << "the following constraints were determined for " << reference.getType() << " ("
<< reference.getValue() << ") :" << endl;
  for (int i = 0; i < res.size(); i++)
  {
    cout << "   contraint: " << res[i].getType() << " (" << res[i].getValue() << ")" <<
endl;
  }
  cout << "end of constraints" << endl;
  return res;
}


void DSConstraintDetection::findSubValConstraints(DSConcept reference, string subValue,
vector<DSConstraint> &res, map<string, int> &foundTypes, string &conType)
{
   bool conLinkedToTypeAdded = false;

   if (valueConstraintIndex.count(subValue)!=0) // check whether the subValue exists in
the map.
   {
      if (hasValueTypeConstraint.count(subValue)!=0) // check whether there are
constraints linked to a type.
      {
         if
(valueTypeConstraintIndex[valueConstraintIndex[subValue]].count(reference.getType())!=0)
// is the type in the list?
         {
```

```
           for (int j=0; j <
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]].size(); j++)
           {
              conType =
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j].getType();
              if (foundTypes.count(conType)==0)
              {

res.push_back(valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]
][reference.getType()]][j]);// add the constraint.
                 cout << "  constraint type added: " << res[res.size()-1].getType() << ",
" << res[res.size()-1].getValue() << endl;
                 foundTypes[conType]=res.size()-1; //add constraint type to list of found
constraint types.
               }
              else if (res[foundTypes[conType]].getValue() !=
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j].getValue()) // values conflict
              {
                 if (res[foundTypes[conType]].getPriority() ==
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j].getPriority())
                 {
                    res[foundTypes[conType]].setValue("mixed"); // priorities are the
same, set as mixed.
                  }
                 else
                 {
                    if (res[foundTypes[conType]].getPriority() <
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j].getPriority())
                    {
                       res[foundTypes[conType]] =
valueTypeConstraint[valueTypeConstraintIndex[valueConstraintIndex[subValue]][reference.ge
tType()]][j]; // new priority is higher, so replace
                     }
                  }
               }// end else if (values conflict)
            }// end for
            cout << "constraints linked to type added" << endl;
            conLinkedToTypeAdded = true;
         }// end is the type in the list?
      } // end are there type related constraints?
      if (!conLinkedToTypeAdded)
      {
         cout << subValue << " has constraints to add, index = " <<
valueConstraintIndex[subValue] << endl;
         for (int j=0; j < valueConstraintMap[valueConstraintIndex[subValue]].size(); j++)
         {
            //cout << "adding constraint " << j << " of " <<
valueConstraintMap[valueConstraintIndex[subValue]].size() << endl;
            // maybe add rules on assigning constraints???
            conType = valueConstraintMap[valueConstraintIndex[subValue]][j].getType();
            if (foundTypes.count(conType)==0)
            {
               res.push_back(valueConstraintMap[valueConstraintIndex[subValue]][j]);// add
the constraint.
               cout << "constraint type added: " << res[res.size()-1].getType() << ", " <<
res[res.size()-1].getValue() << endl;
               foundTypes[conType]=res.size()-1; //add constraint type to list of found
constraint types.
            }
            else if (res[foundTypes[conType]].getValue() !=
valueConstraintMap[valueConstraintIndex[subValue]][j].getValue()) // values conflict
            {
               if (res[foundTypes[conType]].getPriority() ==
valueConstraintMap[valueConstraintIndex[subValue]][j].getPriority())
               {
```

