Implementation of a Natural Command Language Dialogue System

Jose F. Quesada, Carlos García, J. Gabriel Amores,
Rebecca Jonson, Luis P. Prieto, Pilar Manchón,
Sergio Espinosa, Jose A. Bernal

Distribution: PUBLIC

Specification, Interaction and Reconfiguration in Dialogue Understanding Systems
IST-1999-10516

Deliverable D3.3

October 2002

IST-1999-10516 SIRIDUS

Specification, Interaction and Reconfiguration in Dialogue Understanding Systems

Göteborg University
Department of Linguistics

Linguamatics Ltd

Telefónica Investigación y Desarrollo SA Unipersonal
Speech Technology Division

Universität des Saarlandes
Department of Computational Linguistics

Universidad de Sevilla
Departamento de Lengua Inglesa

For copies of reports, updates on project activities and other SIRIDUS-related information, please look on our website at www.ling.gu.se/projekt/siridus.

For technical matters, please contact the Technical Coordinator, Robin Cooper and for administrative matters, please contact the Administrative Coordinator, Mareike Schmitt.

Prof. Robin Cooper
Department of Linguistics
Gothenburg University
Box 200
SE-405 30 Gothenburg
Sweden

Phone: +46 31 773 2536
Fax: +46 31 773 4853
cooper@ling.gu.se

Mareike Schmitt
European Project Office
Saarland University
c/o EURICE GmbH
Science Park Saar
Stuhlsatzenhausweg 69
D-66123 Saarbrucken
Germany
Phone: +49 (0) 681 959 233 66
Fax: +49 (0) 681 959 233 70
ms@eurice.de

Contents

1 Introduction **9**

2 Technical specification of hardware platform **11**

2.1 Decomposition 11

2.1.1 The PABX 12

2.1.2 The PCU (PABX Control Unit) 14

2.1.3 The VRU (Voice Response Unit) 14

2.2 Interconnectivity 15

2.2.1 PCU-PABX 15

2.2.2 VRU-PCU 15

2.2.3 VRU-PABX 17

3 Software architecture **18**

3.1 Implementing an Agent Architecture using KQML 18

3.2 Agent Manager (AM) 19

3.3 Text Input/Output Agents 21

3.3.1 TEXT-IN Agent 21

3.3.2 TEXT-OUT Agent 22

3.3.3 TRANSCRIPTION Agent 22
3.4 Speech Input / Output Agents 23
3.4.1 LINE-INT Agent 24
3.4.2 Natural Language Resources 25
3.5 PABX Control Agent 26
3.5.1 PABX Agent 26
3.6 Database Control Agent 28
3.6.1 The Database 29
3.6.2 The DB Agent 29
3.7 Dialogue Manager Agent 31
3.7.1 DM Agent 31

4 Dialogue Management **33**

4.1 Introduction 33
4.2 The architecture of the Dialogue Management Agent 35
4.2.1 Input/Output Management: LINE-INT Agent and Terminal Simulator 36
4.2.2 Action Management: PABX Agent and PABX Simulator 37
4.2.3 Knowledge Management: DataBase Agent and Corporate Directory Simulator 40
4.3 User's Interface 42
4.3.1 Introduction 42
4.3.2 Language, Register and Terminlogy 42
4.3.3 Intonation 43
4.3.4 Personality 44
4.3.5 Voice 44

4.3.6 User Confidence 45
4.3.7 Output Variety 45
4.4 Natural Language Understanding and Semantic Interpretation 45
4.4.1 Introduction 45
4.4.2 Lexicon 46
4.4.3 Grammar 47
4.5 Dialogue Manager 49
4.5.1 Dialogue Modeling and Management Components 49
4.5.2 Language Modeling based on Knowledge, Expectations and Actions: Specification Level 51
4.5.3 Expectations Control and Management 52
4.5.4 Action Execution Control 56
4.5.5 External Actions 58
4.5.6 Pre and Post Actions 59
4.6 Dialogue Move Selection 60
4.6.1 The communication between the Natural Language Understanding and the Dialogue Manager modules 61
4.6.2 Intelligent and Expectation–driven Dialogue Move Selection 65
4.6.3 The architecture of the Dialogue Move Selector 67
4.6.4 Dialogue moves fusion: unification–based equality or inclusion 68
4.6.5 Temporal priority marks 69
4.6.6 Recovery of dialogue moves 71
4.6.7 Expectation–driven dialogue move selection 72

5 Installation Guide **75**

5.1 Software components to install in the PCU 75
5.1.1 CTConnect and Ericsson ApplicationLink/CSTA 75
5.1.2 Remote shell server 76
5.1.3 PABX Agent 76
5.2 Software components to install in the VRU 76
5.2.1 Speech recognition and synthesis resources 77
5.2.2 Speech recognition models 77
5.2.3 Dialogue Management Resources 77
5.2.4 Database 79
5.2.5 Executable system agents 79

6 User's Guide **80**

6.1 General System Functionality 80
6.2 How to Activate the Commands 81
6.2.1 To make a phone call 81
6.2.2 To redial 81
6.2.3 To find out the last dialed number 82
6.2.4 To make conference call 83
6.2.5 To transfer your calls to a different number. 84
6.2.6 To cancel a call transfer 85
6.2.7 To find out the current destination of your call transfer 86
6.2.8 To Look up the email or office number of somebody in the directory . 86
6.2.9 To ask for help 87
6.2.10 To redo the last command 89

6.2.11 To list all matches 90
6.2.12 To quit the system 91
6.2.13 To have something repeated 92
6.2.14 Greetings 92
6.3 Additional Functionality 93
6.3.1 Last Utterance Repetition 93
6.3.2 Multiple Command Utterances 93
6.3.3 Temporal Markers 96
6.3.4 Recovery Actions 97
6.3.5 Task Accommodation 98
6.3.6 Task Re–Accommodation 98

Chapter 1

Introduction

This document describes the implementation of the demonstrator developed in SIRIDUS Work Package 3. This demonstrator is intended to provide a real example of development of a natural command language dialogue system within the overall context of the SIRIDUS project.

The work underlying this deliverable relates to other work being carried out inside the project. First, this demonstrator may be considered as an instantiation of the intended system architecture described in Deliverable D6.1 *Siridus System Architecture and Interface Report (Baseline)* [Lewin *et al* 2000]. Therefore, we have followed the general architecture specification described in Chapter 7 of the previous report, with the following modifications: First, the Trindi DME component (Figure 7.1, [Lewin *et al* 2000]) has been replaced by the Dialogue Manager described in Chapter 4 of Deliverable D3.2 *Design of a Natural Command Language Dialogue System* [Quesada *et al* 2000]. Secondly, we have not made use of prosodic information during the recognition and synthesis stages of the system.

However, the similarities between both architectures are worth noting. First, we have used an agent–based architecture (Chapter 3 of this document) following the KQML standard [Labrou & Finnin 1996, Finnin *et al* 1993]. Second, we have adopted an Information State Update approach to dialogue management, as proposed in the TRINDI project [Traum *et al* 1999].

This document presents first the demonstrator hardware architecture, (Chapter 2). Then, Chapter 3 describes the software architecture used in the development of the demonstrator (a multi–agent distributed architecture), as well as the agents that have been implemented and the messages they interchange in order to obtain the global functionality.

Chapter 4 makes a description of the implementation of the Dialogue Manager Agent, one of the most important agents in the demonstrator.

Chapter 5 is an installation guide for the Spanish Natural Language Command Demonstrator. The different software components of the demonstrator are described.

Finally, Chapter 6 is a user's guide, which gives a guidance of the demonstrator from the point of view as user of the system.

Chapter 2

Technical specification of hardware platform

This chapter describes the hardware platform used for the implemented Spanish Natural Command Language Demonstrator. It comprises a series of elements, which are described individually in section 2.1. This section describes the hardware elements and the role they play within the complete demonstrator.

To achieve the desired functionality it is necessary that these hardware elements interact with each other. Section 2.2 describes how this interaction is carried out through several communication lines, as well as the information they carry.

Figure 2.1 represents the demonstrators's hardware architecture, including the different hardware elements composing it, and the communication lines interconnecting them.

2.1 Decomposition

The Natural Command Language demonstrator is a system that allows personnel in a large or medium sized institution to perform simple and some advanced telephone functions, as well as some information query functions, using their voice over the phone and natural language (in particular commands issued in a natural way).

For a real–time functionality, the demonstrator needs to be composed of the following three hardware elements:

1. A PABX providing:
    - Internal telephone extensions for the people in an institution.
    - External telephone lines for external telephone calls.

2. A PABX Control Unit (PCU) that controls the PABX in order to perform the telephone functions that the system provides.

3. A Voice Response Unit (VRU) in charge of the user interaction. As this interaction will be held over the phone and using only speech and natural language, this unit is responsible for:

   - Speech recognition.
   - Speech synthesis.
   - Natural language processing.
   - Dialogue management.

The following three subsections describe each of these three hardware elements.

2.1.1 The PABX

The demonstrator uses an Ericsson MD110 PABX currently in operation at Telefónica I+D.

This PABX currently provides about 200 internal extensions for the employees of Telefónica I+D at Valladolid.

The following telephonic services are accessible (by dialling the appropriate codes) from the internal extensions:

1. Internal telephone calls.
2. External telephone calls.
3. Redialling.
4. Multi party conference (up to 8 speakers).
5. Automatic notification when an internal extension gets free.
6. Notification of a second incoming call.
7. Switching between two active incoming calls.
8. Transference of an established incoming call.
9. Capture of incoming calls from another extension.
10. Automatic transference of incoming calls (immediate / if busy / if there is no answer).

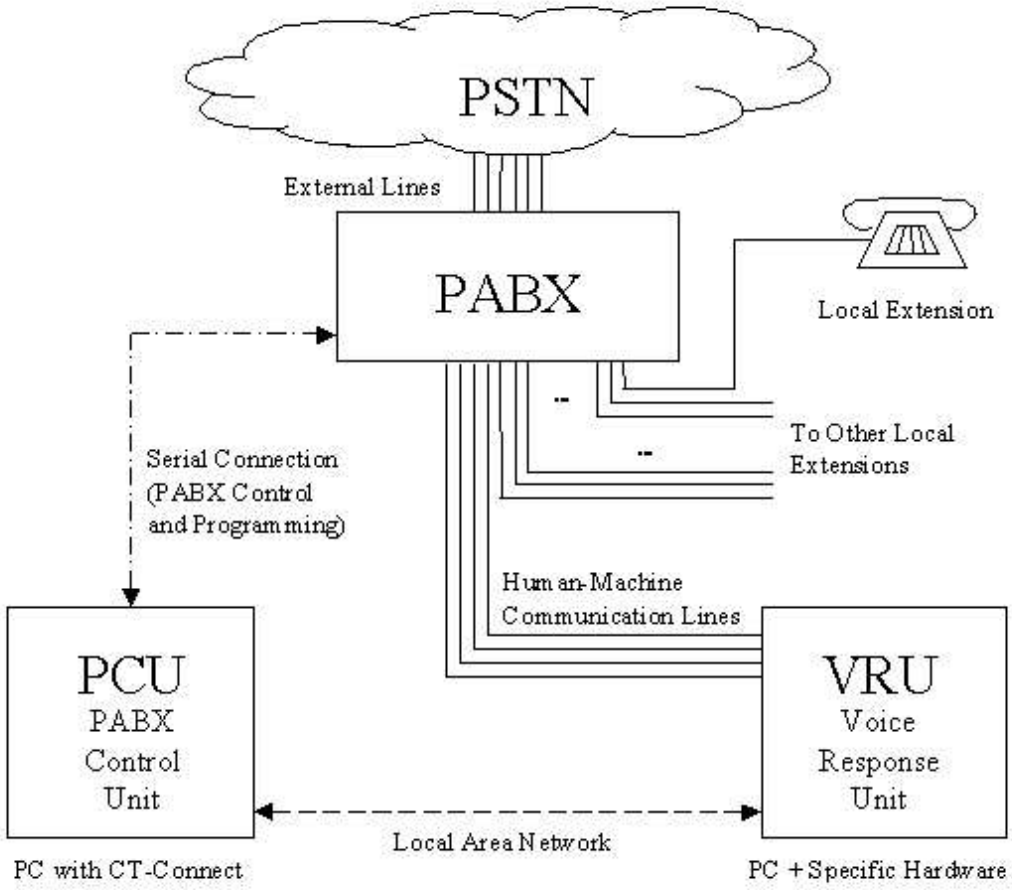

Figure 2.1: Hardware Architecture Decomposition and Connectivity

However, in the demonstrator only some of these telephone services (only services described under 1, 2, 3, 4 (for 3 speakers) and 10) have been made accessible through the automatic dialogue system, via voice over the phone. In order to do so, it has been necessary to control the PABX in a completely different way, from an external computer connected to the PABX itself and to the computer on which the dialogue system runs. We have called this external computer *PABX Control Unit* (*PCU*), and the computer in charge of the dialogue system and all the aspects related to the user input/output via voice *Voice Response Unit* (*VRU*). Both computers are described in the next two sections.

2.1.2 The PCU (PABX Control Unit)

The PABX Control Unit (PCU) is a Pentium III PC running under Windows NT 4.0.

In order to be able to control the PABX it needs two special software packages:

- CTConnect V4.0.

  This software facilitates the control of several PABXs of different vendors from an external PC. Only several models of PABXs are supported. The software is only available for a limited number of platforms, not including Solaris for PC.

- Ericsson ApplicationLink/CSTA V3.0.2.

  This is a lower level software, specific for the Ericsson PABX, that allows CTConnect to effectively control this particular model of PABX.

2.1.3 The VRU (Voice Response Unit)

The Voice Response Unit (VRU) is in charge of the system–user interaction. This interaction will be held entirely via voice over telephone lines. Therefore, it needs software to perform voice recognition and voice synthesis over the phone.

In order to provide these means, the VRU consists of a Pentium II PC running under SUN Solaris 2.5.1 and some specialised hardware and software:

- Two Dialogic Antares 3000/50 PC Boards

  These are DSP boards with four Texas Instruments DSPs. They improve the digital signal processing capabilities of the PC in order to be able to process digital samples of telephonic speech for recognition more efficiently, and to be able to produce digital samples of telephonic synthetic speech.

  In the VRU one of these boards is used to process the samples of the user's voice in order to perform the speech recognition. Telefónica's Speech Recogniser runs partially on this board.

The other board is used to generate a synthetic voice that will constitute the system's output towards the user. Telefónica's Speech Synthesiser runs on this board.

- A Dialogic D41E PC board.

  This board provides an interface for up to four telephone lines, which can be sampled and connected digitally to the Antares DSP boards through a specialised bus (SC-BUS) different from the internal PC bus.

  The libraries that come along with this board allow a program to perform basic functions with the telephone line, including waiting for an incoming telephone call, connecting the incoming telephone call to an Antares Board, and disconnecting the line.

Figure 2.2 represents the hardware elements that compose the VRU, as well as the interconnections between them and the external inputs and outputs.

2.2 Interconnectivity

In order to perform the intended functions, the three hardware elements that compose the demonstrator need to be interconnected and interchange information of different sorts.

2.2.1 PCU-PABX

The PABX Control Unit (PCU) is connected to the PABX (as can be seen in Figure 2.1) by means of a serial line. Through this line, the PABX control unit issues control commands to the PABX and receives notifications from the PABX.

The communication through this serial line is made possible through the CTConnect software and the underlying Ericsson ApplicationLink software. Making use of this software, an application has been developed to properly handle the notifications from the PABX and to issue the commands to the PABX as required. This application runs on the PCU and needs to establish a communication with the Voice Response Unit (VRU). This communication is analysed in the next section.

2.2.2 VRU-PCU

The demonstrator has as its main functionality to facilitate the access to several telephone functions by means of the natural language speech over the phone. For that reason, it is essential that communication exists between the Voice Response Unit (VRU), in charge of the voice interaction with the user, and the PABX Control Unit.

This communication is made through a Local Area Network (LAN) (as shown in Figure 2.1) to which both computers (PCU and VRU) are connected.