```
                  res[foundTypes[conType]].setValue("mixed"); // priorities are the same,
set as mixed.
               }
               else
               {
                  if (res[foundTypes[conType]].getPriority() <
valueConstraintMap[valueConstraintIndex[subValue]][j].getPriority())
                  {
                     res[foundTypes[conType]] =
valueConstraintMap[valueConstraintIndex[subValue]][j];
                  }
               }
            }
         } // end for
      } // end if not type linked constraints added
   } // end if subvalue is in the map
}
```

Concept

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: concept.h
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: Datastructure for concept.
//               A concept has the following values:
//               - type
//               - value
//               - timestamp
//               - referent
//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSCONCEPT_H
#define DSCONCEPT_H

#include <string>
#include <vector>

/**
 * Data structure which represents a concept.
 * A concept has the following values:
 * - type
 * - value
 * - timestamp
 * - referent
 * - list of subconcepts
```

```
 * - list entries
 */

class DSConcept
{
  public:

   static const DSConcept null;

   /**
    * Constructor.
    */

   DSConcept ();


  /**
   * Constructor.
   * @param newType The type of the concept.
   * @param newValue The value of the concept.
   * @param newTimestamp The timestamp of the value.
   */

  DSConcept(string newType, string newValue, int newTimestamp);

  /**
   * Constructor.
   * @param newType The type of the concept.
   * @param newValue The value of the concept.
   * @param newTimestamp The timestamp of the concept.
   * @param newListConcepts The list of list entries.
   * @param newSubConcepts The list of sub-concepts.
   */

  DSConcept(string newType, string newValue, int newTimestamp, vector<DSConcept>
*newSubConcepts, vector<DSConcept> *newListConcepts);


  /**
   * Constructor.
   * @param newType The type of the concept.
   * @param newValue The value of the concept.
   * @param newTimestamp The timestamp of the concept.
   * @param newSubConcepts The list of sub-concepts.
   * @param newListConcepts The list of list entries.
   * @param newReferent The referent of the concept.
   */

  DSConcept(string newType, string newValue, int newTimestamp, vector<DSConcept>
*newSubConcepts, vector<DSConcept> *newListConcepts, DSConcept *newReferent);



  /**
    * Returns the type of the concept.
    * @return The type of the concept.
    */

   string getType();

   /**
    * Returns the value of the concept.
    * @return The value of the concept.
    */

   string getValue();

   /**
    * Returns the timestamp of the concept.
    * @return The timestamp of the concept.
    */
```

```
int getTimestamp();

/**
 * Returns the referent of the concept.
 * @return The referent of the concept.
 */

DSConcept *getReferent();

/**
 * Returns the list of sub-concepts.
 * @return The list of sub-concepts.
 */

vector<DSConcept> *getSubConcepts();

/**
 * Returns the list entries.
 * @return The list entries.
 */

vector<DSConcept> *getListEntries();

/**
 * Returns the input origin.
 * @return The input origin.
 */

string getInputOrigin();

/**
 * Returns the text.
 * @return The text.
 */

string getText();

/**
 * Returns the this concept value.
 * @return The concept.
 */

string getConcept();

/**
 * Returns the superconcept.
 * @return The superconcept.
 */

string getSuperConcept();

/**
 * Gets the function of the concept.
 * @return the function.
 */

string getFunction(string);


/**
 * Sets the type of the concept.
 * @param newType The new type of the concept.
 */

void setType(string newType);

/**
 * Sets the value of the concept.
 * @param newValue The new value of the concept.
 */
```

```
void setValue(string newValue);

/**
 * Sets the timestamp of the concept.
 * @param newTimestamp The new timestamp of the concept.
 */

void setTimestamp(int newTimestamp);

/**
 * Sets the referent of the concept.
 * @param newReferent The new referent of the concept.
 */

void setReferent(DSConcept *newReferent);

/**
 * Sets the sub-concepts.
 * @param newSubConcepts The new sub-concepts.
 */

void setSubConcepts(vector<DSConcept> *newSubConcepts);

/**
 * Sets the list entries.
 * @param newListEntries The new list entries.
 */

void setListEntries(vector<DSConcept> *newListEntries);

/**
 * Sets the input origin.
 * @param input The input origin.
 */

void setInputOrigin(string input);

/**
 * Sets the text.
 * @param input The text.
 */

void setText(string input);

/**
 * Sets the this concept value.
 * @param thisconcept The concept.
 */

void setConcept(string thisconcept);

/**
 * Sets the superconcept.
 * @param superconcept The superconcept.
 */

void setSuperConcept(string superconcept);

/**
 * Sets the function of the concept.
 * @param function The function.
 */

void setFunction(string function);

/**
 * == operator for DSConcept.
 */

bool operator==(DSConcept a);
```

```
  protected:
   // none

  private:
   string type;
   string value;
   string thisconcept;
   string superconcept;
   int timestamp;
   string inputOrigin;
   DSConcept *referent;
   vector<DSConcept> *subConcepts;
   vector<DSConcept> *listEntries;
   string text; // the input received for this concept in text form.
   string function; //information on subject, object, etc.
};

#endif // DSCONCEPT_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: concept.cc
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: Datastructure for concept.
//               A concept has the following values:
//               - type
//               - value
//               - timestamp
//               - referent
//               - subConcepts
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "conlist.h"
#include "concept.h"

/**
 * Constructor.
 */

DSConcept::DSConcept ()
{
   referent = NULL;
   subConcepts = NULL;
   listEntries = NULL;
   thisconcept = "";
   superconcept = "";
}
```

```
/**
 * Constructor.
 * @param newType The type of the concept.
 * @param newValue The value of the concept.
 * @param newTimestamp The timestamp of the value.
 */

DSConcept::DSConcept(string newType, string newValue, int newTimestamp)
{
   type = newType;
   value = newValue;
   timestamp = newTimestamp;
   referent = NULL;
   subConcepts = NULL;
   listEntries = NULL;
   thisconcept = "";
   superconcept = "";
}

/**
 * Constructor.
 * @param newType The type of the concept.
 * @param newValue The value of the concept.
 * @param newTimestamp The timestamp of the concept.
 * @param newSubConcepts The list of sub-concepts.
 * @param newListConcepts The list entries.
 */

DSConcept::DSConcept(string newType, string newValue, int newTimestamp, vector<DSConcept>
*newSubConcepts, vector<DSConcept> *newListConcepts)
{
   type = newType;
   value = newValue;
   timestamp = newTimestamp;
   referent = NULL;
   subConcepts = newSubConcepts;
   listEntries = newListConcepts;
   thisconcept = "";
   superconcept = "";
}

/**
 * Constructor.
 * @param newType The type of the concept.
 * @param newValue The value of the concept.
 * @param newTimestamp The timestamp of the concept.
 * @param newSubConcepts The list of sub-concepts.
 * @param newListConcepts The list entries.
 * @param newReferent The referent of the concept.
 */

DSConcept::DSConcept(string newType, string newValue, int newTimestamp, vector<DSConcept>
*newSubConcepts, vector<DSConcept> *newListConcepts, DSConcept *newReferent)
{
   type = newType;
   value = newValue;
   timestamp = newTimestamp;
   subConcepts = newSubConcepts;
   listEntries = newListConcepts;
   referent = newReferent;
   thisconcept = "";
   superconcept = "";
}


/**
 * Returns the type of the concept.
 * @return The type of the concept.
 */
```

```
   string DSConcept::getType()
   {
     return type;
   }

/**
 * Returns the value of the concept.
 * @return The value of the concept.
 */

   string DSConcept::getValue()
   {
     return value;
   }

/**
 * Returns the timestamp of the concept.
 * @return The timestamp of the concept.
 */

   int DSConcept::getTimestamp()
   {
     return timestamp;
   }

/**
 * Returns the referent of the concept.
 * @return The referent of the concept.
 */

   DSConcept *DSConcept::getReferent()
   {
     return referent;
   }

   /**
    * Returns the input origin.
    * @return The input origin.
    */

string DSConcept::getInputOrigin()
{
  return inputOrigin;
}

   /**
    * Returns the text.
    * @return The text.
    */

string DSConcept::getText()
{
  return text;
}


   /**
    * Returns the this concept value.
    * @return The concept.
    */

string DSConcept::getConcept()
{
  return thisconcept;
}

   /**
    * Returns the superconcept.
    * @return The superconcept.
    */
```

```
string DSConcept::getSuperConcept()
{
  return superconcept;
}

/**
 * Returns the sub-concepts.
 * @return The sub-concepts.
 */

   vector<DSConcept> *DSConcept::getSubConcepts()
   {
     return subConcepts;
   }

/**
 * Returns the list-entries.
 * @return The list-entries.
 */

   vector<DSConcept> *DSConcept::getListEntries()
   {
     return listEntries;
   }

   /**
    * Gets the function of the concept.
    * @return the function.
    */

   string DSConcept::getFunction(string)
   {
     return function;
   }

/**
 * Sets the type of the concept.
 * @param newType The new type of the concept.
 */

   void DSConcept::setType(string newType)
   {
     type = newType;
   }

/**
 * Sets the value of the concept.
 * @param newValue The new value of the concept.
 */

   void DSConcept::setValue(string newValue)
   {
     value = newValue;
   }

/**
 * Sets the timestamp of the concept.
 * @param newTimestamp The new timestamp of the concept.
 */

   void DSConcept::setTimestamp(int newTimestamp)
   {
     timestamp = newTimestamp;
   }

/**
 * Sets the referent of the concept.
 * @param newReferent The new referent of the concept.
 */

   void DSConcept::setReferent(DSConcept *newReferent)
```

```
   {
     referent = newReferent;
   }

/**
 * Sets the new sub-concepts.
 * @param newSubConcepts The new sub-concepts.
 */

   void DSConcept::setSubConcepts(vector<DSConcept> *newSubConcepts)
   {
     subConcepts = newSubConcepts;
   }

/**
 * Sets the new list entries.
 * @param newListEntries The new list entries.
 */

   void DSConcept::setListEntries(vector<DSConcept> *newListEntries)
   {
     listEntries = newListEntries;
   }

   /**
    * Sets the input origin.
    * @param input The input origin.
    */

void DSConcept::setInputOrigin(string input)
{
  inputOrigin = input;
}

   /**
    * Sets the text.
    * @param input The text.
    */

void DSConcept::setText(string input)
{
  text = input;
}

   /**
    * Sets the this concept value.
    * @param input The concept.
    */

void DSConcept::setConcept(string thisconcept)
{
  this->thisconcept = thisconcept;
}

   /**
    * Sets the superconcept.
    * @param input The superconcept.
    */

void DSConcept::setSuperConcept(string superconcept)
{
  this->superconcept = superconcept;
}

void DSConcept::setFunction(string function)
{
  this->function = function;
}

   /**
    * == operator for DSConcept.
```

```
    */


/**
 * == operator for DSConcept.
 */

bool DSConcept::operator==(DSConcept a)
{
  return (a.value == value)&&(a.type == type);//&&(a.timestamp == timestamp);
}

//  protected:
   // none

//  private:
  // none
```

Constraint

Header file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: constr.h
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: Datastructure for constraint.
//               A concept has the following values:

//
////////////////////////////////////////////////////////////////////////////////
////////

#ifndef DSCONSTRAINT_H
#define DSCONSTRAINT_H

#include <string>

/**
 * Data structure which represents a constraint.
 * A constraint has the following values:
 * - type
 * - value
 */

class DSConstraint
{
  public:
   static const DSConstraint null;
```

```
/**
 * Constructor.
 */

  DSConstraint();

/**
 * Constructor.
 * @param newType The type of the constraint.
 * @param newValue The value of the constraint.
 */
   DSConstraint(string newType, string newValue);

/**
 * Constructor.
 * @param newType The type of the constraint.
 * @param newValue The value of the constraint.
 * @param newPriority The value of the priority.
 */
   DSConstraint(string newType, string newValue, int newPriority);

   /**
    * Returns the type of the constraint.
    * @return The type of the constraint.
    */

   string getType();

   /**
    * Returns the value of the constraint.
    * @return The value of the constraint.
    */

   string getValue();

   /**
    * Returns the priority of the constraint.
    * @return The priority of the constraint.
    */

   int getPriority();

   /**
    * Sets the type of the constraint.
    * @param newType The new type of the constraint.
    */

   void setType(string newType);

   /**
    * Sets the value of the constraint.
    * @param newValue The new value of the constraint.
    */

   void setValue(string newValue);

     /**
    * Sets the priority of the constraint.
    * @param newPriority The new priority of the constraint.
    */

   void setPriority(int newPriority);

      protected:
   // none

  private:
   string type;
   string value;
   int priority;
};
```

```
#endif // DSCONSTRAINT_H
```

Implementation file

```
////////////////////////////////////////////////////////////////////////////////
////////
//
//
//               Copyright (C) 2001 Philips GmbH Dialog Systems
//
//               All rights reserved
//
//
//
////////////////////////////////////////////////////////////////////////////////
////////
//
//  File: constr.cc
//
//  Last changed by:
//  Last changed on:
//
//  Created by: Dimitri Woei-A-Jin
//  Created on: January 17, 2001
//
//  Description: Datastructure for constraint.
//               A constraint has the following values:
//               -type
//               -value
//
////////////////////////////////////////////////////////////////////////////////
////////

#include "constr.h"

//  public:



/**
 * Constructor.
 */

DSConstraint::DSConstraint()
{
  // none
}
/**
 * Constructor.
 * @param newType The type of the constraint.
 * @param newValue The value of the constraint.
 */

DSConstraint::DSConstraint(string newType, string newValue)
{
     type = newType;
     value = newValue;
     priority = 1; //(default)
}

/**
 * Constructor.
 * @param newType The type of the constraint.
 * @param newValue The value of the constraint.
 * @param newPriority The value of the priority.
 */
```

```
DSConstraint::DSConstraint(string newType, string newValue, int newPriority)
{
     type = newType;
     value = newValue;
     priority = newPriority;
}

   /**
    * Returns the type of the constraint.
    * @return The type of the constraint.
    */

string DSConstraint::getType()
{
     return type;
}

   /**
    * Returns the value of the constraint.
    * @return The value of the constraint.
    */

string DSConstraint::getValue()
{
     return value;
}

   /**
    * Returns the value of the priority.
    * @return The value of the priority.
    */

int DSConstraint::getPriority()
{
     return priority;
}

   /**
    * Sets the type of the constraint.
    * @param newType The new type of the constraint.
    */

void DSConstraint::setType(string newType)
{
     type = newType;
}

   /**
    * Sets the value of the constraint.
    * @param newValue The new value of the constraint.
    */

void DSConstraint::setValue(string newValue)
{
     value = newValue;
}

   /**
    * Sets the value of the constraint.
    * @param newValue The new value of the constraint.
    */