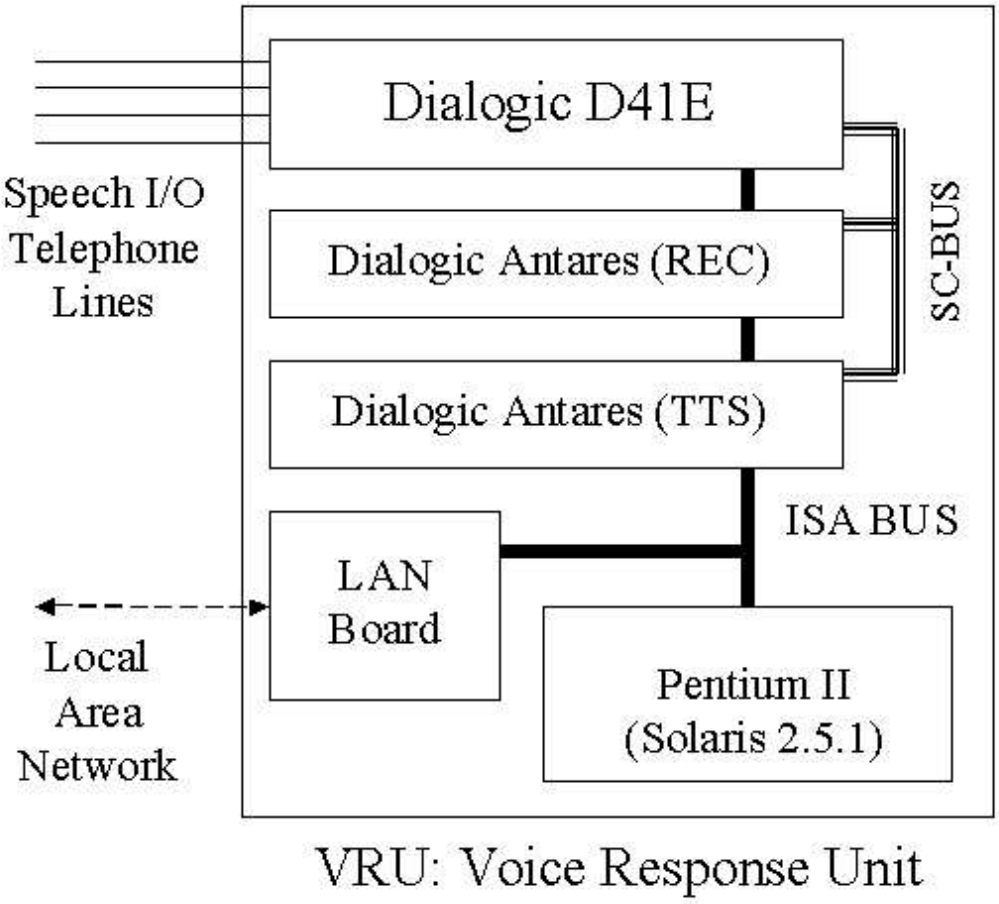

Figure 2.2: Voice Response Unit Hardware Architecture

This communication occurs from the VRU to the PCU, as its main goal is to allow the VRU to send the appropriate commands to the PCU in order to perform the functions requested by the user. This way the PCU is mainly in charge of translating these commands into PABX commands, which are, in turn, sent to the PABX.

Communication in the opposite direction is also necessary as the PABX agent must tell the DM agent some pieces of information, such as the extension number of the user and the state of the current task.

2.2.3 VRU-PABX

As can be seen in Figure 2.1, the VRU is connected to the PABX by means of up to four telephone lines. All system–user interaction will be held through these telephone lines, which will carry both the user's and the synthetic system's voice.

These telephone lines also carry the usual telephone signalling information (i.e. incoming call tone), which the VRU needs to handle as well (in particular, the telephone line interface PC board will handle it).

Chapter 3

Software architecture

This chapter describes the implementation of the software architecture for the demonstrator that has been developed in the SIRIDUS Work Package 3. This software runs on the two PCs that form part of the demonstrator, the PABX Control Unit (PCU) and the Voice Response Unit (VRU).

3.1 Implementing an Agent Architecture using KQML

The software architecture design pursued basically three characteristics: modularity, multi–platform facilities and distributed execution, as the software was going to be developed between three different groups, with two computers with different operating systems. To be able to fulfill these requirements, a multi–agent distributed architecture was chosen.

The software was therefore divided into modules (or agents) that play a particular role within the complete system, and that communicate with each other in order to obtain the global functionality.

Each agent has been built as a separate program that communicates with other agents through KQML ASCII text messages. KQML stands for Knowledge Query and Manipulation Language, and is a standardised language for inter–agent communication. Different specification versions of this language can be found in [Labrou & Finnin 1996, Finnin *et al* 1993].

The following example of a KQML message shows the simple syntax of this agent communication language:

```
(request :sender X :receiver Y :content (quit))
```

This message could be interpreted such as that the agent X has sent a request with the content

quit to agent Y, i.e. it asks agent Y to finish its execution.

The agent architecture we have opted for the demonstrator is a centralised architecture, in the sense that there is a central agent (Agent Manager, AM) through which all the messaging pass. This central agent routes the messages to their destinations, and provides additional services to the rest of the agents integrating the demonstrator.

As a result of the adoption of a multi–agent distributed architecture for our demonstrator, the software that has been developed can be described as a set of independent C programs or agents that run concurrently in the same or different machines and interchange KQML messages in order to achieve the desired functionality.

The following sections describe the different agents that are considered for the demonstrator, as well as the messages that they are able to handle and send to other agents. These messages are the only interaction between agents. Therefore, describing the messages handled by an agent is equivalent to a complete description of its interface.

Figure 3.1 shows the agents integrating the demonstrator. As can be seen, all the agents run on the Voice Response Unit (VRU). The only exception is the PABX control agent, which runs on the PABX Control Unit (PCU).

3.2 Agent Manager (AM)

This agent is the centre of communications of the agent architecture. All messages between any two agents necessarily pass through this agent.

Every agent in the architecture has to register by sending a message to the AM before it can send a message to any other agent. This way the AM agent maintains a record of the agents connected to the system, and knows how to send a message to any of the connected agents.

- Messages sent:

  None. It only routes messages to the appropriate agents.

- Messages received:

  - `(register :receiver AM :name N)`
    This causes an agent to be registered by the AM with the name N.
  - `(tell :receiver AM :content (ready))`
    This message is sent by each agent to inform the AM that the agent is ready to receive messages.
  - `(broadcast :receiver AM :content C)`
    This makes the AM to send a message with content C to all the agents connected at that moment.

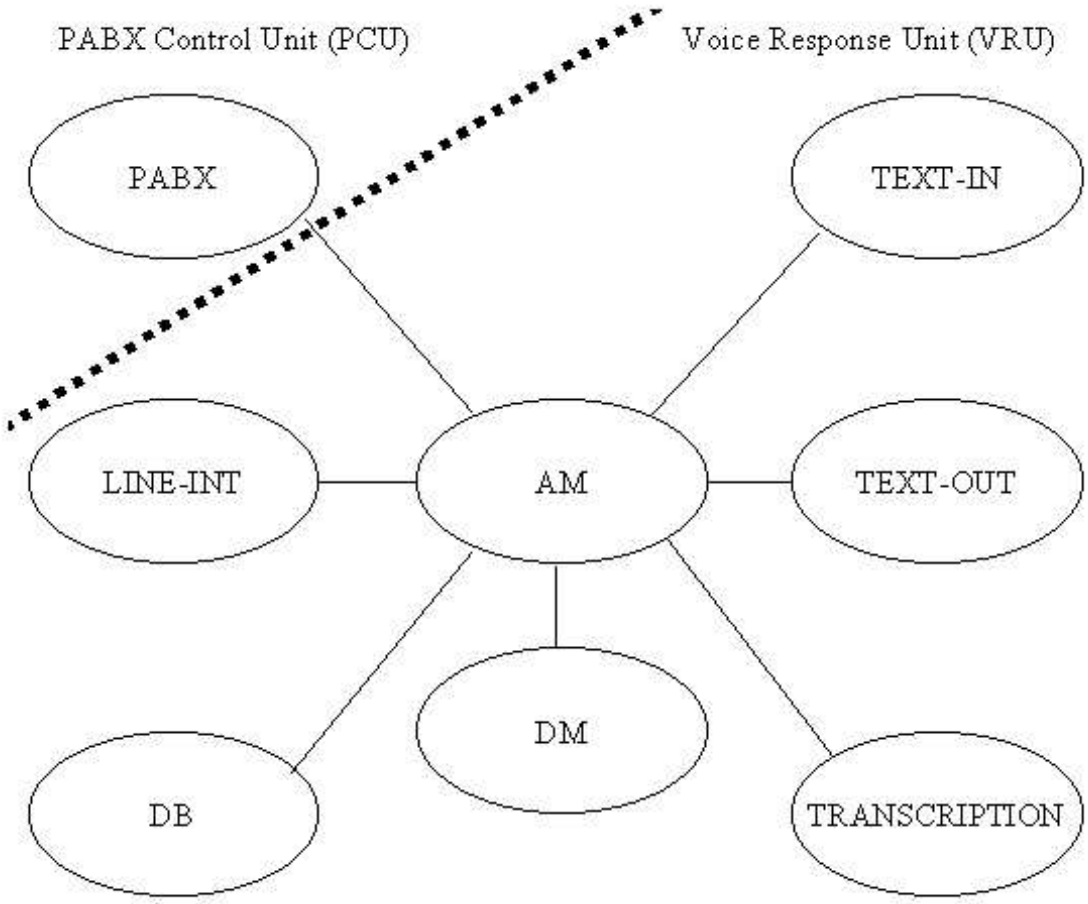

Figure 3.1: Software Architecture. Distributed Multi–Agent Architecture

– Finally the AM will receive several messages with different contents of the type tell, request and reply that are directed, via the AM, from one agent to one or several agents.

3.3 Text Input/Output Agents

The main input and output modality in the demonstrator to be produced in SIRIDUS work package 3 is voice over the phone. However, in the development and use of the demonstrator it was convenient to have a collection of agents allowing the user to introduce text as input and to obtain text as output. We also thought it would be interesting to have the complete transcription of the dialogue in text format. For that reason the following agents were developed.

3.3.1 TEXT-IN Agent

This agent allows a user to introduce textual input to the demonstrator. This way the demonstrator works with either speech or text input, which is very convenient for its debugging and also for performing demonstrations.

This agent simulates the message sent to the rest of the demonstrator when a speech input has been produced (provided that the recognised sentence is the same as the typed text). Therefore, for the rest of the demonstrator it is indistinguishable whether the input has been spoken or typed.

- Messages sent:
  - `(register :receiver AM :name "TEXT-IN")`
    TEXT-IN agent registration message.
  - `(tell :receiver AM :content (ready))`
    This message is sent to inform the AM that TEXT-IN is ready to receive messages.
  - `(broadcast :receiver AM :content (tell :content (user-input "xxxx")))`
    Each time a text input is finished, the TEXT-IN agent sends a broadcast message to inform all the agents in the system of the new user input. The agents with most interest in this information, however, are the dialogue manager (DM) agent and the TRANSCRIPTION agent. Other agents discard this message.
- Messages received:
  - `(tell :content (line-start))`
    When a user calls the system and the LINE-INT broadcasts this message, the TEXT-IN agent opens and resets the window that shows the user's text input.

- `(tell :content (line-stop))`
  When the call finishes and the LINE-INT broadcasts this message, the TEXT-IN agent closes the window that shows the user's text input.

3.3.2 TEXT-OUT Agent

The system always produces output in two modalities: voice over the phone and text on the screen. The main modality is, however, voice over the phone, being text on the screen only a useful add–on for demonstration and debugging purposes.

This agent presents the output of the system in textual form on the screen of the VRU. In order to do so, it processes the broadcast messages which the LINE-INT agent issues to inform the rest of the system of a new spoken system output.

- Messages sent:
  - `(register :receiver AM :name "TEXT-OUT")`
    TEXT-OUT agent registration message.
  - `(tell :receiver AM :content (ready))`
    This message is sent to inform the AM that TEXT-OUT is ready to receive messages.
- Messages received:
  - `(tell :content (advanced-system-output "xxxx"))`
    This message is sent by the LINE-INT agent to confirm that a new system's spoken output has been issued.
  - `(tell :content (line-start))`
    When a user calls the system and the LINE-INT broadcasts this message, the TEXT-OUT agent opens and resets the window that shows the system output.
  - `(tell :content (line-stop))`
    When the call finishes and the LINE-INT broadcasts this message, the TEXT-OUT agent closes the window that shows the system output.

3.3.3 TRANSCRIPTION Agent

This agent maintains the transcription of the whole user–system dialogue on the screen of the VRU in a readable format. As the other console input/output agents, its main utility is to facilitate the demonstration of the system and for debugging and dialogue refining purposes.

- Messages sent:

  - `(register :receiver AM :name "TRANSCRIPTION")`
    TRANSCRIPTION agent registration message.
  - `(tell :receiver AM :content (ready))`
    This message is sent to inform the AM that TRANSCRIPTION is ready to receive messages.

- Messages received:

  - `(tell :content (user-input "xxxx"))`
    This message is sent by either the LINE-INT agent or the TEXT-IN agent to inform all the system of a new user input (either spoken or written). This way the TRANSCRIPTION agent can update the transcription on the screen.
  - `(tell :content (advanced-system-output "xxxx"))`
    This message is sent by the LINE-INT agent as a confirmation of a new system output, and is used by the TRANSCRIPTION agent to add the system output to the transcription on the screen.
  - `(tell :content (line-start))`
    When a user calls the system and the LINE-INT broadcasts this message, the TRANSCRIPTION agent opens and resets the window that shows the dialogue transcription.
  - `(tell :content (line-stop))`
    When the call finishes and the LINE-INT broadcasts this message, this agent closes the window that shows the dialogue transcription.

3.4 Speech Input / Output Agents

The agent described in this section allow the demonstrator to work with speech input and output over a telephone line, which is the main input/output modality in the Natural Command Language Demonstrator.

The restriction of using voice over the phone imposed a series of additional requirements on the functionality. We had to manage not only a speech recogniser and synthesiser, but also a telephone line. In particular, the state of the telephone line had to be monitored in order to connect and disconnect the line to the speech recogniser and synthesiser when appropriate, and to notify the rest of the system when an incoming call was received and when the telephone call had ended.

Due to these requirements it seemed to be easier to group all the functions related to the telephone line (including speech recognition and synthesis) in a single agent which pretends to be an abstraction of the telephone line. We have called this agent the Telephone Line Interface (LINE-INT) agent.

3.4.1 LINE-INT Agent

This agent provides the necessary interface to control a telephone line with speech recognition and speech synthesis capabilities.

- Messages sent:
  - `(register :receiver AM :name "LINE-INT")`
    LINE-INT agent registration message.
  - `(tell :receiver AM :content (ready))`
    This message is sent to inform the AM that LINE-INT is ready to receive messages.
  - ```
    (broadcast :receiver AM
               :content (tell :content (line-start)))
    ```
    This way LINE-INT notifies all the agents in the system that an incoming call has been established. This message that indicates that a new conversation starts is processed by the Dialogue Manager agent, as well as by the graphical agents the TRANSCRIPTION agent, the TEXT-OUT agent and the TEXT-IN agent.

    ```
    (broadcast :receiver AM
               :content (tell :content (line-stop)))
    ```

    This way LINE-INT notifies all the agents in the system that a previously established telephone call has ended (on the user or system's initiative). This message, which indicates that the conversation has finished is of great interest for the Dialogue Manager agent (DM). However, the graphical agents, TRANSCRIPTION, TEXT-OUT and TEXT-IN also takes this message into account as they need to know when they should close their windows.

- Messages received:
  - `(request :receiver LINE-INT :content (say "xxxx"))`
    This message is sent to the LINE-INT agent to ask it to produce a synthetic speech output to the user.
    LINE-INT will respond with the following message to indicate that the output sent to it is going to be produced as synthetic speech.

    ```
    (broadcast :receiver AM
               :content (tell :content
                        (advanced-system-output "xxxx")))
    ```

    This message is used by the TRANSCRIPTION and the TEXT-OUT agents.
    Once the synthetic speech has been produced and sent to the user, LINE-INT replies with a broadcast message indicating that a new system output has been produced.

    ```
    (broadcast :receiver AM
               :content (tell :content (system-output "xxxx")))
    ```
    ```

– `(request :receiver LINE-INT :content (recognize))`
  This message is sent to the LINE-INT agent to start the speech recogniser. The speech recogniser ends automatically when it detects a long silence. When it has finished, the LINE-INT agent broadcasts a message to inform all the agents in the system of the recognition results.