void DSConstraint::setPriority(int newPriority)
{
     priority = newPriority;
}

//  protected:
   // none

//  private:
```

```
// none
```

Appendix F

Usability test tasks

T11 You have no appointments this evening and you decide to sit back, relax and watch some sports on TV. Search for a sports program that is running tonight around 8 PM and put in on your watch list.

T12 You just got home and wonder what is on CNN right now, so you switch to that channel.

T12 This whole week you have been busy and never got around to read the newspaper or watch the news. And tonight you have another appointment. You find the lat night news compact and up to date, so you decide to record it.

T21 Someone recommended a program about Science Frontiers to you. It airs on Discovery Channel around 8 PM on Thursday. You decide to record it.

T22 You want to watch a late night movie tonight. You decide to put the one with Robert de Niro on your watch list.

T23 Ross Kemp is an actor you enjoy watching. On Wednesday around 6 PM there is a serie in which he stars. You decide to record that serie.

T31 Your appointment for tomorrow evening is cancelled, so you would like to know if there is anything interesting on TV tomorrow night. You search for an entertainment show on Channel 5 and put it on your watch list.

T32 Your brother is curious what kind of movies Sky movie Premiere broadcasts at prime time. Rather than lending him your programming guide, you decide to record a movie for him on Friday that starts around 8 PM so he can see for himself.

T33 The other day, your best friend gave you some healthy advice: try to limit the time you watch TV and exercise some more. You decide to remove one program from your watchlist for starters.

Appendix G

Test Results

Offline tests

The following examples were tested offline with the following results:

- *Simulation program list 1 is 'displayed on the screen'.*

SPICE, are there any movies starring Mel Gibson today?

- *Simulation program list 2 is 'displayed on the screen'.*

Can you show me more information about this movie?

- *Besides the parsed sentence, an extra concept* deicticmovie1 *is added to the input, with input origin value* deixis. **This movie** *is correctly resolved to* **deicticmovie1**.
- *Simulation program information is 'displayed on the screen'.*

Could you show me the list again?

- **The list** *is correctly resolved to* **program list 2.**

Please record the Mad Max movies.

- *The definite description* **the Mad Max movies** *was resolved to* **the group of Mad Max movies in program list 2**. *This was possible because the movies had the subconcept Mad Max.*
- *Simulation program list 4 is 'displayed on the screen'.*

Please record the Mad Max movie.

- *The definite description* **the Mad Max movie** *is correctly resolved to* **the top most movie Mad Max**. *This is because a movie in the list had the subconcept* protagonist: Mad Max, *which was added specifically to test this situation.*

Are there any samurai movies today?

- *Simulation program list 6 is 'displayed on the screen'.*

Who is the director of this one?

- *Besides the parsed sentence, an extra concept* deicticmovie2 *is added to the input, with input origin value* deixis. **This one** *is correctly resolved to* **deicticmovie2**. *The director is resolved to* **Akira Kurasowa**, *which was added specifically as a subconcept of* deicticmovie2 *to test this case. The structure superconcept – concept made it possible to solve this situation.*

Are there any other movies directed by him this month?

- **Him** *is correctly resolved to* **Akira Kurasowa**.

Are there any movies by Roman Polansky?

- *Simulation program list 9 is 'displayed on the screen'.*

In which of these does he star himself?

He and **himself** are correctly resolved to **Roman Polansky.**

- *Simulation program list 10 is 'displayed on the screen'.*

Please record the most recent one.

- *Since no information about release dates of movies are present, this reference could not be resolved.*
- *Simulation program list 11 is 'displayed on the screen'.*

Gimme info on the fourth movie.

- **The fourth movie** *is correctly resolved to* **the fourth movie in program list 11**.
- *Simulation program list 12 is 'displayed on the screen'.*

Can I see the last one?

- **The last one** *is correctly resolved to* **the last program in program list 12**.

Gimme info on that movie.

- **That movie** *is correctly resolved to* **the fourth movie.**
- *Simulation program list 13 is 'displayed on the screen'.*

I want to see a James Bond movie.

- *Simulation program list 007 is 'displayed on the screen'.*

Do you have other movies with him?

- **Him** *correctly resolved to* **James Bond**.
- *Simulation program list 15 is 'displayed on the screen'.*

Do you have more information about the last thing?

- **The last thing** *correctly resolved to* **the last program in program list 15.**
- *Simulation program list 16 is 'displayed on the screen'.*

What is that about?

- **That** *correctly resolved to* **the referent of 'the last thing'**.
- *Simulation program list 17 is 'displayed on the screen'.*

What time does it start?

- **It** *correctly resolved to* **the referent of 'that'**.
- *Simulation program list 18 is 'displayed on the screen'.*

Are there any movies with X next week?

Simulation program list 19 is 'displayed on the screen'.

Which of them is together with Y?

- **Them** *correctly resolved to* **the movies in the list as a group**.
- *Simulation program list 20 is 'displayed on the screen'.*

O.k. so please record the first one!

- **The first one** *correctly resolved to* **the first program in program list 20.**
- *Simulation program list 21 is 'displayed on the screen'.*

I am looking for a movie with Kate Winslet where she plays an Australian girl.

- **She** *correctly resolved to* **Kate Winslet**.
- *Simulation program list 22 is 'displayed on the screen'.*

From the last list of movies, the second one.

- **The last list of movies** *correctly resolved to* **program list 21**.
- **The second one** *correctly resolved to* **the second movie of program list 21**.
- *Simulation program list 23 is 'displayed on the screen'.*

I said: a movie with Robert Redford! He does not act in these ones.

- **He** *correctly resolved to* **Robert redford.**
- **These ones** *correctly resolved to* **the group of movies in program list 23.**
- *Simulation program list 24 is 'displayed on the screen'.*

The serial I saw last night, when will it be continued?

- **The serial I saw last night** *resolved to* **NULL**, *because no such entry exists in the history list.*
- **It** *resolved to* **NULL**, *because since 'the serial I saw last night' did not have a referent, it was tagged incorrectly and was not put in the s-list. It was assumed that a reference would always have a referent, so only a check is made for referents, and no attempt is made to try to determine the newness for references.*
- *Simulation program list 25 is 'displayed on the screen'.*

Is this a science fiction movie?

- *Besides the parsed sentence, an extra concept* deicticmovie3 *is added to the input, with input origin value* deixis.**This** *correctly resolved to* **deicticmovie3.**
- *Simulation program list 26 is 'displayed on the screen'.*

There is a movie with Billy Crystal and Meg Ryan where they play two singles in New York.

- **They** *correctly resolved to* **Billy Crystal and Meg Ryan**.
- *Simulation program list 27 is 'displayed on the screen'.*

Are there any other movies with him or her?

- **Him** *correctly resolved to* **Billy Crystal.**
- **Her** *correctly resolved to* **Meg Ryan.**
- *Simulation program list 28 is 'displayed on the screen'.*

I am looking for a movie. It should start around 8pm.

- **It** *correctly resolved to* **a movie**.
- *Simulation program list 29 is 'displayed on the screen'.*

Give more information on the last one.

- **The last one** *correctly resolved to* **the last program in program list 29**.
- *Simulation program list 30 is 'displayed on the screen'.*

I want to remove that one.

- **That one** *correctly resolved to* **the referent of 'the last one'.**
- *Simulation program list 31 is 'displayed on the screen'.*

Put it on my recording list.

- **It** *correctly resolved to* **the referent of 'that one'.**
- *Simulation program list 32 is 'displayed on the screen'.*

Remove the earlier one.

- **The earlier one** *correctly resolved to* **the program with the lowest start time in program list 32**.
- *Simulation program list 33 is 'displayed on the screen'. One of the movies has the subconcept actor = Julia Roberts.*

Show me the one that has Julia Roberts in it.

- **The one that has Julia Roberts in it** *correctly resolved to* **the movie with subconcept 'actor = Julia Roberts' in program list 33.**
- **It** *correctly resolved to* **the one that has Julia Roberts in it.**
- *Simulation program list 33 is 'displayed on the screen'.*

Online tests

During online testing, a set of tasks described in appendix F was used to test the references. In the following list is described what was said, what was understood during the different attempts, and what was resolved :

what's on cnn right now
- *what time cnn right now*
- *what's on*
- *on cnn right now*

switch to that channel
- *richard a program*
- *this the a channel* → **this** *resolved to* **cnn**
- *switch to a channel*
- *switch to*
- *its that a channel* → **its** *resolved to* **cnn**
- *whitch that channel* → **that channel** *correctly resolved to* **cnn**
- *switch to that channel* → **that channel** *correctly resolved to* **cnn**

#silence#
- *ok*

reset
- *that*→ **that** *resolved to* **NULL**
- *eight that*→ **that** *resolved to* **NULL**

are there any sports tonight
- *are there any sports*
- *are there any sports*
- *(there) any sports tonight*
- *are there any sports tonight*

record the second program
- *record this* → **this** *resolved to* **NULL**
- *record the second programs* → **the second programs** *correctly resolved to* **football**
- *record the second program* → **the second program** *correctly resolved to* **football**

what news is on tonight
- *what muses on tonight*
- *what news the*
- *what's uses on tonight*
- *what news is on*
- *what news is on tonight*

record the ten o'clock news
- *record it that o' clock news* → **it** *was resolved to* **NULL** *and* **that** *was resolved to* **it**.
- *record it them o'clock news* → **it** *was resolved to* **NULL** *and* **them** *was resolved to the group of programs:* **programs 6**
- *record at ten o'clock news*

record the eight p.m. news
- *record at eight p.m. news*
- *record the eight p.m news* → **the eight p.m. news** *correctly resolved to* **world news.**

record the last news program

- *record the last shows program* → **the last shows program** *resolved to* **NULL**, *because no shows are on the list.*
- *record the last news program* → **the last news program** *correctly resolved to* **cnn news**. *This one was not the last on the displayed list, but the last on the list of programs.*

are there any programs on science frontiers on thursday

- *are there any programs on science frontiers today*

science frontiers on thursday

- *science frontiers on thursday*

record it

- *record it* → **it** *correctly resolved to* **avalanche** *(a program about science frontiers)*

are there any movies with robert de niro tonight

- *are there any movies this from the new tonight* → **this** *resolved to* **avalanche**
- *are there any movies with the real*
- *a movies*
- *...*
- *After many retries this query was still not recognized, so finally it was just left untested.*

are there any entertainment tomorrow evening

- *any entertainment tomorrow evening*

record the one on channel 5

- *record the one on channel 5* → **the one on channel 5** *correctly resolved to* **it's only tv but I like it**

any movies on friday

- *any movies on friday*

record the second one from below

- *record the second one from below* → **the second one from below** *correctly resolved to* **intimate relations 1995**.

show me the watch list

- *show me the watch list (the watch list not resolved, because it is treated as a name)*

remove the earlier program

- *remove the earlier program* → **the earlier program** *correctly resolved to* **100 per cent**

Appendix H

Constraints

```
# This file is used for constraint detection
# Priority value indicates whether a constraint value of a certain type overwrites
another conflicting constraint value (if it is higher) or whether the constraint value
should become mixed.
# format:
#
# concept value1:
# constraint type1, constraint value1, priority value
# >concept type1
# )concept type1 constraint type1, concept type1 constraint value1, concept type1
priority value1
# ...
# )concept type1 constraint typen, concept typen constraint valuen, concept typen
priority valuen
# >;
# ...
# constraint typen, constraint valuen
# ;
# ...
#
# concept value:
# ...
# ;
#
# note: 'listentry', 'recency' and 'relativetimeposition' are special constraints. These
are reserved.
#

dummy:
number, dummy, 10
person, dummy, 10
type, dummy, 10
abstract, dummy, 10
;

director:
number, singular, 1
person, person, 1
type, director, 1
abstract, no, 1
;
directors:
number, plural, 1
person, person, 1
type, director, 1
abstract, no, 1
;

star:
number, singular, 1
person, person, 1
type, actor, 1
abstract, no, 1
;
stars:
number, plural, 1
person, person, 1
type, actor, 1
abstract, no, 1
;

moviestar:
number, singular, 1
person, person, 1
type, actor, 1
abstract, no, 1
;
moviestars:
number, plural, 1
person, person, 1
type, actor, 1
abstract, no, 1
;
actor:
number, singular, 1
person, person, 1
gender, male, 1
type, actor, 1
abstract, no, 1
;
actors:
number, plural, 1
person, person, 1
# gender, male, 1
type, actor, 1
abstract, no, 1
;
actress:
number, singular, 1
person, person, 1
gender, female, 1
type, actor, 1
abstract, no
```

```
;
actresses:
number, plural, 1
person, person, 1
gender, female, 1
type, actor, 1
abstract, no, 1
;
guy:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
guys:
number, plural, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
man:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
men:
number, plural, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
mister:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
misters:
number, plural, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
boy:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
boys:
number, plural, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
gentleman:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
gentlemen:
number, plural, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
hunk:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
hunks:
number, plural, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
lad:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
lads:
number, plural, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
gall:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
galls:
number, plural, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
woman:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
women:
number, plural, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
lady:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
ladies:
number, plural, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
girl:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
girls:
number, plural, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
babe:
number, singular, 1
```

```
person, person, 1
gender, female, 1
abstract, no, 1
;
babes:
number, plural, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
chick:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
chicks:
number, plural, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
lass:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
lasses:
number, plural, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
person:
number, singular, 1
person, person, 1
abstract, no, 1
;
persons:
number, plural, 1
person, person, 1
abstract, no, 1
;
He:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
he:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
Him:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
him:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
Himself:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
himself:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
His:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;
his:
number, singular, 1
person, person, 1
gender, male, 1
abstract, no, 1
;

She:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
she:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
Her:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
her:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
Herself:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
herself:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
Hers:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
hers:
number, singular, 1
person, person, 1
gender, female, 1
abstract, no, 1
;
```

```
It:
number, singular, 1
person, nonperson, 1
abstract, no, 1
;
it:
number, singular, 1
person, nonperson, 1
abstract, no, 1
;
Its:
number, singular, 1
person, nonperson, 1
abstract, no, 1
;
its:
number, singular, 1
person, nonperson, 1
abstract, no, 1
;
Itself:
number, singular, 1
person, nonperson, 1
abstract, no, 1
;
itself:
number, singular, 1
person, nonperson, 1
abstract, no, 1
;

Them:
number, plural, 1
abstract, no, 1
;
them:
number, plural, 1
abstract, no, 1
;
Themselves:
number, plural, 1
abstract, no, 1
;
themselves:
number, plural, 1
abstract, no, 1
;
They:
number, plural, 1
abstract, no, 1
;
they:
number, plural, 1
abstract, no, 1
;
Their:
number, plural, 1
abstract, no, 1
;
their:
number, plural, 1
abstract, no, 1
;
Theirs:
number, plural, 1
abstract, no, 1
;
theirs:
number, plural, 1
abstract, no, 1
;

This:
number, singular, 2
#inputOrigin, deixis, 1
abstract, no, 1
;
this:
number, singular, 2
#inputOrigin, deixis, 1
abstract, no, 1
;
That:
number, singular, 2
#inputOrigin, deixis, 1
abstract, no, 1
;
that:
number, singular, 2
#inputOrigin, deixis, 1
abstract, no, 1
;

These:
number, plural, 1
#inputOrigin, deixis, 1
abstract, no, 1
;
these:
number, plural, 2
#inputOrigin, deixis, 1
abstract, no, 1
;
Those:
number, plural, 2
#inputOrigin, deixis, 1
abstract, no, 1
;
those:
number, plural, 2
#inputOrigin, deixis, 1
abstract, no, 1
;

list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
;
schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
;
selection:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
;
programme:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
;
watch list:
number, singular, 1
person, nonperson, 1
type, list, 1
```