  ```
  (broadcast :receiver AM
             :content (tell :content (user-input "xxxx")))
  ```

  This message is used by the Dialogue Manager (DM) agent and the TRANSCRIPTION agent.
– `(request :receiver LINE-INT :content (line-stop))`
  This message requests the LINE-INT agent to disconnect a previously established telephone call. This message may be sent by the Dialogue Manager (DM) agent to ask the LINE-INT agent to finish a conversation with a user.
  When the telephone call has finished the LINE-INT agent informs the rest of the agents by broadcasting the message

  ```
  (broadcast :receiver AM
             :content (tell :content (line-stop)))
  ```

3.4.2 Natural Language Resources

The LINE-INT agent receives voice recognition and synthesis requests from other agents in the system. When such requests are received it makes use of the speech recogniser and the TTS system developed at Telefónica I+D.

The recogniser is a large vocabulary continous speech recognition system which makes use of acoustic and syntactic information to select the sequence of words most probably uttered by the user.

In fact, it employs Continuous HMM to model triphonemes, and makes use of statistical language models based on trigrams.

The vocabulary and language model for this demonstrator have been obtained from a corpus of sentences in Spanish. These sentences are a simulation of different possible interactions of users with this system. A corpus from Telefónica I+D was initially used, and then it was partially modified by the University of Seville, based on the initial tests of the system they made.

The demonstrator uses a vocabulary of about 2500 words, which include names, numbers, commands and other Spanish words.

The TTS engine is a system based on unit concatenation. It can synthetise Castilian Spanish words and sentences, and runs on one of the Antares Boards used in the demonstrator. A detailed description of this system can be found in [Castejón *et al* 1994].

3.5 PABX Control Agent

The control of the PABX is the responsibility of the PABX Control Unit (PCU). The software involved in this control runs on this computer. The agent in charge of controlling the PABX is the PABX agent.

3.5.1 PABX Agent

This agent accepts requests from the Dialogue Manager (DM) agent asking it to perform several control actions over the PABX, such as connecting two or more extensions, establishing/cancelling the forwarding of incoming calls, etc.

The messages that this agent handles are therefore mainly interchanged with the dialogue manager with exception of the registration and ready messages. The message communication that takes place with this agent is the following:

- Messages sent:
  - `(register :receiver AM :name "PABX")`
    PABX agent registration message.
  - `(tell :receiver AM :content (ready))`
    This message is sent to inform the AM that PABX is ready to receive messages.
  - `(tell :receiver DM :content (connected :dn "xxxxx"))`
    This message is sent to inform the DM that a connection between the user and other terminal/s (i.e. a phone call) was successful.
  - `(tell :receiver DM :content (notconnected :dn "xxxxx"))`
    This message is sent to inform the DM that a connection between the user and other terminal/s (i.e. a phone call) was unsuccessful, for a variety of reasons.
  - `((reply :content (incoming_ext :extension "xxxx"))`
    This message is sent as a response to a request made by any other agent, inquiring which extension is currently connected to the VRU.
  - `(error :receiver xxxx :in-reply-to yyyy`
    `:code zzzz :comment "uuuu:vvvv")`
    This message is sent when any error situation arises in the PABX agent.
- Messages received:
  - `(request :receiver PABX :content (get_incoming_ext))`
    This message asks the PABX control agent about which is the extension currently connected to the VRU (we'll call it the "active" extension).

– `(request :receiver PABX :content (startline :dn "xxxxx"))`

  This message asks the PABX control agent to "register" certain extension, so the PABX can control it from now on.

– `(request :receiver PABX :content (endline :dn "xxxxx"))`

  This message asks the PABX control agent to "unregister" certain extension, so the PABX will not be able to control it any more (or until it is registered again).

- `(request :receiver PABX :content (activate-forwarding :dn "xxxxx" :to "yyyyy"))`
  This message asks the PABX control agent to activate the forwarding of incoming calls in the "active" extension (xxxxx) to another extension (yyyyy).
- `(request :receiver PABX :content (deactivate-forwarding :dn "xxxxx"))`
  This message is sent to the PABX control agent to deactivate the forwarding of incoming calls from the current extension (xxxxx) to another extension number.
- `(request :receiver PABX :content (makecall :dn "xxxxx" :to "yyyyy"))`
  This message asks the PABX control agent to connect the "active" extension (xxxxx) to another extension or external line (yyyyy).
- `(request :receiver PABX :content (conference :dn "xxxxx" :to1 "yyyyy" :to2 "zzzzz"))`
  This message asks the PABX control agent to connect the "active" extension to two other extensions or external lines.
- `(request :receiver PABX :content (exit))`
  This message tells the PABX control agent to stop controlling all the extensions and finish its execution.

When the PABX agent is executed, and after it is connected and registered with the Agent Manager (AM), it waits for requests. When one of these requests arrives, it processes it and fulfills the request, using the underlying API, CTConnect (which, itself, uses the Application-Link API). These requests can include:

- Activating and deactivating the control which this agent has over an extension.
- Information about which (if any) extension is currently connected to the VRU.
- Activation and deactivation of call forwarding in the "active" extension. In the process, the user is momentarily disconnected from the VRU.
- Making calls or multi conferences between the "active" extension and one or more extensions or external lines. In this case, the user is "held" until the other party is contacted or (if the connection failed for any reason) it is reconnected to the VRU for further conversation with it. In either case, the PABX control agent notifies the requester about the result of the action (i.e. whether it was successful or not).

3.6 Database Control Agent

The demonstrator has been designed to provide voice access to two external resources: the PABX and a database containing the directory of an institution (Telefónica I+D in this case).

The PABX and its control is conceptualised as an agent, and the database is conceptualised as another agent, the database (DB) agent.

This way, all the necessary interactions with the PABX will be done through the PABX agent, and all necessary interactions with the database will be done through the DB agent.

3.6.1 The Database

The database server that has been deployed in the demonstrator is a MYSQL server, as this seemed to be a fast and easy open source choice. The database that has been applied consists of the directory of the personnel in the example institution, Telefónica I+D, that arounds 1300 persons.

The database contains the following fields for each user of the system (each person in the example institution, Telefónica I+D):

- Database record number (`record`).
- First name (`name`).
- First surname (`surname1`)
- Second surname (`surname2`)
- Telephone extension (`extension`).
- Office number (`office`).
- E-mail address (`email`).
- The extension to which this terminal is forwarded (`fwdext`), or 0 if not forwarded.
- The last number called by the user (`lastcall`).
- The service mode (`mode`). Each user may choose between having this automatic service deactivated, partially or completely activated. This field indicates the degree of activation of the automatic service chosen by the user.

3.6.2 The DB Agent

The DB Agent handles requests from the dialogue system to access the database in either read or write mode. In addition, it also takes care of the start–up of the MYSQL server, in case this is not already running.

The DB agent allows access to all the information gathered in the database. Under certain circumstances the data supplied may match several entries.

The interface with the DB Agent consists of the following four functions (supported as messages between the agents).

- Request a select operation.
- Return the number of entries that match the pattern of the last select operation.
- Return the Nth selected record.
- Update a record.

In cases of ambiguity (more than 1 record matches the query) these operations allow the dialogue manager to set up a disambiguation sub–dialogue. For instance, after the user has requested to call Peter, and if there are two 'Peter' entries, the system should ask the user:

Do you want to call Peter Johnson or Peter Smith?

instead of just repeating

Who do you want to call?

The messages that the agent receive are interpreted into SQL messages which are the communication way with MYSQL server.

- Messages sent:
  - `(register :receiver AM :name "DB")`
    DB agent registration message.
  - `(tell :receiver AM :content (ready))`
    This message is sent to inform the AM that DB is ready to receive messages.
- Messages received:
  - `(request :content (select :name XXXX`
    `:surname1 YYYY`
    `:surname2 ZZZZ`
    `<other field-value specifications> ) )`
    This message requests the DB agent to look for all the records in the database that match the partial record specification in the select command and to create a temporal table with the records found. The answer from the DB agent takes the form:

```
(reply :content (found :matches N) )
```

This answer indicates the number of records that match the currently activated select message.

- `(request :content (get :record_number N) )`
  This message requests the database agent to return the Nth record that matched the previous select operation. The database agent will reply to this message with a message that indicates the values of every field in that record:

```
(reply :content (record :record_number N
                        :<field1> <value1>
                        ....
                        :<fieldZ> <valueZ> ) )
```

- `(request :content (update :extension N`
  `:<field_to_modify> <new_value>`
  `....`
  `:<field_to_modify> <new_value> ) )`
  This message asks the DB agent to modify (update) the specified fields of the record with the extension N. The only fields permitted for modification are the forwarded extension, the last call and the mode. In addition, the DB agent only allows the DM agent and PABX agent to modify the database (messages always include a field :sender that specifies the sender of the message, so that the DB agent may identify the agent that requested an update command).
  The DB agent may reply indicating that the operation has been successfully executed with the following message:

```
(reply :content (updated))
```

or with an indication of an error

```
(reply :content (update-error))
```

3.7 Dialogue Manager Agent

Last but not least, it is time to describe the agent in charge of the natural language processing and the interpretation of the user's input, which is the Dialogue Manager (DM) agent.

3.7.1 DM Agent

This agent receives as its input the results of the speech recogniser (output of the LINE-INT agent) or the output of the TEXT-IN agent and analyses it and processes it, generating the appropriate messages to the PABX control (PABX) agent and the database (DB) agent, processing their answers, and generating an appropriate output that is sent to the LINE-INT agent to produce the synthetic voice.

- Messages sent:

  - `(register :receiver AM :name "DM")`
    DM agent registration message.
  - `(tell :receiver AM :content (ready))`
    This message is sent to inform the AM that the DM is ready to receive messages.

  The DM agent, is a central figure in the system and is responsible for most of the input messages that are sent to the other agents.

  - The communication with the LINE-INT consists of requesting the LINE-INT to produce a specific speech output, to be ready for recognizing or to finish an established call.
  - The messages sent to the PABX are instructions of control actions that the DM wishes the PABX to perform, such as connect two extensions or establish the forwarding of incoming calls.
  - The DM also sends requests to the DB to retrieve certain information from the database and to update a user's record.

  All these messages are represented in their correct form in the correspondent section for the agent that sends them to the DM.

- Messages received:

  The DM agent manages most of the output messages from the PABX, DB, LINE-INT and TEXT-IN agents.

  - The DM will be informed by the LINE-INT agent about the state of the line, i.e. if a conversation is starting or has finished. Additionally, it will also take into account the messages received with the user's input that it will analyse and process. The input may as well be textual and sent from the TEXT-IN agent. Another message that the DM expects is the state of TTS, i.e. it needs to know when the synthesiser finish the system's speech output production.
  - From the PABX the DM will expect information about the extension number corresponding to the current telephone call. It also waits for messages informing about the state of the operations that the PABX is performing
  - The messages received from the DB will be replies with the information that the DM has requested.

  The messages described are represented in their KQML form in the section of messages sent for the correspondent agent.

The next chapter contains a more detailed description of the internal implementation of the natural understanding and dialogue manager modules.

Chapter 4

Dialogue Management

4.1 Introduction

Chapter 3 has concentrated on the study of the overall software architecture of the system. It also described the set of agents and the messages they interchange.

Chapter 4 will describe the specific structure of the dialogue management agent.

Basically, the Dialogue Management Agent (DMA) may be described as the intelligent kernel of the whole system. This agent in fact controls and drives the rest of agents. Once the telephone connection has begun, depending on the user's interaction, this agent decides when to consult the database, asks the user for new information or commit some operations.

During its implementation, we had to take into account different technical constraints. These were described in Deliverable 3.1 *User Requirements on a Natural Command Language Dialogue System* [Torre, Amores & Quesada 2000].

Two main technical constrainsts determined the strategy for the design and implementation of this agent:

1. The hardware, software and communication infrastructure currently used by Telefonica I+D imposed the need of the use of C/C++ for the implementation of all the agents involved.
2. The use of KQML as the agent communication architecture.

Taking into account these constraints, we decided to use Episteme and Delfos for the implementation of the Dialogue Management agent. These tools were previously developed by the Natural Language Processing Research Team of the University of Seville, and they are

implemented in C. They had also been used in the DHomme project (Dialogue in the Home Machine Environment) for the design and implementation of a Dialogue Management agent in the home domain.

In order to allow the use of Episteme and Delfos in the Automatic Telephone Operator scenario of Siridus it has been necessary to design and implement a KQML wrapper over these tools. This wrapper isolates the internal structure of the Dialogue Management agent from the specific characteristics of the KQML and telephone environments.

The use of the agent requires then a real and global scenario including the KQML software infrastructure, the hardware and software for the connection with real telephone lines and the use of a PABX. This complete scenario is available in the headquarter of Telefonica I+D in Madrid, and has been partially reproduced at the University of Seville (without the PABX).

In order to allow the demo of the system in a less demanding environment, we considered the development of a parallel system based on OAA. In this scenario, we have substituted the real telephone and PABX infrastructure by simulated software agents.

As a result, two different systems have been obtained as part of Work Package 3.

The first one, called K-TeleDelfos2, based on KQML and able to use a real PABX and telephone lines. The second one, called O-TeleDelfos2, based on OAA, uses a simulated environment.

Both systems share the Dialogue Management agent (TeleDelfos2). The internal structure of this agent as well as its connection to the other different agents is described in section 2.

Internally, the Dialogue Management Agent is composed of a set of modules. These are:

- the Natural Language Understanding and Semantic Interpreter,
- the Dialogue Manager.
- the Dialogue Move Selector,

An additional module was considered as part of the design of the Dialogue Management Agent, the Speech Prospector. This module has been developed. It mainly allows the context–free parsing of word–lattices. Nevertheless, due to technical limitations, this module has not been incorporated into the TeleDelfos2 implementation.

The reason is that it is not possible to access the word–laticce of the Speech Recogniser of Telefonica I+D once it is running over the telephone interface card. Instead, the design, implementation and evaluation of the strategy of the Speech Prospector has been included as part of Work Package 2, and it has been described in Deliverable 2.3.

Next, section 2 concentrates on the internal structure of the Dialogue Management agent.

The KQML strategy (agents and messages) has been covered in chapters 2 and 3 of this document. The OAA set of agents is also described in section 2.

Section 3 describes the interface with the final user. This section focuses on aspects such as the specific domain, the language register and other aspects of the interface design.

The following sections concentrate on the previously enumerated main components of the Dialogue Management Agent. Section 4 describes the Natural Language Understanding and Semantic Interpreter module, and specifically the lexicon and grammar developed to cover the automatic telephone operator scenario.

Section 5 analyses the Dialogue Manager module. It describes the dialogue rules implemented following the Information State Update approach. Also, it is worth noting the incorporation of several strategies to improve the naturalness and flexibility of the dialogue system, following the guidelines on cooperative behaviour discussed in Deliverable 1.4.

Section 6 describes the Dialogue Move Selector. The implemention of the Dialogue Move Selector as an independent module in the Dialogue Management agent is one of the main innovative results of this Work Package. This module permits the manipulation of very common and natural linguistic and discourse phenomena where the semantic interpretation of the dialogue moves and their order of processing is determined by the user's utterance, the dialogue history and the current expectations triggered by the dialogue manager.

4.2 The architecture of the Dialogue Management Agent

One of the main goals in the design of the architecture of this agent has been portability. Taking into account the special characteristics of the automatic telephone scenario, the portability goal becomes a very hard challenge.

Ir order to allow such portability without lossing efficiency, the architecture of the Dialogue Management agent is divided in two main blocks: the kernel and the wrapper.

1. **Dialogue Management Kernel**. This component has been built using Episteme and Delfos. This way, the kernel may be described as a general tool for natural language processing (Episteme) and Dialogue Management (Delfos).

   The kernel itself contains four main modules:

   (a) *Natural Language Understanding and Semantic Interpreter* (described in section 3 of this chapter).
   (b) *Dialogue Management* (described in section 4 of this chapter).
   (c) *Dialogue Move Selection* (described in section 5 of this chapter).

2. **Dialogue Management Wrapper**. This wrapper implements the message interchange functions specific of the agent architecture.

As this component is dependent of the agent architecture, two different wrappers have been implemented.

(a) *KQML Dialogue Management Wrapper.* This wrapper has been implemented in C++ and supports the KQML functions needed for the communication between the Dialogue Management agent and:

   i. *The LINE-INT agent*, which connects a user through a telephone line.

   ii. *The PABX agent*, which controls the PABX and supports the phone call, transfer call and conference call functions.

   iii. *The DB agent*, which contains all the information about the directory of the corporation, in this case, the main building of Telefonica I+D in Madrid.

(b) *OAA Dialogue Management Wrapper.* This wrapper has been implemented in C and supports the OAA functions needed for the communication between the Dialogue Management agent and the agents of the simulated scenario:

   i. *The Terminal Simulator.* This agent simulates a telephone terminal. It also incorporates the speech input/output functionality.

   ii. *The Action Manager or PABX Simulator agent.*

   iii. *The Knowledge Manager or Corporate Directory agent*

It is worth noting that by means of the wrapper–based technology exactly the same Dialogue Management agent is being used in both scenarios: the real and the simulated.

4.2.1 Input/Output Management: LINE-INT Agent and Terminal Simulator

The functions controlled by the input/output interface (between the Dialogue Management and the LINE-INT agents) are:

1. CallStart (line-start message)
2. CallStop (line-stop message)
3. SystemOutput (say and system-output messages)
4. UserInput (recognize and user-input messages)

Terminal Simulator

This agent has been implemented in Java usign the Swing toolkit for the design of its graphical user interface (figure 4.1).

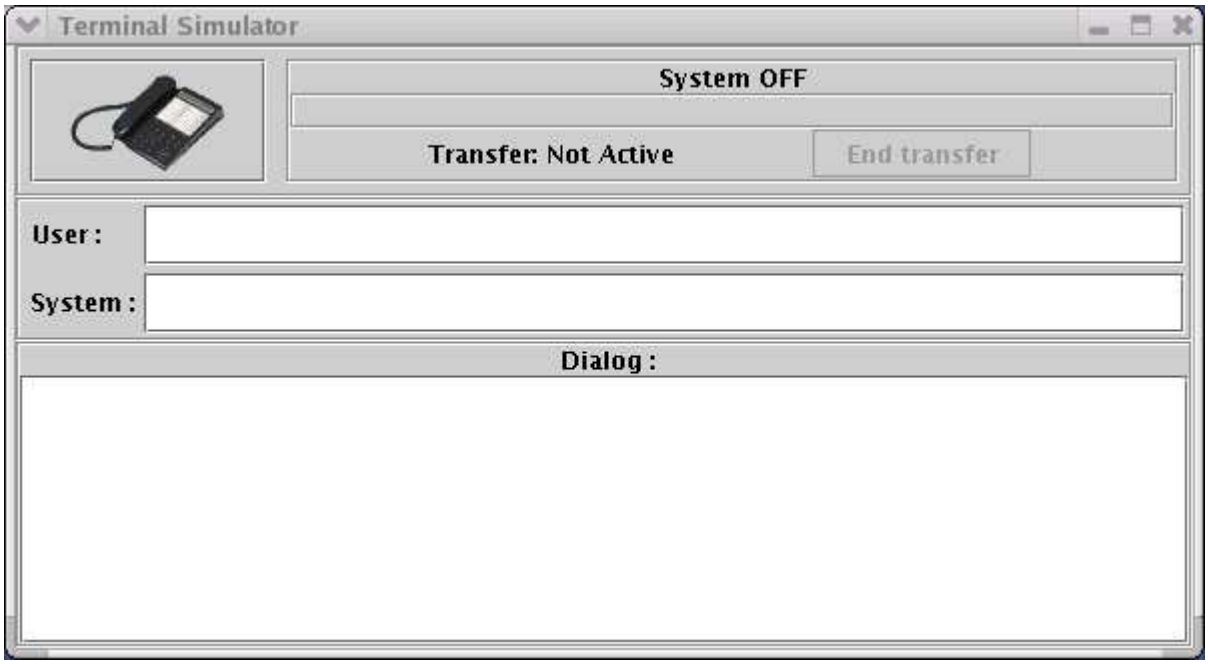

Figure 4.1: Terminal Simulator agent (GUI)

It is possible to initiate a phone conversation with the dialogue manager clicking on the telephone icon at the top–left corner.

The user and system fields contains the last user and system utterance, respectively.

The dialog text area contains a transcription of the full dialogue so far. Figure 4.2 shows an interaction where the user first asked to cancel the transfer mode and make a phone call. The picture has been taken in the simulated moment when the terminal simulator is making the phone call.

4.2.2 Action Management: PABX Agent and PABX Simulator

This agent supports the following functions:

1. MakeCall (using the makecall message),
2. ConferenceCall (using the conference message),
3. CallTransfer (using the activate-forwarding message),
4. CancelCallTransfer (using the deactivate-forwarding message).

For a complete description of this agent see chapters 2 and 3 of this document.

Figure 4.2: A telephone conversation using the Terminal Simulator agent

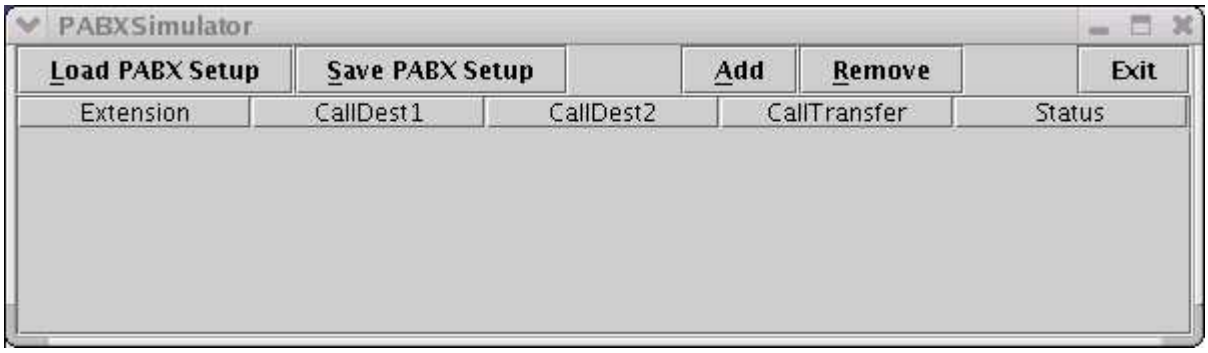

Figure 4.3: PABX Simulator agent (GUI)

PABX Simulator

The simulated PABX has been implemented as a table where each extension controlled by the PABX is represented in a different row.

As figure 4.3 shows, each extension has a unique identification (consisting of five digits).

The PABX Simulator allows the simulation of all the functions of the real PABX used in the automatic telephone operator scenario of Siridus:

1. MakeCall: an extension may initiate a phone call with another extension or with an external phone number (of nine digits). In the first case, both the calling and the destination extensions are marked as occupied in the status field.
2. ConferenceCall: an extension initiates a conference call with two internal extensions, two external phone numbers or an extension and an external phone number. In this case, all the extensions implied in the conference call are marked as occupied.
3. CallTransfer: an extension is associated to another extension, so all the incoming calls to the first one are forwarded to the second one.
4. CancelCallTransfer: the forwarding mechanism is deactivated.

Figure 4.4 shows an example where:

- Extension 67506 is making a conference call with extensions 67512 and 67536.
- Extension 67516 is calling extension 65504.
- Extension 67510 is currently forwarded to extension 67555

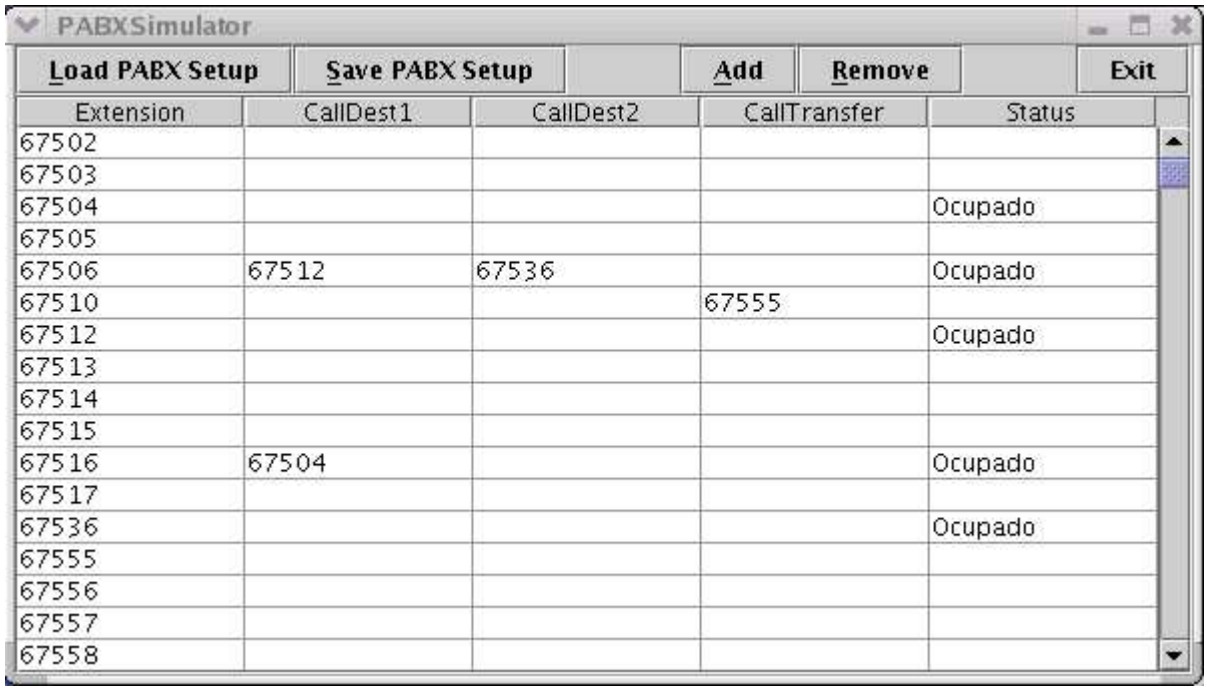

Figure 4.4: PABX Simulator agent (GUI)

4.2.3 Knowledge Management: DataBase Agent and Corporate Directory Simulator

Basically, this agent stores all the information about the corporate directory of the institution where the PABX is installed.

The functions supported by the DB agent are:

1. LookUp: using the select message this agent looks for all the records in the database that match a pattern search.
2. Get: by means of the get message, the DB agent can return to the Dialogue Management agent the set of records obtained after the lookup operation.
3. Update: the data base must be updated incorporating information about the last call and the transfer status of the different extensions.

Corporate Directory Simulator

The initial specification and design of the DataBase agent was quite simple. During the implementation of the whole environment we have found several functions that should be included in a real automatic telephone scenario.

Figure 4.5: Corporate Directory Simulator agent (GUI)

1. **Destination (name) resolution**: The first one is the incorporation of a more sophisticated destination resolution algorithm. People tends to use partial names (just the first or last name), nicknames or even short forms of the names. This is even more complicated in Spanish, as people have two last names, and it is very common to have and use double first names.

2. **Personal directory**: Second, in addition to the corporate directory, people like usually to use personal directories with personal directory entries ("call home", "call my secretary", and so on).

3. **User profile**: The history of the interaction with the automatic telephone system may improve the naturalness of the interaction. Usually people need to call or transfer their calls to the same destination, or even have more or less fixed patterns of use. This way, it is unnatural to disambiguate the expression "call Peter" each time the same user tries it. To be collaborative, the system should access the dialogue history and suggest if the user wants to call again the "same Peter" as the last time.

In a first attempt to cover these new functions we have implemented a Corporate Directory Simulator (figure 4.5).

The destination resolution algorithm (specially for names) has been included as part of the Knowledge Manager. It defines a set of rules for the disambiguation of destination by means of names.

The use of a Personal Directory is allowed by the Corporate Directory agent as it can be seen in figure 4.5. Finally, this agent stores the registry of previous calls for each user. This way, this information is available to improve the destination resolution strategy.

4.3 User's Interface

4.3.1 Introduction

In this section we will describe the system interface with the final user. We will focus on aspects such as the specific domain, the language register and other aspects of the interface design.

It is important to note the relevance of a good interface in spoken dialogue systems. Robust and flexible systems with great functionality may be eclipsed by robotic, impersonal, inappropriate or even dull interfaces.

One of the goals of these systems is to establish a natural, comfortable and efficient line of communication between human speakers and automated systems. This can only be achieved if the automated systems can not only understand human speakers with a reasonable degree of confidence but also show it. The users need to be certain that their instructions are being understood and the service will indeed be performed as indicated. This is particularly important in applications where sensitive information is handled.

This is however not enough. Beyond conscientiousness and accuracy, the system needs to take into account other aspects of human spoken communication such as language, register, terminology, intonation, etc. Human factors are important in automated systems.

The implementation described here is an Automated Phone Operator. We have implemented it with the company directory of Telefonica I+D, so the interface is designed for a company environment with multiple users and a common directory.

4.3.2 Language, Register and Terminlogy

Given a certain domain, it is important to take into account not only the vocabulary relevant to that domain, but also the type of user that will be using the application.

In the case of language, register and terminology, the system must be able to handle the speaker's expressions as well as being capable of expressing itself appropriately. The system

utterances must not be excessively complex. As a matter of fact, the utterances should be simple and easy to understand in order to ease the user's task. As with excesses in general, oversimplicity is not advisable either.

It is also important to be ensure consistency between input and output, that is, the system must never use terminology or expressions that would not be able to handle at recognition level. It is a fact that human speakers often use the same expressions to refer to elements that have already come up in the conversation.

In our case, given that the system domain is not highly technical and the expected user is the average speaker, the system interface has been designed in Spanish, with polite–friendly personality and neutral register. This does not mean that the system expects users to address it using the same register only.

The system is flexible enough to handle direct commands, as well as indirect expressions, using formal or informal forms (Spanish has both formal and informal pronoun and verb forms).

Utterances such as the following would be equally understood by the system:

- Haz una llamada a Juan Alonso.
  (Make a phonecall to Juan Alonso)
- Llame a Juan Alonso.
  (Call Juan Alonso –**formal**–*)*
- Me gustaria que llamaras a Juan Alonso por favor.
  (I would like you to call Juan Alonso please)

4.3.3 Intonation

Another important aspect of the user interface is the intonation. Unfortunately, no prosody recognizer was available for this project and therefore our system does not take it into account in the recognition process.

Nonetheless, natural synthetic speech is an important factor in a spoken dialogue system. Inappropriate intonation might not only jeopardize the communication by misleading the user, but would also make the conversation unnatural. If the system can not deliver its messages accurately and naturally, the user's confidence in the system efficacy may be undermined.

In addition to this, the system must be able to convey naturally all the information necessary to continue with the conversation. Part of this information is not always overtly alluded to, but conveyed throughout different channels. Intonation is one of those channels.

The ability to generate the appropriate intonation given an utterance and the context in

which it must be generated, depends not only on the system architecture itself , but also on the synthesizer capabilities. Most synthesizers generate their intonation automatically based on a prosodically labeled corpus. Some of them also include some additional heuristics that help the synthesizer decide on ambiguous situations; others customize their output to the current context throughout configuration files.

The OAA–based demo for the Automatic Telephone Operator System (O-TeleDelfos2, as described in section 2 of this chapter) incorporates Festival, a synthesizer developed at the University of Edinburgh. This particular synthesizer is capable of processing prosodically labeled text. There are different speech synthesis markup standards, but it is SABLE that has been implemented in Festival.

The infrastructure of our system permits the use of dialogue information to determine, not only what dialogue move, response or wording would be more appropriate, but what intonation would suit the situation better, given the current dialogue history, expectations and context.

4.3.4 Personality

Most systems opt for a neutral polite–friendly personality. This does not need to be the case however. Depending on the purpose of the application, different personalities can be designed. It is consistency throughout the interface that is important.

As mentioned above, our system interface has been designed as a polite–friendly agent.

Given the system architecture, it is fairly easy to change its personality at language level. It would be necessary to generate an additional prompt file that may have a different register, more concise wording, wordier explanations, etc. Personality changes related to system initiative or collaborativeness would require additional work in the dialogue manager.

4.3.5 Voice

The voice chosen for a particular application is often relevant. Although at a basic level the chosen voice should simply be intelligible and pleasant, in certain contexts it may be preferable to select a male voice rather than a female voice, or the other way around, or higher or lower pitch within the gender.

Voice is directly related to personality, so the choice of voice will also have an impact on the overall user impression.

4.3.6 User Confidence

One of our goals during the system design was to ensure that the final user would feel confident when interacting with the system. To achieve this goal, several strategies have been implemented.

One of these strategies consists of consistently confirming the commands and their parameters before executing any actions with unrecoverable consequences. It is however important to find a balance between the need to confirm and the dullness of confirming too often.

Ideally, the confirmation strategy should not only be related to the importance of the consequences of executing the command, but also to the recognition confidence level. Unfortunately, the recognizer chosen for the current system implementation does not provide the recognition confidence score, so our current strategy is based primarily on relevance.

4.3.7 Output Variety

Some systems, although efficient and robust, have a unique set of prompts per dialogue phase, which ends up making them monotonous and predictable.

In order to avoid the monotony of hearing the same output over and over, the system has been designed with multiple prompts per situation. The prompts are randomized so that the conversation is never exactly the same.

This is one of the steps we have taken towards making the system as natural and human–like as possible.

4.4 Natural Language Understanding and Semantic Interpretation

4.4.1 Introduction

Obtaining the correct semantic interpretation of the user input is essential in a spoken dialogue system. It is for this reason that we have paid special attention to the system coverage. The system can understand the following commands:

1. Domain specific commands:
   (a) Make a phone call.
   (b) Redial.

(c) Last dialed number.
(d) Conference call.
(e) Call Transfer.
(f) Cancel the call transfer.
(g) Current transfer destination.
(h) Look up an email or an office number.
(i) Help.
(j) List.
(k) Repeat.

2. General commands:

(a) Quit
(b) Greet
(c) Retry

Each of the items in the list above simply represents the general functionality covered by the system, i.e., different forms of the commands are equally acceptable. Examples and more details about the system coverage will be given in subsequent sections.

The two main domain–specific system components at this level are the lexicon and the grammar.

The details about the syntax used for the lexicon and grammar are given in Deliverable 3.2, chapter 5. To outline the main characteristics, the lexicon syntax is Mph-Vtree [Quesada & Amores 1995], which is a powerful and efficient combination of systems designed as a specification environment for unification grammar lexicons. The analysis grammar consists of three components: a series of configuration parameters, a set of context–free productions and a set of functional equations that may or not be contained in the productions.

Next, we will focus on the details of the Automated Phone Operator implementation.

4.4.2 Lexicon

The lexicon generated for the current implementation of the Automated Phone Operator covers all the commands listed above, as well as most of their synonyms, related words, all the first and last names in Telefonica's directory, numbers, and other particles such as temporal markers, auxiliaries, etc.

The lexicon is pseudo–semantically structured, i.e., the chosen categories do not correspond to morphological or syntactic categories, but rather to the semantic functionality of the items.

Its structure is however not entirely semantic, since some elements semantically identical have been classified under different categories for convenience at grammar level.

Some of the categories include additional information that the dialogue manager and the dialogue move selector will use to decide on the next move. In the case of temporal markers for example, items incorporate the feature `Priority` and a certain value. This feature and its value will be inherited by the command immediately preceded by the temporal marker. This feature will determine the order in which a series of commands within the same utterance will be performed:

```
User:    Llama a luis    pero primero    desvia mis llamadas.
         Command1        Temporal        Command2
                         Marker
```

`Command2` is immediately preceded by the temporal marker and therefore inherits its priority value, which in this case is `10`. Since `Command1` is not preceded by any marker, its priority defaults to 0. Therefore, `Command2` will be executed first.

4.4.3 Grammar

Grammar and lexicon are intimately interrelated. Therefore, the lexicon structure will determine to a certain degree the expressive power of the grammar.

The grammar covers all the commands listed above. We must stress that these merely represent instances of the coverage of each of the cases. The example below illustrate the multiple possibilities covered for just some of the cases of the `MakeCall` function:

```
(M001 : MakeCall -> LMakeCall)
{ @up = @self-1; }


(M002 : MakeCall -> LDo LCall)
{ @up = @self-1; }


(M003 : MakeCall -> LMakeCall Destination)
{ @up = @self-1;
  @if (@self-2.Name) @then {
  @up.Dest = @self-2.Name; }
  @else @if (@self-2.Extension) @then {
  @up.Dest = @self-2.Extension; }
  @else @if (@self-2.PhoneNumber) @then {
  @up.Dest = @self-2.PhoneNumber; }
```

```
}
```

These rules above are just three out of a total of eight rules contained in the grammar to cover the commands related to making a phone call. The utterances below are examples of the great variety of utterances covered by these rules.

- Llama
- Llama a Luis
- Haz una llamada
- Ponme con la extensión 12345
- Quiero hacer una llamada al número 123456789
- Me gustaría hablar con Antonio
- Pásame a luis, por favor
- Márqueme el número 123456789
- Etc.

The purpose of this redundancy of equivalent utterances is to ensure that the users will enjoy the maximum flexibility when talking to the system. This will spare them from having to remember fixed commands and will allow them to speak naturally.

This grammar has been designed to pass up all relevant features contained in the lexical categories, such as Content (CONT), Priority (Priority), Literal (LIT), etc. These are nonetheless not the only features handled by the grammar. There are also features whose values depend on the co–occurrence of lexical categories. Note for instance the case of rules such as:

```
(M007 : MakeCall -> LRepeat LMakeCall Destination)
{ @up = @self-2;
  @if (@self-3.Name) @then {
  @up.Dest = @self-3.Name;
  @up.Dest.Rep = "True"; }
  @else @if (@self-3.Extension) @then {
  @up.Dest = @self-3.Extension; }
  @else @if (@self-3.PhoneNumber) @then {
  @up.Dest = @self-3.PhoneNumber; }
}
```

which covers cases such as:

- Vuelve a llamar a Pedro
  (Call Pedro again)
- Vuelve a ponerme con el número 123456789
  (Put me through with the number 123456789 again)
- Etc.

In the case of utterances where there is a name, it may be necessary to disambiguate the destination of the call. The grammar then passes up the feature `Rep` (Repetition), if the utterance itself informs the system of the pre–occurrence of the destination within the dialogue history ("Vuelve a ..." "Again ..."). This information may be used later to resolve the ambiguity without going through the disambiguation process overtly.

4.5 Dialogue Manager

Our approach to dialogue focuses on the following objectives:

1. At a formal level, it shares the motivation behind collaborative plans —help subdialogues, confirmations, flexible patterns, etc.— and information state revision — dialogue management, dialogue history, etc—.
2. It uses a declarative model to specify the dialogue. The goal is to overcome the limitations of models based on dialogue grammars, state transition networks or plan recognition such us flexibility, naturalness, error recovery complex mechanisms, etc.
3. The dialogue is considered a cooperative phenomenon directed by the semantic representation of the information provided by the user, and the expectations generated by the system itself.

This approach integrates a semantic representation formalism as well as a strategy for the manipulation of pragmatic information [Scott & Kamp 1995, Zue & Glass 200000, Kamp & Reyle 1993, Barwise & Perry 1983]. This approach has been implemented before in another European project: Dhomme (IST-2000-26280) [Quesada *et al* 2001].

4.5.1 Dialogue Modeling and Management Components

There are four basic levels or modules:

1. **Specification Level**

It specifies at high level the systems characteristics and functions. Its basic structure is the *Dialogue Rule*. It is at this level that we will discuss dialogue management in detail in subsequent sections.

2. **User Interface Level**

   This module manages the interaction between the user and the system. It consist of several modules: Speech Recognizer, Dialogue Move Input Pool, Dialogue Move Selector, Natural Language Generator and Speech Synthesizer.

   Its basic structure is the *Dialogue Move*. Both the user's utterances and the system's responses will be semantically represented as one or more dialogue moves, which will in turn be processed by the NL Generator and the synthesizer.

   This level consolidates and organizes the dialogue moves: repetitions, intervention order, sub–dialogues, critical interruptions, etc. At this level, advanced systems incorporate semantic and pragmatic information to improve speech recognition and synthesis.

3. **Representation Level**

   It manages the dialogue history and its basic structure is the *Information State*. Each instance of the Information State is stored in a repository which is accessible by the rest of the system modules. It allows for anaphor resolution and other dialogue related phenomenon, as well as the resolution of expectations and other dialogue–management related events.

4. **Execution Level**

   The Dialogue Manager (DM) is in charge of this level. Its basic structure is the *Dialogue Phase*. The DM receives the Dialogue Rules from the Specification Level, and the Dialogue Moves from the User Interface Level. It also has access to the information at the Representation level. It makes use of all this information to generate an output through the User Interface Level, resolve anaphors or expectations, add new Information States to the Dialogue History execute whatever actions emerge from the resolution of the Information States, etc.

   A Dialogue Phase is the instantiation of a Dialogue Rule for a Dialogue Move, which generates an Information State.

$$\text{DPhase} = \text{DRule} + \text{DMove} + \text{IState}$$

   This level may also incorporate other modules specialized in Knowledge or Action Management, that is, modules that allow for the resolution of domain specific references or actions. In the current implementations, both the Knowledge Manager and the Action manager have been incorporated and are discussed in section 2 of this chapter.

4.5.2 Language Modeling based on Knowledge, Expectations and Actions: Specification Level

Knowledge–based Modeling

One of the main triggering mechanisms to activate the Dialogue Rules in the Specification Level, is the information received at the Execution Level throughout the User Interface Level. The representation of this information is based on complex feature structures and the Dialogue Rule selection is based on the unification of the TriggeringConditions field within the Dialogue Rules.

For the case in which more than one Dialogue Rule is triggered by the same conditions, the system incorporates `PriorityLevel`, which would disambiguate between rules.

Note for example the following sentence:

- **In[1]**: Querría hacer una llamada telefónica.

According to the semantic representation model for natural language command–based dialogue systems [Amores & Quesada 2000], we could represent sentence [1] as:

```
DMOVE: specifyCommand
TYPE:  MakeCall
ARGS:  [Dest]
CONT:
```

Given a Dialogue System with the following rules:

```
(RuleID: MAKECALL;
 PriorityLevel:    15;
 TriggeringCondition:
     (DMOVE:specifyCommand,TYPE:MakeCall);
 ...)


(RuleID: REDIAL;
 PriorityLevel:    15;
 TriggeringCondition:
     (DMOVE:specifyCommand,TYPE:ReDial);
 ...)

(RuleID: SAFETYNET;
```

```
PriorityLevel:    10;
TriggeringCondition: ();
...)
```

We can see that both rules MAKECALL and SAFETYNET would be activated by the previous feature matrix. However, the PriorityLevel of the TELEPHONECALL rule is higher than that of the SAFETYNET rule. The latter is one of the mechanisms of the dialogue manager to ensure robustness, i.e., this rule will unify with any input rejected by all other rules preventing the system from crashing.

The process of activation of a dialogue rule (DRule) generates an instance of the same rule *dialogue phase (DPhase)* at the Execution level, which is associated with the information state rule (IState), which in turn contains the dialogue move (DMove) which activated the rule in the first place.

4.5.3 Expectations Control and Management

Similarly to how the user's initiative is captured and manipulated by means of knowledge–based modeling, the systems initiative is modeled throughout expectations.

The feature ARGS specifies the set of attributes that a state must contain in order to be coherent. The model used to specify multiple models of subcategorised attributes is of the type *disjunction of conjunctions*. The coherence principle states that for every value of ARGS, there must exist an instantiated feature with the same name in the structure. The recursive application of this principle implies that the coherence of the structure is directly dependent on the coherence of its parts.

Any incoherent feature in an information state generates an expectation with the same name in the corresponding dialogue phase.

In the previous example, ARGS includes the Dest parameter, which does not exist in the information state. The `ActionsExpectations` field in each dialogue rule indicates which actions must be executed in case certain combinations of expectations were activated.

```
(StateID: MAKECALL;
 PriorityLevel:    15;
 TriggeringCondition:
     (DMOVE:specifyCommand,TYPE:MakeCall);
 DeclareExpectations: {
     Dest <=
         (DMOVE:specifyParameter,
          TYPE:Extension|Name|PhoneNumber); }
 ActionsExpectations: {
     [Dest] => { ExecuteDMFunction(MakeCallDest); }
```

```
}
...)
```

In this case, when the `Dest` expectation is active, the system is instructed to execute the `MakeCallDest` function. `ExecuteDMFunction()` is a macro–function whose content is specified in the function file.

The `DeclareExpectations` field indicates how to solve expectations. There are two mechanisms:

1. According to a pre–specified feature model, the dialogue move must complete the feature generated by the expectations. It is an application of the resolution of the triggering conditions.
2. A dialogue rule may be specified as a condition to solve a given expectation. This allows for the case in which if in a subsequent dialogue state this rule is generated and proven coherent, the corresponding information state will unify with the information state that generated the expectation. This will originate a new dialogue phase associated to the original rule with an updated information state.

In the previous example, the MAKECALL rule specifies in the `DeclareExpectations` field that the `Dest` feature must be solved throughout a rule that can unify with

```
(DMOVE:specifyParameter,TYPE:Extension|Name|PhoneNumber)
```

which is any specifyParameter DMOVE of TYPE Extension, Name or PhoneNumber.

Given the following dialogue:

- **Out**[**1**]: A quién quiere llamar?.
- **In**[**1**]: Al número 123456789.

The system will generate the following dialogue move as a result of the user's response:

```
DMOVE: specifyCommand
TYPE:  MakeCall
ARGS:  []
CONT:  123456789
```

This dialogue move is captured by the PHONENUMBER rule:

```
(RuleID:    PHONENUMBER;
 PriorityLevel:    20;
 TriggeringCondition:
     (DMOVE:specifyParameter,TYPE:PhoneNumber);
)
```

Since ARGS is empty, this structure is coherent; therefore this phase has been completed and the information state of this phase becomes the `Dest` feature of the information state in the previous phase. This generates a new dialogue phase associated to the same rule with a new information state:

```
DMOVE: specifyCommand
TYPE:  TelephoneCall
ARGS:  [Dest]
CONT:  -
Dest:
        DMOVE: specifyParameter
        TYPE:  PhoneNumber
        ARGS:  []
        CONT:  123456789
```

There are other types of expectations. While the field `DeclareExpectations` indicates how to solve the expectations predefined in the ARGS field of the DTAC, the field `SetExpectations` permits to declare expectations at a different level. This field is usually used to force the confirmation of a command before its execution:

- **In**[**1**]: Llama a Luis Rodríguez.
  (Call Luis Rodríguez)
- **Out**[**2**]: Quiere llamar a Luis Rodríguez. Es correcto?
  (You want to call Luis Rodríguez. Is that correct?)
- **In**[**2**]: Sí.
  (Yes)
- **Out**[**3**]: Le paso con Luis Rodríguez. *(I'll put you through with Luis Rodríguez)*

The way to specify these expectations is identical to that of the `DeclareExpectations` field.

```
(   RuleID:    MAKECALL;
    PriorityLevel:    15;
    TriggeringCondition:
        (DMOVE:specifyCommand,TYPE:MakeCall);
```

```
DeclareExpectations: {
    Dest <=
        (DMOVE:specifyParameter,
         TYPE:Extension|Name|PhoneNumber); }
SetExpectations: {
    Confirm <= (DMOVE:answerYN); }
ActionsExpectations: {
    [Dest] => { ExecuteDMFunction(MakeCallDest); }
    [Confirm] => {
        ExecuteDMFunction(MakeCallDisam);
        @if (@is-MAKECALL.Dest.DestRes.Quantity == 1) @then {
            ExecuteDMFunction(MakeCallConfirm);
        }  @else {
            CloseState();
        }
    }
}
...)
```

There is still another field to indicate what the system should do given a certain combination of expectations. The `RecoveryActions` field is similar to the `ActionsExpectations` field since it usually considers the same cases. This is however a especial field to resume the execution of an action that was indicated some time before the execution of the last command. We are referring to cases like the following one:

- **In[1]**: Desvía mis llamadas a Juan Gómez y llama a Luis Rodríguez.
  (Transfer my calls to Juan Gómez and call Luis Rodríguez)
- **Out[2]**: Quiere desviar sus llamadas a Juan Gómez Correcto?
  (You want to transfer your calls to Juan Gómez. Is that correct?)
- **In[3]**: Sí.
  (Yes)
- **Out[3]**: A partir de este momento todas sus llamadas serán desviadas a Juan Gómez.
  (From this moment on all your calls will be transferred to Juan Gómez.)
- **Out[4]**: Previamente indicó que quería llamar a Luis Rodríguez Desea realizar esa llamada a continuación?
  (You previously mentioned you wanted to call Luis Rodríguez. Would you like to make that call now?)

This strategy makes the system sound more intelligent, since it remembers the commands that have not been executed yet, and in addition to this, it retakes the conversation very smoothly. Here are for instance the RecoveryActions of MakeCall:

```
RecoveryActions: {
    [Dest] => {
            ExecuteDMFunction(RAMakeCallDest);
    }
    [Confirm] => {
            ExecuteDMFunction(RAMakeCallConfirm);
    }
}
```

4.5.4 Action Execution Control

In this context, we will consider *Actions* all those functions that the dialogue system executes directly, or throughout modules, or external resources or agents. The system includes a basic set of internal and external actions.

Internal Actions

- **UserPrompt**: It generates and sends an output directly to the Speech Synthesizer and through to the user.

  ```
  UserPrompt(
      @firstMessage(
          "Que funcion desea realizar?",50,
          "Por favor, indique la funcion que desee realizar.",50),
      @onInputError(
          "No he recibido ninguna respuesta. Por favor, indique
                 la funcion que desea ejecutar.",
          "Sigo sin recibir ninguna respuesta. Por favor
                 indique alguno de los comandos disponibles o
                 solicite ayuda."));
  ```

  The UserPrompt usually requests some information from the user and its execution will stop the system until user input is provided.

  As it can see in the structure above, the UserPrompt may consist of @firstMessage and @onInputError. The first one would be the wording that would be sent to the synthesizer when the prompt is played in normal circumstances. The second one contains the prompts that should be played if the system has not understood the input given when the @firstMessage prompt was played, or no input at all was given.

  The @firstMessage structure also incorporates the ability to include up to 10 different prompts that will be selected at random according to the value immediately following them (50 in this case). The second prompt contained in the @onInputError structure will be played on subsequent error inputs.

- **UserMessage**: As in the case of the UserPrompt, the UserMessage generates and sends an output directly to the Speech Synthesizer and through to the user. The only difference is that the former expects and waits for an answer from the user, whereas the latter does not. The UserMessage does not incorporate the @onInputError structure.