```
list, watch, 1
abstract, no, 1
;
show list:
number, singular, 1
person, nonperson, 1
type, list, 1
list, watch, 1
abstract, no, 1
;
watch schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
list, watch, 1
abstract, no, 1
;
show schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
list, watch, 1
abstract, no, 1
;
record list:
number, singular, 1
person, nonperson, 1
type, list, 1
list, record, 1
abstract, no, 1
;
recording list:
number, singular, 1
person, nonperson, 1
type, list, 1
list, record, 1
abstract, no, 1
;
vcr list:
number, singular, 1
person, nonperson, 1
type, list, 1
list, record, 1
abstract, no, 1
;
record schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
list, record, 1
abstract, no, 1
;
recording schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
list, record, 1
abstract, no, 1
;
vcr schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
list, record, 1
abstract, no, 1
;

recording:
list, record, 1
;
```

```
info:
type, info, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
information:
type, info, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
infos:
type, info, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
details:
type, info, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
explanation:
type, info, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
description:
type, info, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;

mel gibson:
number, singular, 1
person, person, 1
type, actor, 1
gender, male, 1
abstract, no, 1
;

akira kurasowa:
number, singular, 1
person, person, 1
type, director, 1
gender, male, 1
abstract, no, 1
;

kate winslet:
number, singular, 1
person, person, 1
type, actor, 1
gender, female, 1
abstract, no, 1
;

billy crystal:
number, singular, 1
person, person, 1
type, actor, 1
gender, male, 1
abstract, no, 1
;

robert redford:
number, singular, 1
```

```
person, person, 1
type, actor, 1
gender, male, 1
abstract, no, 1
;

julia roberts:
>actor
)number, singular, 1
)person, person, 1
)type, actor, 1
)gender, female, 1
)abstract, no, 1
>;
number, singular, 1
person, person, 1
type, actor, 1
gender, female, 1
abstract, no, 1
;

meg ryan:
number, singular, 1
person, person, 1
type, actor, 1
gender, female, 1
abstract, no, 1
;

sandra bullock:
number, singular, 1
person, person, 1
type, actor, 1
gender, female, 1
abstract, no, 1
;

richard crenna:
number, singular, 1
person, person, 1
type, actor, 1
gender, male, 1
abstract, no, 1
;

robert de niro:
number, singular, 1
person, person, 1
type, actor, 1
gender, male, 1
abstract, no, 1
;

ross kemp:
number, singular, 1
person, person, 1
type, actor, 1
gender, male, 1
abstract, no, 1
;

paul bown:
number, singular, 1
person, person, 1
type, actor, 1
gender, male, 1
abstract, no, 1
;

humphrey bogart:
number, singular, 1
person, person, 1
type, actor, 1
gender, male, 1
abstract, no, 1
;

paul newman:
number, singular, 1
person, person, 1
type, actor, 1
gender, male, 1
abstract, no, 1
;

james bond:
number, singular, 1
person, person, 1
type, protagonist, 1
gender, male, 1
abstract, no, 1
;

new york yankees:
number, plural, 1
type, team, 1
abstract, no, 1
game, baseball, 1
;
boston red sox:
number, plural, 1
type, team, 1
abstract, no, 1
game, baseball, 1
;
chicago white sox:
number, plural, 1
type, team, 1
abstract, no, 1
game, baseball, 1
;
boston bruins:
number, plural, 1
type, team, 1
abstract, no, 1
game, hockey, 1
;
chicago blackhawks:
number, plural, 1
type, team, 1
abstract, no, 1
game, hockey, 1
;
los angeles dodgers:
number, plural, 1
type, team, 1
abstract, no, 1
game, baseball, 1
;
arizona diamondbacks:
number, plural, 1
type, team, 1
abstract, no, 1
game, baseball, 1
;
san fransisco giants:
number, plural, 1
type, team, 1
abstract, no, 1
game, baseball, 1
;
```

```
and:
number, plural, 2
;
,:
number, plural, 2


first:
listentry, 1, 1
number, singular, 2
;
second:
listentry, 2, 1
number, singular, 2
;
third:
listentry, 3, 1
number, singular, 2
;
fourth:
listentry, 4, 1
number, singular, 2
;
fifth:
listentry, 5, 1
number, singular, 2
;
sixth:
listentry, 6, 1
number, singular, 2
;
seventh:
listentry, 7, 1
number, singular, 2
;
eighth:
listentry, 8, 1
number, singular, 2
;
last:
listentry, -1, 1
number, singular, 2
;

the first from below:
listentry, -1, 1
number, singular, 1
;
the second from below:
listentry, -2, 1
number, singular, 1
;
the third from below:
listentry, -3, 1
number, singular, 1
;
the fourth from below:
listentry, -4, 1
number, singular, 1
;
the fifth from below:
listentry, -5, 1
number, singular, 1
;
the sixth from below:
listentry, -6, 1
number, singular, 1
;
the seventh from below:
listentry, -7, 1
number, singular, 1
;
the eighth from below:
listentry, -8, 1
number, singular, 1
;

the first one from below:
listentry, -1, 1
number, singular, 1
;
the second one from below:
listentry, -2, 1
number, singular, 1
;
the third one from below:
listentry, -3, 1
number, singular, 1
;
the fourth one from below:
listentry, -4, 1
number, singular, 1
;
fifth one from below:
listentry, -5, 1
number, singular, 1
;
the sixth one from below:
listentry, -6, 1
number, singular, 1
;
the seventh one from below:
listentry, -7, 1
number, singular, 1
;
the eighth one from below:
listentry, -8, 1
number, singular, 1
;

the first entry from below:
listentry, -1, 1
number, singular, 1
;
the second entry from below:
listentry, -2, 1
number, singular, 1
;
the third entry from below:
listentry, -3, 1
number, singular, 1
;
the fourth entry from below:
listentry, -4, 1
number, singular, 1
;
the fifth entry from below:
listentry, -5, 1
number, singular, 1
;
the sixth entry from below:
listentry, -6, 1
number, singular, 1
;
the seventh entry from below:
listentry, -7, 1
number, singular, 1
;
the eighth entry from below:
listentry, -8, 1
number, singular, 1
;
```

```
the first program from below:
listentry, -1, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no
;
the second program from below:
listentry, -2, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the third program from below:
listentry, -3, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the fourth program from below:
listentry, -4, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the fifth program from below:
listentry, -5, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the sixth program from below:
listentry, -6, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the seventh program from below:
listentry, -7, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the eighth program from below:
listentry, -8, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;

the first programme from below:
listentry, -1, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the second programme from below:
listentry, -2, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the third programme from below:
listentry, -3, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the fourth programme from below:
listentry, -4, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the fifth programme from below:
listentry, -5, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the sixth programme from below:
listentry, -6, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the seventh programme from below:
listentry, -7, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the eighth programme from below:
listentry, -8, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;

the first title from below:
listentry, -1, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the second title from below:
listentry, -2, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the third title from below:
listentry, -3, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the fourth title from below:
listentry, -4, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
```

```
;
the fifth title from below:
listentry, -5, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the sixth title from below:
listentry, -6, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the seventh title from below:
listentry, -7, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the eighth title from below:
listentry, -8, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;

the first thing from below:
listentry, -1, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the second thing from below:
listentry, -2, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the third thing from below:
listentry, -3, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the fourth thing from below:
listentry, -4, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the fifth thing from below:
listentry, -5, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the sixth thing from below:
listentry, -6, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the seventh thing from below:
listentry, -7, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
the eighth thing from below:
listentry, -8, 1
number, singular, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;

the first movie from below:
listentry, -1, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the second movie from below:
listentry, -2, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the third movie from below:
listentry, -3, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the fourth movie from below:
listentry, -4, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the fifth movie from below:
listentry, -5, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the sixth movie from below:
listentry, -6, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the seventh movie from below:
listentry, -7, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
```

```
;
the eighth movie from below:
listentry, -8, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;

the first film from below:
listentry, -1, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the second film from below:
listentry, -2, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the third film from below:
listentry, -3, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the fourth film from below:
listentry, -4, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the fifth film from below:
listentry, -5, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the sixth film from below:
listentry, -6, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the seventh film from below:
listentry, -7, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;
the eighth film from below:
listentry, -8, 1
number, singular, 1
type, programme, 1
category, movie, 1
person, nonperson, 1
abstract, no, 1
;

the first emmission from below:
listentry, -1, 1
number, singular, 1
type, programme, 1
category, emmission, 1
person, nonperson, 1
abstract, no, 1
;
the second emmission from below:
listentry, -2, 1
number, singular, 1
type, programme, 1
category, emmission, 1
person, nonperson, 1
abstract, no, 1
;
the third emmission from below:
listentry, -3, 1
number, singular, 1
type, programme, 1
category, emmission, 1
person, nonperson, 1
abstract, no, 1
;
the fourth emmission from below:
listentry, -4, 1
number, singular, 1
type, programme, 1
category, emmission, 1
person, nonperson, 1
abstract, no, 1
;
the fifth emmission from below:
listentry, -5, 1
number, singular, 1
type, programme, 1
category, emmission, 1
person, nonperson, 1
abstract, no, 1
;
the sixth emmission from below:
listentry, -6, 1
number, singular, 1
type, programme, 1
category, emmission, 1
person, nonperson, 1
abstract, no, 1
;
the seventh emmission from below:
listentry, -7, 1
number, singular, 1
type, programme, 1
category, emmission, 1
person, nonperson, 1
abstract, no, 1
;
the eighth emmission from below:
listentry, -8, 1
number, singular, 1
type, programme, 1
category, emmission, 1
person, nonperson, 1
abstract, no, 1
;

the first serie from below:
listentry, -1, 1
```

```
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the second serie from below:
listentry, -2, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the third serie from below:
listentry, -3, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the fourth serie from below:
listentry, -4, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the fifth serie from below:
listentry, -5, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the sixth serie from below:
listentry, -6, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the seventh serie from below:
listentry, -7, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the eighth serie from below:
listentry, -8, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;

the first serial from below:
listentry, -1, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the second serial from below:
listentry, -2, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the third serial from below:
listentry, -3, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the fourth serial from below:
listentry, -4, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the fifth serial from below:
listentry, -5, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the sixth serial from below:
listentry, -6, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the seventh serial from below:
listentry, -7, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;
the eighth serial from below:
listentry, -8, 1
number, singular, 1
type, programme, 1
category, serie, 1
person, nonperson, 1
abstract, no, 1
;

one:
number, singular, 1
abstract, no, 1
;
ones:
number, plural, 1
abstract, no, 1
;

movie:
>programme
)number, singular, 1
)category, movie, 1
)type, programme, 2
```

```
)person, nonperson, 1
)abstract, no, 1
>;
number, singular, 1
category, movie, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
movies:
number, plural, 1
category, movie, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
emmission:
number, singular, 1
category, emmission, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
emmissions:
number, plural, 1
category, emmission, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
entertainment:
number, singular, 1
category, emmission, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
film:
number, singular, 1
category, movie, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
films:
number, plural, 1
category, movie, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
serie:
number, singular, 1
category, serie, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
series:
number, plural, 1
category, series, 1
type, programme, 1
person, nonperson, 1
abstract, no, 1
;
serial:
number, singular, 1
category, serie, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
serials:
number, plural, 1
category, series, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
sport:
number, singular, 1
category, sport, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
sports:
number, plural, 1
category, sports, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
;
kids:
category, kids, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
number, sigular, 1
;
children:
category, kids, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
number, sigular, 1
;
children's:
category, kids, 1
type, programme, 2
person, nonperson, 1
abstract, no, 1
number, sigular, 1
;
news:
category, news, 2
type, programme, 1
person, nonperson, 1
abstract, no, 1
number, singular, 1
;


title:
type, programme, 2
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
titles:
type, programme, 2
person, nonperson, 1
number, plural, 1
abstract, no, 1
;
programme:
type, programme, 2
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
programmes:
```

```
type, programme, 2
person, nonperson, 1
number, plural, 2
abstract, no, 1
;
program:
type, programme, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
programs:
type, programme, 1
person, nonperson, 1
number, plural, 1
abstract, no, 1
;

channel:
type, channel, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
channels:
type, channel, 1
person, nonperson, 1
number, plural, 1
abstract, no, 1
;
station:
type, channel, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
stations:
type, channel, 1
person, nonperson, 1
number, plural, 1
abstract, no, 1
;
net:
type, channel, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
nets:
type, channel, 1
person, nonperson, 1
number, plural, 1
abstract, no, 1
;

bbc:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, bbc1, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
bbc1:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, bbc1, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
bbc2:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, bbc2, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
bbc prime:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, bbc prime, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
bbc world:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, bbc world, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
cnn:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, cnn, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
channel 4:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, channel 4, 1
```

```
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
channel 5:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, channel 5, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
discovery:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, discorvery channel, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
discovery channel:
type, channel, 1
channel, discorvery channel, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
mtv:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, mtv, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
sky cinema:
type, channel, 1
channel, sky cinema, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
sky:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, sky cinema, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
sky movie premiere:
type, channel, 1
channel, sky movie premiere, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
sky movie:
type, channel, 1
channel, sky movie premiere, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;

BBC1:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, bbc1, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
BBC2:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, bbc2, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
BBC_PRIME:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, bbc prime, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
BBC_WORLD:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, bbc world, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
CNN:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
```