```
UserMessage(
    @firstMessage(
        "Hola, le atiende su operador telefonico
            automatico.",50,
        "Esta usted al habla con su operador telefonico
            automatico.",50));
```

  For convenience, all UserPrompts and UserMessages have been grouped into a function file that complements the dialogue manager. This file contains not only all prompts, but also some macro–functions, that is, sets of functions that are used together in several places. This division provides two advantages mainly: on the one hand, the dialogue manager is free from much of its reiteration of functions throughout it; on the other hand, the dialogue manager is also free from the specific wording of the interface, and in addition to this, the interface can be easily modified without modifying the dialogue manager itself. This advantage for example would facilitate the creation of several personalities at language level that could be loaded dynamically.

- **UserGeneration**: It uses an information state as input to generate an output by means of a natural language generation intermediate module.

- **ActivateRule**: This mechanism establishes a strong link between the dialogue rules, different from the knowledge–based model previously presented. It allows for the direct activation of a dialogue rule and specific phenomena modeling such as dialogue grammars or finite state network strategies.

- **CreateInformationState**: It creates an information state associated to the dialogue phase where the corresponding rule has been activated. It complements the ActivateRule function and allows for the resolution of phenomena such as Task Accommodation, by creating an initial information state as default model.

```
( StateID:    FUNCTION;
  PriorityLevel:    10;
  PreActions: {
      CreateInformationState (
          (DMOVE: DM,
           TYPE : Operator,
           ARGS : [specifyCommand]) );
  }
  ...)
```

- **AnaphorResolution**: The system can perform anaphor resolution if the semantic representation includes anaphoric marks. It is a unification–based strategy that makes use of the information states associated to previous phases.

- **ActivateBackgroundState**: This is one of the functions created to make the system more flexible in terms of the user's needs. This function activates a dialogue rule (in this case DEFAULT) which enables the system to accommodate to an incomplete command, even at the very beginning of the dialogue.

```
( RuleID:    DEFAULT;
  PriorityLevel:    10;
  PreActions: {
      CreateInformationState (
          (DMOVE: specifyCommand,
           TYPE : MakeCall,
           ARGS : [Dest],
           default: yes ) );
  }
  DeclareExpectations: {
      Dest <= (DMOVE:specifyParameter,
               TYPE:Extension|Name|PhoneNumber);
  }
  PostActions: {
      ActivateState(MAKECALL,@is-DEFAULT);
  }
)
```

  In this domain, our default dialogue rule creates a new MakeCall information state. This IS will be in the background and will take care of situations like the one below:

  - **Out**[**1**]: Hola, le atiende su operador telefónico automático.
    (Hello, your Automated Phone Operator speaking.)
  - **In**[**1**]: Con Juan Antonio Pérez por favor.
    (With Juan Antonio Pérez please.)
  - **Out**[**2**]: Desea hacer una llamada a Juan Antonio Pérez. Correcto?
    (You want to make a phone call to Juan Antonio Pérez. Correct?)

  No command has been uttered: the user has only given the system a parameter. Given the domain however, it is not difficult to guess that the user wants to make a telephone call and is giving the system the destination of that call. It is now that the default information state in the background will come into play.

- **LastUserPrompt**: This function allows the system to recover the last prompt played. This is extremely useful for the user, since they will be able to ask the system to repeat itself at will and therefore facilitate the communication process.

4.5.5 External Actions

- **DestinationResolution**: In the current domain, it is possible to receive ambiguous destinations when the parameter in question is a name.

– **In[1]**: Llama a Luis.
(Call Luis.)

Given that there is a directory of considerable size, there might be more than one *Luis* in it. All destinations of type Name must therefore be disambiguated before continuing with the dialogue. Given the situation presented above, the system would resort to checking the directory for all possible matches, and ask for a more specific choice:

– **Out[1]**: Esta usted al habla con su operador telefonico automatico.
– **Out[2]**: Por favor, indique la función que desee realizar.
– **In[1]**: llama a luis.
– **Out[3]**: He encontrado 82 entradas para LUIS en el directorio. Entre ellas: JOSE LUIS ANTA DE LA IGLESIA, LUIS ANTON GALINDO, LUIS MIGUEL APIO RECIO, etc.. Por favor, indique información adicional.

- **ListOfNames**: This feature enables the system to make use of all the names matching the search. These names will be therefore subject to be listed on the user.s request.
- **LastCallResolution**: This is a domain specific function that reads the directory and finds out the destination of the last user's call. The directory is updated every time a call is made.
- **CurrentCallTransferResolution**: This is another domain specific function that reads the directory and finds out where the user's call are currently being diverted. The directory is updated when the user activates, modifies or cancels a call transfer.
- **ExecuteAction**: The dialogue system can make use of a domain–specific Action Manager to execute the action associated to the semantic representation of the corresponding information state.

4.5.6 Pre and Post Actions

The specification level allows for two action execution instances:

- **PreActions**. They are executed as the rule is activated by TriggeringConditions or ActivateRule, before the coherence analysis and expectations control phase.
- **PostActions**. They are executed once the coherence analysis has been completed, i.e., all the expectations have been solved.

In the following example the LastCallResolution has been placed in the PreActions, and the actual execution of the call occurs in the PostActions, once all the expectations have been fulfilled. In this case, the only expectation to fulfill is the Confirmation.

```
( RuleID: REDIAL;
  PriorityLevel: 15;
  TriggeringCondition:
      (DMOVE:specifyCommand,TYPE:ReDial);
  PreActions: {
      LastCallResolution("Dest");
  }
  SetExpectations: {
      Confirm <= (DMOVE:answerYN);
  }
  ActionsExpectations: {
      [Confirm] => {
              ExecuteDMFunction(ReDialConfirm);
  }
  RecoveryActions: {
      [Confirm] => {
              ExecuteDMFunction(RAReDialConfirm);
        }
  }
  PostActions: {
      @if ((@is-REDIAL.Dest) &&
           (@is-REDIAL.Confirm.TYPE == "YES")) @then {
              ActivateState(MAKECALL,@is-REDIAL);
      } @else @if (!@is-REDIAL.Dest) @then {
              ExecuteDMFunction(ReDialNoDest);
              ActivateState(MAKECALL,@is-REDIAL);
      }
  }
)
```

4.6 Dialogue Move Selection

This section describes the characteristics of the Dialogue Move Selector implemented by the University of Seville.

The initial motivation for this module was analysed in chapter 4 (*Dialogue Management Agent Design*) of Deliverable 3.2 (*Design of a Natural Command Language Dialogue System*).

As it was described in that deliverable, there is an important barrier of synchronism in the interface between the Natural Language Understanding (NLU) and the Dialogue Manager (DM) modules of the Dialogue Management agent.

The reason is that the user may specify multiple dialogue moves on a single utterance or the NLU module may misrecognise or misunderstand the user and generate multiple dialogue

moves. On the other hand, it is very important for the Dialogue Manager to process just one dialogue move at each time. On the contrary, the dialogue will become very obscure and unnatural.

So, it is reasonable to think of an intermediate store of dialogue moves, which we have called Dialogue Move Input Pool. This store may be implemented as a queue or a stack. Nevertheless these trivial implementations do not solve all the characteristics expected at the communication interface between the NLU and DM modules.

This section first concentrates on the study of the communication between these two modules, analysing the queue and stack approaches. The study of several phenomena allows the proposal of a comprehensive and intelligent expectation–driven dialogue move selection strategy. Different phenomera which require the use of this strategy are discussed:

- Subdialogues
- Disconnected partial dialogue moves
- Temporal priority marks
- Expectation–dependent semantic interpretation

Next, the architecture of the Dialogue Move Selector is described. Finally, five general strategies currently implemented as part of the Dialogue Move Selector are presented:

- Dialogue moves fusion: unification–based equality or inclusion
- Temporal priority marks
- Recovery of dialogue moves
- Expectation–driven dialogue move selection

4.6.1 The communication between the Natural Language Understanding and the Dialogue Manager modules

The kernel of the Dialogue Management Agent is based on two main modules. First, the Natural Language Understanding (NLU) module receives the user's utterance and produces a sequence of semantic representations of the dialogue moves contained. This formal representation is based on the DTAC protocol. Second, the Dialogue Manager (DM) module takes one by one the DTACs generated by the Natural Language Understanding module and applies the Update Rules obtaining a new information state.

The communication between these two components conditions the capabilities of the system.

The queue approach

A first naïve approach would implement this communication as a sequential flow of dialogue moves from the Natural Language Understanding module to the Dialogue Manager. This way, this communication may be formally defined and implemented as a queue of dialogue moves.

For instance, in the following dialogue fragment a queue–based strategy would be enough:

- **S(1)**: Bienvenido al Operador Telefónico Automático
  (*Welcome to the Telephone Operator System ...*)
- **U(1)**: Me gustaría hacer una llamada.
  (*I would like to make a phone call.*)
- **S(2)**: Por favor, indique el nombre, extensión o número de teléfono.
  (*Please specify the name, extension or phone number ...*)
- **U(2)**: Extensión 2 1 2 1 2.
  (*Extension 2 1 2 1 2.*)

At **U(1)** the NLU module generates the following DTAC structure:

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & MakeCall \\ ARGS & : & [Dest] \\ CONT & : & \end{bmatrix}$$

This DTAC is captured by the MAKECALL update rule of the DM:

```
( RuleID:    MAKECALL;
    PriorityLevel:    15;
    TriggeringCondition:
        (DMOVE:specifyCommand,TYPE:MakeCall);
    DeclareExpectations: {
        Dest <= (DMOVE:specifyParameter,
                 TYPE:Extension|Name|PhoneNumber);
    }
    SetExpectations: {
        Confirm <= (DMOVE:answerYN);
    }
    ActionsExpectations: {
        [Dest] => {
```

```
                ExecuteDMFunction(MakeCallDest);
        }
    ....
```

Basically, once this rule is triggered (as its triggering conditions unify with the DTAC) it creates a new information state. At the beginning, this information state contains just the information of the DTAC.

Taking into account that this information state requires a new Dest argument in order to be complete, the ActionsExpectation section of the dialogue rule is used. In this case, the dialogue manager function MakeCallDest is executed, which in turn generates S(2), that is, the reply of the system to the first user's utterance.

The next user's utterance U(2) creates the following DTAC:

$$\begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & Extension \\ ARGS & : & \\ CONT & : & 21212 \end{bmatrix}$$

The formal representation of this new dialogue move can be captured by the Expectation strategy of the MAKECALL dialogue rule:

```
( RuleID:    MAKECALL;
    ...
    DeclareExpectations: {
        Dest <= (DMOVE:specifyParameter,
                 TYPE:Extension|Name|PhoneNumber);
    }
    ....
```

As a result, the information state is updated, obtaining the following:

$$\begin{bmatrix} DMOVE & : & specifyCommand & \\ TYPE & : & MakeCall & \\ ARGS & : & [Dest] & \\ CONT & : & & \\ Dest & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & Extension \\ ARGS & : & \\ CONT & : & 21212 \end{bmatrix} & \end{bmatrix}$$

This approach will work correctly for very simple and sequential dialogues, where each user's utterance is limited to just one dialogue move, and where the dialogue move of each new utterance is closely related to the information state of the dialogue manager.

Multiple dialogue moves

In natural and flexible dialogues the first problem that appears with the queue-based approach has to do with utterances containing multiple dialogue moves.

In the following example, the user asks the system to perform a couple of actions:

- **S(1)**: Bienvenido al Operador Telefónico Automático
  (*Welcome to the Telephone Operator System ...*)
- **U(1)**: Me gustaría hacer una llamada y activar los desvíos.
  (*I would like to make a phone call and activate the transfer mode.*)

Sentence **U(1)** generates two DTACs:

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & MakeCall \\ ARGS & : & [Dest] \\ CONT & : & \end{bmatrix}$$

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & CallTransfer \\ ARGS & : & [Dest] \\ CONT & : & \end{bmatrix}$$

The first DTAC, will trigger again the MAKECALL dialogue rule, which will ask the user about the destination of the call. When the user replies to the system:

- **S(2)**: Por favor, indique el nombre, extensión o número de teléfono.
  (*Please specify the name, extension or phone number ...*)
- **U(2)**: Pedro Camacho.
  (*Pedro Camacho.*)

The new DTAC will be included in the queue after the CallTransfer DTAC remaining from the user's last input.

Now the system will recover the CallTransfer DTAC (the first in the queue), which will trigger the CALLTRANSFER update rule, generating a very confusing dialogue:

- **S(3)**: A qué destino desea desviar sus llamadas?
  (*Where would you like to transfer your calls?*)
- **U(3)**: ???

The stack approach

A second approach to this problem is based on the use of a stack instead of a queue to manage the communication between the NLU and the DM modules.

This strategy will solve the previous example. In fact, the stack-based strategy will store first the MakeCall DTAC and next the CallTransfer over it, so the DM will get the CallTransfer first:

- **S(1)**: Bienvenido al Operador Telefónico Automático
  (*Welcome to the Telephone Operator System ...*)
- **U(1)**: Me gustaría hacer una llamada y activar los desvíos.
  (*I would like to make a phone call and activate the transfer mode.*)
- **S(2)**: ¿A qué destino desea desviar sus llamadas?
  (*Where would you like to transfer your calls?*)
- **U(2)**: A la extensión de Juan Rubio. (*To the extension of Juan Rubio.*)
- **S(3)**: Sus llamadas han sido desviadas a la extensión de Juan Rubio. Por favor, indique el nombre, extensión o número de teléfono.
  (*Your calls have been transferred to John Smith's extension. Please specify the name, extension or phone number to make the phone call.*)

The main drawback of this approach is that it generates an unnatural dialogue structure. That is, if the user specifies a sequential set of commands, the natural and expected behaviour of the system is to execute these commands in the order explicitly stated. Again, in command–oriented dialogue systems, the order of execution is very important and may condition the usability of the system, as the result of a command usually will condition the following commands.

4.6.2 Intelligent and Expectation–driven Dialogue Move Selection

The main conclusion of the previous discussion is that both the queue and the stack–based strategies are too generic. The use of just one strategy in a dialogue manager will cover and solve some phenomena but at the same time will not be able to solve some others.

In order to allow a more natural and flexible communication between the Natural Language Understanding and the Dialogue Manager modules, a new and specific module must control the selection of dialogue moves.

This new module, the Dialogue Move Selector, will receive all the dialogue moves generated by the Natural Language Understanding module, and will reorder then so that the Dialogue Manager chooses at any time the more plausible next dialogue move, taking into account the state of the dialogue.

It is possible to distinguish at least four phenomena which require an intelligent approach to dialogue move selection:

1. **Subdialogues**

   In real dialogue systems, the postproccessing, reordering and selection of dialogue moves is not a marginal question. During its interaction with the system, the user passes over different subdialogues: commands, help, disambiguation, confirmation, etc. These subdialogues may overlap.

2. **Disconnected partial dialogue moves**

   The use of speech recognition and chunk–based parsing may generate multiple and disconnected dialogue moves which in fact correspond to just one dialogue move. For instance, if the misrecognised user's input *call ...* **noise ... errors** *... peter* (or the inverse order) generates two dialogue moves, both the queue and the stack strategies will originate non–natural dialogues.

3. **Temporal priority marks**

   A third reason which justifies the implementation of an intelligent and flexible strategy for dialogue move selection is the use of explicit priority marks. For instance, if the user indicates *Do A but first do B.* or *First do A, and next do B.*, the system must be able to adapt its behaviour to the explicit order marked by the user.

   The domain–dependent temporal dependencies are related to this notion of temporal priority explicit marks. That is, although the user states a set of commands in a free order, the dependencies among the commands may determine their order of execution.

4. **Expectation–dependent semantic interpretation**

   Finally, expectations may drive the semantic interpretation of the utterance of the user.

   Consider the following dialogue sketch: the user's utterances in **U(1)** and **U(3)** are exactly the same, but must be interpreted as different dialogue moves.

   U(1) must be considered as the beginning of a MakeCall subdialogue. Nevertheless, the interpretation of **U(3)** must be a positive confirmation.

   - **S(1)**: Bienvenido al Operador Telefónico Automático
     (*Welcome to the Telephone Operator System ...*)
   - **U(1)**: Llama.
     (*Call.*)

- **S(2)**: ¿A quién quiere llamar?
  (*Who would you like to call?*)
- **U(2)**: A Pablo Escalada.
  (*To Pablo Escalada.*)
- **S(3)**: Le paso con Pablo Escalada. ¿Correcto?
  (*I'm getting you through with Pablo Escalada. Is that correct?*)
- **U(3)**: Llama.
  (*Call.*)

This phenomenon may be solved at the dialogue level incorporating specific update rules for it. Nevertheless the list of possible replies of the user in **U(3)** will need to add a considerable list of *ad–hoc* exceptions to the default update rule.

The dialogue move selector allows an elegant and comprehensive treatment of these kind of phenomena.

At any time, this module knows the set of dialogue moves previously generated from the user's utterances (stored in its Dialogue Move Input Pool). Also, the Dialogue Move Selector gets the ordered list of dialogue moves generated from the current user's utterance. And finally, the Dialogue Move Selector can access the current information state as well as the dialogue history. This last source incorporates the set of current active expectations.

So this module has all the required information to apply an intelligent strategy to the problems that appear in the communication interface between the Natural Language Understanding and the Dialogue Manager modules.

4.6.3 The architecture of the Dialogue Move Selector

From a functional perspective, the Dialogue Move Selector is an independent module of the Dialogue Management Agent.

It has been implemented in C (ANSI) and supports all the data structures required to represent and manipulate DTAC feature structures.

Two main components define the architecture of the Dialogue Move Selector.

1. **Dialogue Move Input Pool.** This is an ordered list of DTAC structures, each one containing a different dialogue move generated by the Natural Language Understanding Module.

   Each time this module (the NLU) finishes the analysis of a new user's utterance, the list of dialogue moves is sent to the Dialogue Move Selector. This list is ordered taking into account their position in the user's utterance (from left to right).

   The order of the dialogue moves in this pool is important. The first dialogue move in the input pool will be the first one to be processed by the Dialogue Manager module.

2. **Dialogue Move Selection Rules.** The set of selection rules defines the kernel of the module. The algorithm is invoked by the NLU module each time a new list of dialogue moves has been incorporated to the Dialogue Move Input Pool. Also, it is invoked each time the Dialogue Manager module requires a new dialogue move to continue working.

Additionally, the Dialogue Move Selector controls the execution of the UserPrompt function. This function is used by the Dialogue Manager to present to the user a message which, implicit or explicitly asks the user for a reply. The Dialogue Move Selector must analyse the UserPrompt strategy and decide whether there is or not a dialogue move in the Dialogue Move Input Pool able to solve the expectation associated with the corresponding dialogue state.

4.6.4 Dialogue moves fusion: unification–based equality or inclusion

The first rule of the Dialogue Move Selector (DMS) is the fusion of dialogue moves. This rule allows the DMS to fuse different repeated (total or partially) dialogue moves into just one.

For instante, consider the following sentence:

- **U(1)**: Me gustaría llamar y activar los desvíos uhm si llamar a Pedro.
  (*I would like to phone and cancel the transfer mode uhm ok call Pedro.*)

This sentence will generate three dialogue moves:

1. **DM1/MakeCall**

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & MakeCall \\ ARGS & : & [Dest] \\ CONT & : & \end{bmatrix}$$

2. **DM2/CancelCallTransfer**

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & CancelCallTransfer \\ ARGS & : & \\ CONT & : & \end{bmatrix}$$

3. **DM3/MakeCall+Dest:Peter**

$$\begin{bmatrix} DMOVE & : & specifyCommand & \\ TYPE & : & MakeCall & \\ ARGS & : & [Dest] & \\ CONT & : & & \\ Dest & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & Name \\ ARGS & : & \\ CONT & : & Pedro \end{bmatrix} & \end{bmatrix}$$

The DMS tries to unify each pair of dialogue moves in the Input Pool. If the unification algorithm does not fail this will mean that both dialogue moves contain compatible information. If this is the case, the more informative DTAC is selected and the less informative is simply removed from the Dialogue Move Input Pool.

In the previous example the first one will be removed and the second and third ones will stay in the Input Pool.

The ordering of the dialogue moves implied is very important as it will determine their order of execution. This rule associates the unified dialogue move to the first dialogue move in the input pool.

In the example below, DM1 will point to the content of DM3 and DM3 will be removed, so the first action to be considered by the Dialogue Manager is the call to Peter.

4.6.5 Temporal priority marks

Several particles explicitly specify a temporal mark which the system should take into account.

In order to support this functionality, the formal representation of the dialogue moves contains the Priority feature. Priorities are defined by means of integer numbers (negative and positive).

The following example illustrates this phenomenon:

- **U(1)**: Cancela los desvíos después llama a Felipe Ruiz pero primero busca el correo de Paloma Abad.
  (*Cancel the transfer mode then call Felipe Ruiz but first look up Paloma Abad's email address.*)

In this case, the following three dialogue moves are generated, two of them with priority marks.

1. **DM1/CancelCallTransfer**

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & CancelCallTransfer \\ ARGS & : & \\ CONT & : & \end{bmatrix}$$

2. **DM2/MakeCall+Dest:FelipeRuiz**

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & MakeCall \\ ARGS & : & [Dest] \\ CONT & : & \\ Priority & : & -5 \\ Dest & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & Name \\ ARGS & : & \\ CONT & : & "FelipeRuiz" \end{bmatrix} \end{bmatrix}$$

3. **DM3/LookUp+Email+Dest:PalomaAbad**

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & LookUp \\ ARGS & : & [Email, Name]|[Office, Name] \\ CONT & : & \\ Priority & : & 10 \\ Email & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & Email \end{bmatrix} \\ Name & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & Name \\ ARGS & : & \\ CONT & : & "PalomaAbad" \end{bmatrix} \end{bmatrix}$$

Taking into account the values of the Priority feature (the default value is 0 for dialogue moves without that feature), DM3 will be located at the beginning of the Dialogue Move Input Pool (priority 10), the second one will be DM1 (default priority 0) and the last one will be DM2 (priority -5).

The result of the execution of this multiple command is shown next:

- **S(2)**: El correo de PALOMA ABAD GARCIA es pab@tid.es
  (*The email of PALOMA ABAD GARCIA is pab@tid.es*)
- **S(3)**: No hay desvios activos.
  (*The transfer mode is currently deactivated.*)
- **S(4)**: Quiere llamar a FELIPE RUIZ BLAZQUEZ. ¿Es correcto?
  (*You want to call FELIPE RUIZ BLAZQUEZ. Is that correct?.*)

4.6.6 Recovery of dialogue moves

This strategy is applied when the Dialogue Move Selector sends a dialogue move to the Dialogue Manager which was generated in a previous user's turn.

The following example illustrates this phenomenon. In **U(1)** the user specifies a multiple command which generates two dialogue moves:

- **S(1)**: Esta usted al habla con su operador telefonico automatico.
  (*Welcome to the Telephone Operator System ...*)
- **S(2)**: Por favor, indique la función que desee realizar.
  (*Please, specify the function you want to perform.*)
- **U(1)**: quiero hacer una llamada y cancelar los desvios.
  (*I want to make a phone call and cancel the transfer mode.*)

Next, the system initiates the dialogue corresponding to the first dialogue move corresponding to the phone call.

- **S(3)**: ¿A quien quiere llamar por telefono?
  (*Who would you like to phone*)
- **U(2)**: a roberto alonso.
  (*to roberto alonso*)
- **S(4)**: Quiere llamar a ROBERTO ALONSO COLMENARES. ¿Es correcto?
  (*Yo want to phone ROBERTO ALONSO COLMENARES. Is that ok?*)
- **U(3)**: si.
  (*yes.*)
- **S(5)**: Le pasamos a continuación con ROBERTO ALONSO COLMENARES
  (*Calling ROBERTO ALONSO COLMENARES*)

```
#### ACTION MANAGER:
####    Function: MakeCall
```

When the phone call ends, the system recovers the control of the telephone terminal and restarts the dialogue manager.

- **S(6)**: Su llamada con JUAN JOSE APARICIO GARCIA ha terminado. De nuevo está usted al habla con su Operador Telefónico Automático.
  (*Your phone call with ROBERTO ALONSO COLMENARES has ended. You are again talking with your Automatic Telephone Operator.*)

At this point, when the Dialogue Manager tries to continue the dialogue using the prompt corresponding to the Continuation dialogue rule, the dialogue move selector stops the corresponding prompt and recovers the previously stored dialogue move corresponding to the CancelCallTransfer move of **U(1)**.

As this dialogue move was generated in the first user turn and the current user turn is 3, this dialogue move is sent to the dialogue manager with a `Recovered` mark. This mark permits the dialogue manager to apply the RecoveryActions strategy, which in this case will generate the following subdialogue:

- **S(7)**: Anteriormente indico su deseo de cancelar el desvio de llamadas. Sus llamadas estan actualmente desviadas a LAURA ROMERO PEREZ. ¿Desea cancelar el desvio? (*You previously stated you would like to cancel the transfer mode. Your calls are currently transfered to LAURA ROMERO PEREZ. Would you like to cancel the tranfer?*)

4.6.7 Expectation–driven dialogue move selection

This rule allows the Dialogue Move Selector to integrate the expectations generated by the Dialogue Manager to control the linguistic interface with the user.

Basically, this rule determines whether to send or not one of the dialogue moves in the Dialogue Move Input Pool to the Dialogue Manager.

From an operational point of view, all the interaction process is controlled by the Dialogue Manager. This module begins its execution calling the STARTUP dialogue rule. All the actions specified by this rule will be executed until the dialogue manager cannot go on. In this moment, by means of a UserPrompt action, the dialogue manager will ask the user to provide the information required (for instance, which command the user wants to do next, what the destination for the phone call is, and so on).

The UserPrompt action is taken by the Dialogue Move Selector. First, if there is no dialogue move in the Dialogue Move Input Pool, the UserPrompt action proceeds. In this case, the user receives the message generated by the system and the system waits for the user's reply. But if there are one or more pending dialogue moves in the Dialogue Move Input Pool, the Dialogue Move Selector determines if any of them can solve some of the current active expectations. If this is the case, the UserPrompt action is cancelled and the corresponding dialogue move is moved form the Input Pool to the dialogue manager. Next, this module will resume its state and will continue processing the new dialogue move in the context of the active expectations.

In the previous example, once the dialogue manager has processed the first dialogue move generated in **U(1)** (the MakeCall dialogue move), the dialogue manager determines that a destination is required to continue with the MakeCall update rule:

```
( RuleID:    MAKECALL;
    PriorityLevel:    15;
    TriggeringCondition:
        (DMOVE:specifyCommand,TYPE:MakeCall);
    DeclareExpectations: {
        Dest <= (DMOVE:specifyParameter,
                 TYPE:Extension|Name|PhoneNumber);
    }
    SetExpectations: {
        Confirm <= (DMOVE:answerYN);
    }
    ActionsExpectations: {
        [Dest] => {
                ExecuteDMFunction(MakeCallDest);
        }
    ....
```

The `MakeCallDest` function associated with the Dest expectation defines the following User-Prompt action:

```
MakeCallDest: {
    UserPrompt(
        @firstMessage(
            "A quien quiere llamar por telefono?",20,
            "Por favor, indique el numero, la extension
                 o el nombre al que desea llamar.",10,
            "Para realizar la llamada, debe
                 indicar un numero, extension o nombre.",10),
        @onInputError(
            "No he entendido correctamente el destino de
                 la llamada. Por favor, repitalo de nuevo.",
            "Lo siento, pero no comprendo a quien quiere llamar.
                 Por favor vuelva a repetirlo.")
    );
}
```

At this moment, there is an active expectation (Dest). This expectation may be solved by a DTAC of the type *(DMOVE:specifyParameter, TYPE:Extension—Name—PhoneNumber)*.

In the Dialogue Move Input Pool there is a remaining dialogue move corresponding to the second part of the utterance in **U(1)**. But the DTAC corresponding to this dialogue move has DMOVE specifyCommand and TYPE CallTransfer, so it does not solve the current Dest expectation.

In this situation, the Dialogue Move Selector retains this dialogue move in the memory of

the Dialogue Move Input Pool, and continues with the UserPrompt function, asking the user about the destination of the phone call.

Chapter 5

Installation Guide

This chapter makes a detailed description of the software components which need to be installed to build the Spanish Natural Command Language Demonstrator.

Some of these components have been developed or adapted as part of the Siridus project, while some other had been developed previously by some Siridus partners. There are also some pieces of commercial software which are needed to control the PABX.

This software must be installed in a hardware platform like that described in Chapter 2. Such platform is composed of a PABX and two PCs. No specific software needs to be installed in the PABX. Section 5.1 describes the software component that must be installed in the PCU. The software for the VRU is described en Section 5.2.

5.1 Software components to install in the PCU

This section describes the different software components which need to be installed in the PCU of the Demonstrator.

This PC must be provided with NT Windows operating system (version 4.0 Workstation).

5.1.1 CTConnect and Ericsson ApplicationLink/CSTA

These are two commercial software packages which are needed to control the company's PABX.

CTConnect is a software package which enables applications running on a computer to control a PABX. It can be used with several PABX models, and is available for different platforms

(not including Solaris for PC). CTConnect for NT Windows 4.0 has been used for the Siridus NCL-Demonstrator. This software is distributed and licensed by Intel.

Ericsson ApplicationLink/CSTA is an additional software package which is required by CT-Connect to control the particular PABX model used in Telefònica I+D. It is distributed by the PABX vendor (Ericsson).

5.1.2 Remote shell server

It is necessary to install a remote shell server in the PCU to allow the necessary communication between VRU's and PCU's agents.

During the Demonstrator development, Winsock RSHD/NT version 2.21.03 has been used. This software is downloadable from Denicomp Systems web site (www.denicomp.com). The server is available for several Windows platforms (NT, 2000, XP, 95, 98, ME). The version for NT Windows has been used to be installed in the PCU.

Of course, this software could be replaced by any other remote shell server.

5.1.3 PABX Agent

This agent is in charge of PABX control. It waits for KQML messages with requests, and fulfills them using the underlying API, CTConnect.

It is an executable file for NT Windows.

This software has been developed in Telefónica I+D during the Siridus project, and the executable file is available upon request.

5.2 Software components to install in the VRU

This section describes the different software components which need to be installed in the VRU of the Demonstrator.

Before beginning the installation of the different components, Sun Solaris 2.5.1 for Intel platforms must be installed in the PC. Dialogic drivers for Antares and D41 boards need also to be correctly installed and configured.

5.2.1 Speech recognition and synthesis resources

The NCL-Demonstrator allows the user to interact with the system using his voice over the phone. This is achieved by using Telefónica's Speech Software, which includes a natural language speech recognizer, a text to speech system, and a set of additional resources to control the telephone lines and the Dialogic boards.

This software is available for several platforms (Sun Solaris, Solaris for PC, Unixware), which did not include Windows platforms when the Demonstrator development began. The version used in the Siridus Demonstrator is the one for Solaris 2.5.1 for Intel platforms.

Telefónica's Speech Software for Solaris is a unix software package which includes a set of C++ libraries, as well as some general recognition and synthesis models, and some scripts to start and control the Dialogic boards. It also provides an API which can be used by applications to control the speech recognizer, the TTS system and the telephone lines.

This software was previously developed by Telefónica I+D and is distributed on a licence basis.

5.2.2 Speech recognition models

The Demonstrator makes use of specific recognition models. These models must be generated and installed in the VRU, as the speech recognizer uses them.

These binary files, which include acoustic and linguistic models, and the recognition vocabulary, have been generated during the Siridus project by Telefónica I+D with suggestions from the University of Seville and are available upon request.

5.2.3 Dialogue Management Resources

The Demonstrator makes use of dialogue management software which has been developed at the University of Seville and which is based on the ISU approach.

This software consists of a C++ library available for Solaris 2.5.1 for PC, with an API which can be used by any application. When an event is received by an application (e.g. the beginning of a call to the system or the end of TTS production) it can call the function in the dialogue manager library designed to handle that event, and it will make the appropriate actions (e.g. requesting the TTS to utter an appropriate sentence or requesting the speech recognizer to recognize user input). In this way, the dialogue control is made by this library, and the application needs only to catch the events (beginning of a new call, end of system output, end of user input, message reception from the PABX or the database and end of the call) and call the apropriate function from the library.

This library has been built from the following software modules:

- Episteme

  Episteme is a Natural Language Understanding and Semantic Interpretation kernel. This system was previously developed by the University of Seville team, and is the basis for different dialogue systems.

- Delfos

  Delfos a Dialogue Management kernel. This system was also previously developed by the University of Seville team, and can be configured for different dialogue models and domains.

  Both Episteme and Delfos have been used in several other research projects, including DHomme.

- Lexicon and Grammar specification for the ATOS domain

  This software has been developed at the University of Seville as part of the Siridus project and contains the episteme-based specification of the lexicon and the grammar for the Telephone application.

- Dialogue specification for the ATOS domain

  This software has also been developed at the University of Seville as part of the Siridus project and contains the delfos-based specification of the dialogue model for the the Telephone application.

- Knowledge manager and Action Manager

  These two new modules have been added to delfos in order to control the knowledge data base (interface to the Telephone Directory) and the PABX (interface to the PABX agent).

- KQML wrapper

  This software has been developed during the Siridus project to allow the integration of the Dialogue Manager into the system software architecture.

  On the one hand, a C++ library has been developed at Telefónica I+D, which allows the dialogue manager functions to send and receive KQML messages to and from other system agents.

  On the other hand, a wrapper for the Delfos-based dialogue manager has been developed at the University of Seville in order it to become, from an application point of view, a library of event-handling functions with an appropriate API.

These components have been compiled and a C++ library for Solaris 2.5.1 for Intel platforms has been generated. This library, which is available upon request, must be installed in the VRU to build the NCL-Demonstrator.

5.2.4 Database

The NCL-Demonstrator makes use of a personnel directory containing some data about the company's employees.

A database must be installed to manage this directory.

Mysql has been used for this purpose. This is a freeware database server which is downloadable from Mysql web site (www.mysql.com). The version for Solaris Intel has been used for the Demonstrator.

Apart from installing the database server, the personnel directory needs to be created as a database table and installed in the unit.

The directory is available from Telefónica I+D for the different Siridus partners to work on this project.

5.2.5 Executable system agents

The executable files for the agents described in this deliverable have to be installed in the VRU.

These agents have been developed or adapted at Telefónica I+D as part of the Siridus project, and are available upon request.

They all consist of executable files for Solaris 2.5.1 for Intel platforms, and some of them make use of the resources described above.

In fact, the LINE-INT agent makes use of the speech recognition and synthesis resources, the Database agent makes use of the database and the Dialogue Manager agent makes use of the Dialogue Management Resources.

A script to start all the agents is also available. When this script is executed, every agent is started and connected to the rest of the agents. In this way, the Demonstrator is started and is ready to receive calls from any user. A user's guide is presented in the next chapter of this deliverable.

Chapter 6

User's Guide

6.1 General System Functionality

The Automated Phone Operator incorporates a series of possible commands that define the system functionality. This list of possibilities can be divided in two major types of commands:

- Domain specific commands:
  - Make a phone call.
  - Redial.
  - Last dialed number.
  - Conference call.
  - Call Transfer.
  - Cancel the call transfer.
  - Current transfer destination.
  - Look up an email or an office number.
  - Help.
  - List all the matching names.
- General commands:
  - Quit
  - Greet
  - Retry
  - Repeat the last message.

6.2 How to Activate the Commands

The basic strategy to activate any of the commands is to just utter it in a natural way. The system is ready to understand any natural way to express any of the commands listed above.

Listed below there are some examples to illustrate the system flexibility in terms of natural commands:

6.2.1 To make a phone call

- Quiero hacer una llamada.
- Por favor llamame al numero 123456789.
- Pongame con Juan Alonso Martin
- Llamar a la extension 12345.
- Me gustaria que me pusieses con la extension de la senorita Paloma Abad.

Although these are just some of the many possibilities, it is obvious that the system can cope with simple as well as very elaborated commands, formal or informal register, and complete as well as incomplete commands.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: hola llama a luis gallo.

Out[3]: Hola, que tal

Out[4]: Quiere llamar a LUIS GALLO ALONSO. Es correcto?

In[2]: si.

Out[5]: Le pasamos a continuacion con LUIS GALLO ALONSO

#### ACTION MANAGER:
####    Function: MakeCall
```