```
)abstract, no, 1
>;
type, channel, 1
channel, cnn, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
CHANNEL_4:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, channel 4, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
CHANNEL_5:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, channel 5, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
DISCOVERY:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, discorvery channel, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
DISCOVERY_CHANNEL:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, discorvery channel, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
MTV:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, mtv, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
SKY_CINEMA:
type, channel, 1
channel, sky cinema, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
SKY:
>programme
)number, singular, 1
)type, programme, 2
)person, nonperson, 1
)abstract, no, 1
>;
type, channel, 1
channel, sky cinema, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;
SKY_MOVIE_PREMIERE:
type, channel, 1
channel, sky movie premiere, 1
person, nonperson, 1
number, singular, 1
abstract, no, 1
;

the last list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
recency, -1, 1
;
the previous list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
recency, -1, 1
;
the last schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
recency, -1, 1
;
the previous schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
recency, -1, 1
;

the last show list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, watch, 1
recency, -0, 1
;
the previous show list:
number, singular, 1
person, nonperson, 1
type, list, 1
```

```
abstract, no, 1
list, watch, 1
recency, -1, 1
;
the last watch list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, watch, 1
recency, -0, 1
;
the previous watch list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, watch, 1
recency, -1, 1
;
the last show schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, watch, 1
recency, -0, 1
;
the previous show schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, watch, 1
recency, -1, 1
;
the last watch schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, watch, 1
recency, -0, 1
;
the previous watch schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, watch, 1
recency, -1, 1
;

the last recording list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -0, 1
;
the previous recording list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -1, 1
;
the last record list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -0, 1
;
the previous record list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -1, 1
;
the last recording schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -0, 1
;
the previous recording schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -1, 1
;
the last record schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -0, 1
;
the previous record schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -1, 1
;
the last vcr list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -0, 1
;
the previous vcr list:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -1, 1
;
the last vcr schedule:
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -0, 1
;
the previous vcr schedule:
```

```
number, singular, 1
person, nonperson, 1
type, list, 1
abstract, no, 1
list, record, 1
recency, -1, 1
;

earlier:
relativetimeposition, min, 1
type, programme, 0
;

later:
relativetimeposition, max, 1
type, programme, 0
;

act:
type, abstract, 1
abstract, yes, 1
person, abstract, 1
;

time:
type, start time, 1
;

start time:
type, start time, 1
;

end time:
type, end time, 1
;

date:
type, date, 1
;

day:
type, date, 1
;

# This file is used for constraints imposed by concepts modifying another concept.
# format
#
# concept value1:
# >premisses1 constraint type1, premisses1 constraint value1
# ...
# >premisses1 constraint typen, premisses1 constraint valuen
# >;
# )constraint1 type1, constraint1 value1
# ...
# )constraint1 typen, constraint1 valuen
# );
# >premisses2 constraint type1, premisses2 constraint value1
# ...
# >premisses2 constraint typen, premisses2 constraint valuen
# >;
# )constraint2 type1, constraint2 value1
# ...
# )constraint2 typen, constraint2 valuen
# );
# ;
#
# note: a list automaticaly modifies a reference, in that it will be used to search for
the referent.
#

record:
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 2
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 2
);
;

tape:
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;

video:
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;
```

```
directed:
>person, person
>abstract, no
>number, singular
>;
)type, director, 1
);
>person, person
>abstract, no
>number, plural
>;
)type, directors, 1
);
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;

directs:
>person, person
>abstract, no
>number, singular
>;
)type, director, 1
);
>person, person
>abstract, no
>number, plural
>;
)type, directors, 1
);
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;

acted:
>person, person
>abstract, no
>number, singular
>;
)type, actor, 1
);
>person, person
>abstract, no
>number, plural
>;
)type, actors, 1
);
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;

acts:
>person, person
>abstract, no
>number, singular
>;
)type, actor, 1
);
>person, person
>abstract, no
>number, plural
>;
)type, actors, 1
);
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;

starred:
>person, person
>abstract, no
>number, singular
>;
)type, actor, 1
);
>person, person
>abstract, no
>number, plural
>;
)type, actors, 1
);
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;

stars:
>person, person
>abstract, no
>number, singular
>;
)type, actor, 1
```

```
);
>person, person
>abstract, no
>number, plural
>;
)type, actors, 1
);
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;

played:
>person, person
>abstract, no
>number, singular
>;
)type, actor, 1
);
>person, person
>abstract, no
>number, plural
>;
)type, actors, 1
);
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;

plays:
>person, person
>abstract, no
>number, singular
>;
)type, actor, 1
);
>person, person
>abstract, no
>number, plural
>;
)type, actors, 1
);
>person, nonperson
>abstract, no
>number, singular
>;
)type, programme, 1
);
>person, nonperson
>abstract, no
>number, plural
>;
)type, programmes, 1
);
;

# This file is used for type constraint detection
# format:
# concept type1:
# constraint type1, constraint value1
# ...
# constraint typen, constraint valuen
#
# ...
# concept typen:
# ...
#

#don't refer to yes
yes:
person, yes
type, yes
abstract, yes
;

#don't refer to dummy
dummy:
person, dummy
type, dummy
abstract, dummy
;

#don't refer to contents
contents:
type, contents
person, contents
abstract, contents
;

actor:
person, person
type, actor
abstract, no

director:
person, person
type, director
abstract, no

list:
person, nonperson
type, list
abstract, no
list, general

category list:
person, nonperson
type, list
abstract, no
list, category
```

```
start time list:
person, nonperson
type, list
abstract, no
list, start time

end time list:
person, nonperson
type, list
abstract, no
list, end time

channel list:
person, nonperson
type, list
abstract, no
list, channel

date list:
person, nonperson
type, list
abstract, no
list, date

watch list:
person, nonperson
type, list
list, watch
abstract, no

record list:
person, nonperson
type, list
list, record
abstract, no

info:
type, info
person, nonperson
abstract, no

programme:
type, programme
person, nonperson
abstract, no

channel:
type, channel
person, nonperson
abstract, no

date:
type, date
person, abstract
abstract, yes

given_date:
type, date
person, abstract
abstract, yes

command:
type, command
person, abstract
abstract, yes

topic:
type, topic
person, nonperson
abstract, no

title:
type, title
person, nonperson
abstract, no

start time:
type, time
person, nonperson
abstract, no

end time:
type, time
person, nonperson
abstract, no

start_time:
type, time
person, nonperson
abstract, no

end_time:
type, time
person, nonperson
abstract, no

category:
type, category
person, nonperson
abstract, no

time:
type, time
person, nonperson
abstract, no

duration_time:
type, time
person, nonperson
abstract, no

time_duration:
type, time
person, nonperson
abstract, no

info_command_title:
type, command
person, abstract
abstract, yes

date_and_time:
type, mixed
person, abstract
abstract, yes
```

Appendix I

Literature Survey

Reference Resolution

A literature survey

Dimitri Woei-A-Jin
November 2000

TU DELFT PHIS