6.2.2 To redial

- Quiero hacer una rellamada.

- Vuelve a llamarle.
- Rellamada.

As in the former case, any natural way to ask the system to redial will be perfectly fine.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: rellamada.

Out[3]: El destino de su ultima llamada fue:LUIS GALLO ALONSO.
        Quiere volver a llamar?

In[2]: si.

Out[4]: Le pasamos a continuacion con LUIS GALLO ALONSO

#### ACTION MANAGER:
####    Function: MakeCall
```

In the case of actions that may have undesired consequences if there is a mistake, the system is designed to always confirm with the user whether this is really what they want to do.

6.2.3 To find out the last dialed number

- A donde hice mi ultima llamada?
- Cual fue el destino de mi ultima llamada?
- A quien hiciste la ultima llamada?

In this case the system not only provides the information, but also becomes collaborative and offers to call that last number again.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: a donde hice mi ultima llamada.
```

```
Out[3]: El destino de su ultima llamada fue:LUIS GALLO ALONSO.
        Quiere volver a llamar?

In[2]: no gracias.

Out[4]: Que mas puedo hacer por usted?
```

One of the advantages of the current system set–up is that the last number dialed is available even if it was dialed in a different session. When a call is made, the system automatically updates the directory that contains the user's information, including the last dialed number.

6.2.4 To make conference call

- Multiconferencia.
- Quiero hacer una multiconferencia con Luis y Antonio.
- Multillamada a Luis.
- Me gustaria llamar a Paloma Abad y Luis Asensio.

The user has the option to give all arguments at once, part of them or none of them. In any case, the system will know what information is required and will prompt the user appropriately.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: multiconferencia con luis y paloma.

Out[3]: He encontrado 82 entradas para LUIS en el directorio.
        Entre ellas: JOSE LUIS ANTA DE LA IGLESIA, LUIS ANTON
        GALINDO, LUIS MIGUEL APIO RECIO, etc.. Por favor, indique
        informacion adicional.

In[2]: anta.

Out[4]: He encontrado 5 entradas para PALOMA en el directorio.
Entre ellas: PALOMA ABAD BLAZQUEZ, PALOMA ALMARAZ MAZARIEGOS,
PALOMA CARDENAL CAMPOAMOR, etc.. Por favor, indique informacion
adicional.

In[3]: cardenal.
```

```
Out[5]: Quiere llamar en multiconferencia a JOSE LUIS ANTA DE
        LA IGLESIA y PALOMA CARDENAL CAMPOAMOR. Es correcto?

In[4]: si.

#### ACTION MANAGER:
####    Function: ConferenceCall
```

In the case above, both arguments were given straight away by the user. However these arguments were ambiguous and needed to be disambiguated. In the case below, both destinations were clear, but the user did not provide them straight away and had to be prompted to do it.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: multiconferencia.

Out[3]: Digame con quienes quiere hablar por favor.

In[2]: paloma abad y luis anta.

Out[4]: Quiere llamar en multiconferencia a PALOMA ABAD BLAZQUEZ
        y JOSE LUIS ANTA DE LA IGLESIA. Es correcto?

In[3]: si.

#### ACTION MANAGER:
####    Function: ConferenceCall
```

6.2.5 To transfer your calls to a different number.

- Desvia mis llamadas por favor.
- Quiero que desvies mis llamadas a Pedro Gomez.
- Podrias desviar mis llamadas por favor.
- Desvio de llamadas a la extension 12345.

All cases would be equally valid to activate the call transfer function. As in other cases, if the destination of the transfer is ambiguous, it will be disambiguated and confirmed with the user before executing the transfer.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: desvia mis llamadas a carlos garcia martinez.

Out[3]: Quiere desviar sus llamadas a CARLOS GARCIA MARTINEZ.
        Correcto?

In[2]: si.

Out[4]: A partir de este momento todas sus llamadas seran desviadas
        a CARLOS GARCIA MARTINEZ

#### ACTION MANAGER:
####    Function: CallTransfer
```

It is also important to note that in addition to the confirmation request, the system will report the final result of the command to assure the user that it will be executed. This is important in cases in which the execution of the command has no immediate or obvious results to the user.

6.2.6 To cancel a call transfer

- Cancela el desvio de llamadas.
- Interrumpe el desvio.
- Desactiva el desvio de llamadas.
- Anuleme el desvio de llamadas por favor.

The system will not deactivate the current call transfer without first reporting what the current destination of the transfer is, and confirming the cancellation with the user.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: desactiva el desvio de llamadas.

Out[3]: Sus llamadas estan actualmente desviadas a CARLOS GARCIA
        MARTINEZ. Desea cancelar el desvio?
```

```
In[2]: si.

Out[4]: A partir de este momento todas sus llamadas seran recibidas de nuevo en su termina

#### ACTION MANAGER:
####    Function: CancelCallTransfer
```

6.2.7 To find out the current destination of your call transfer

- A quien estan desviadas mis llamadas?
- A donde estan desviadas mis llamadas?

With this function the user can make sure of the current destination of their call transfer.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: a quien estan desviadas mis llamadas.

Out[3]: No hay desvios activos.

Out[4]: Que mas puedo hacer por usted?
```

6.2.8 To Look up the email or office number of somebody in the directory

- Busca el email de Carlos Garcia
- El correo electronico de Luis Asensio por favor.
- Dame el numero de oficina de Paloma.
- Encuentrame el correo de Pedro Martinez.
- Busca.

As in former cases, the system will accept either complete or incomplete commands. In the latter case, it will prompt the user for the information needed to continue with the execution of the command. This command requires the name of somebody in the directory, so as we have seen in other cases, the system will also disambiguate any ambiguous names.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.
```

```
Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: dame el email de paloma abad.

Out[3]: El email de PALOMA ABAD BLAZQUEZ es pab@tid.es

Out[4]: Alguna otra funcion con la que desee continuar?
```

In the example above, the user provides thede system with all the information at once. However, there are cases in which the user might not be as collaborative and the system has to guide them step by step, as in the example below.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: buscame.

Out[3]: Que informacion desea saber?

In[2]: el correo.

Out[4]: De quien quiere saber la direccion de correo electronico?

In[3]: de paloma.

Out[5]: He encontrado 5 entradas para PALOMA en el directorio.
Entre ellas: PALOMA ABAD BLAZQUEZ, PALOMA ALMARAZ MAZARIEGOS, PALOMA CARDENAL CAMPOAMOR, e

In[4]: abad.

Out[6]: El email de PALOMA ABAD BLAZQUEZ es pab@tid.es
```

Note that in this case the system does not need to confirm whether it is Paloma Abad Blazquez or not, before reporting the result of the search. Confirmation is reserved for cases in which a mistake would have undesirable consequences, such aslike for instance, calling or transferring calls to the wrong person.

6.2.9 To ask for help

- Ayuda.
- Ayudame con la multiconferencia.

- Ayuda sobre el desvio de llamadas.

The user can ask for help at any point during the conversation. The system has been designed to provide contextual help, i.e., it will provide information about the current dialogue phase taking into account active expectations.

If the user just asks for help and there is no context, then general help about the system will be provided. If the user asks for specific help about one of the functions, then the information will be specific to that function. The user can however ask for help while in the process of providing information for the execution of a particular command. In that case, the system will analyze the context in which help was requested and will adequate the message to those particular circumstances.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: ayudame.

Out[3]: Soy su Operador Telefonico Automatico. A peticion suya, puedo
        hacer llamadas, rellamadas, multiconferencias, desviar
        sus llamadas, buscar la oficina o el email de alguien
        o cancelar sus desvios. Simplemente digame que le gustaria hacer.
        Tambien puedo darle ayuda especifica sobre cada una
        de esas funciones.

Out[4]: Alguna otra funcion con la que desee continuar?

In[2]: ayuda sobre multiconferencia.

Out[5]: Para hablar con dos personal al mismo tiempo, puede
        usted hacer una multiconferencia. Diga que quiere hacer
        una multillamada o multiconferencia y especifique los
        nombres o extensiones de destino.

Out[6]: Que mas puedo hacer por usted?
```

These are cases in which there is no context defined, and then general help is provided. When the user specifies the type of help they need, specific help is provided. In the case below however, the context is determined by the current dialogue phase and active expectations.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?
```

```
In[1]: busca.

Out[3]: Que informacion desea saber?

In[2]: ayudame.

Out[4]: Con esta funcion, usted puede buscar el numero de oficina
        o la direccion de email de cualquier persona en el directorio
        general. Simplemente digame que busque la informacion
        deseada.

Out[5]: Que informacion desea saber?

In[3]: el numero de oficina.

Out[6]: De quien quiere saber el numero de oficina?

In[4]: ayuda.

Out[7]: Necesito el nombre de la persona de la que quiere saber
        el numero de oficina.

Out[8]: De quien quiere saber el numero de oficina?

In[5]: paloma abad.

Out[9]: La oficina de PALOMA ABAD BLAZQUEZ es la numero 104-E
```

6.2.10 To redo the last command

- Vuelve a hacerlo.
- Hazlo otra vez.
- Quiero que lo vuelvas a hacer.

The system can make use of the dialogue history to recuperate the last command executed and redo it.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: multiconferencia a paloma abad y luis gallo.
```

```
Out[3]: Quiere llamar en multiconferencia a PALOMA ABAD BLAZQUEZ
        y LUIS GALLO ALONSO. Es correcto?

In[2]: si.

#### ACTION MANAGER:
####    Function: ConferenceCall

Out[4]: Que mas puedo hacer por usted?

In[3]: vuelve a hacerla.

Out[5]: Quiere llamar en multiconferencia a PALOMA ABAD BLAZQUEZ
        y LUIS GALLO ALONSO. Es correcto?

In[4]: si.

#### ACTION MANAGER:
####    Function: ConferenceCall
```

6.2.11 To list all matches

We have enabled our system to collect all the matching entries in order to disambiguate names. What follows is to enable the user to have all the matching entries listed on request. This is very useful in the case in which the user may not remember the first name or last name of the person they want to call, but would recognize it if they heard it.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: llama a alvaro.

Out[3]: He encontrado 7 entradas para ALVARO en el directorio.
        Entre ellas: RICARDO ALVARO FERNANDEZ, ALVARO ARMENTEROS
        PACHECO, SUSANA BERNABE ALVARO, etc.. Por favor, indique
        informacion adicional.

In[2]: Listame las opciones por favor

Out[3]: Las entradas para ALVARO en el directorio son:
        RICARDO ALVARO FERNANDEZ, ALVARO ARMENTEROS
        PACHECO, SUSANA BERNABE ALVARO, ALVARO CAPPA GOMEZ,
```

```
        ALVARO MARTINEZ REOL, JESUS ALVARO MATE ENRIQUEZ y por ultimo,
        ALVARO SAINZ-PARDO GUTIERREZ. Por favor, seleccione una opcion.
```

6.2.12 To quit the system

- Adios.
- Sal.
- Abandona.

All these commands will make the system prompt the user for confirmation before actually quitting.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: llama a paloma abad.


Out[3]: Quiere llamar a PALOMA ABAD BLAZQUEZ. Es correcto?

In[2]: si.

Out[4]: Le pasamos a continuacion con PALOMA ABAD BLAZQUEZ

#### ACTION MANAGER:
####    Function: MakeCall

Out[5]: Su llamada con PALOMA ABAD BLAZQUEZ ha terminado. De
        nuevo esta usted al habla con su Operador Telefonico Automatico.


Out[6]: Alguna otra funcion con la que desee continuar?

In[3]: adios.

Out[7]: Quiere usted abandonar el sistema?

In[4]: si.

Out[8]: Muchas gracias. Adios
```

6.2.13 To have something repeated

One of the common events in a conversation, especially over the phone, is that one of the speakers may not be able to hear the utterance from the other one due to noise or other possibilities. In any case, human speakers always have the option to ask their counter speaker to repeat what they said last. This is why our system can be asked to repeat its last prompt on request from the user.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: llama a valverde.

Out[3]: He encontrado 3 entradas para VALVERDE en el directorio: JOSE
ANTONIO AGUILAR VALVERDE, MARIA CRUZ MENDEZ VALVERDE y JOSE MANUEL
PALACIOS VALVERDE. Por favor, indique una de ellas.

In[2]: Me lo puede repetir por favor.

Out[4]: Por supuesto.

Out[5]: He encontrado 3 entradas para VALVERDE en el directorio: JOSE
ANTONIO AGUILAR VALVERDE, MARIA CRUZ MENDEZ VALVERDE y JOSE MANUEL
PALACIOS VALVERDE. Por favor, indique una de ellas.
```

6.2.14 Greetings

The system interface has been designed with a polite–friendly personality, so it is important that the system recognizes polite formulae and responds politely.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: hola buenos dias.

Out[3]: Hola, que tal

Out[4]: Que funcion desea realizar?
```

Now that all basic functions have been explained, the additional functionality of the system can be illustrated.

6.3 Additional Functionality

6.3.1 Last Utterance Repetition

The system incorporates the ability to repeat the last prompt played on request. There are situations, especially in this domain, when the user has missed the message or part of it due to some external noise or distraction. In human–human conversation, the most natural course of action is to ask the speaker to repeat the message, and in this case, the system will respond naturally to this situation.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: deseo llamar a pedro.

Out[3]: He encontrado 26 entradas para PEDRO en el directorio.
        Entre ellas: PEDRO ACEBES BAYON, PEDRO ANTONIO ALONSO
        BAIGORRI, PEDRO ALTARES PEREZ, etc.. Por favor, indique
        informacion adicional.

In[2]: perdon podria repetir.

Out[4]:Por supuesto.

Out[5]: He encontrado 26 entradas para PEDRO en el directorio.
        Entre ellas: PEDRO ACEBES BAYON, PEDRO ANTONIO ALONSO
        BAIGORRI, PEDRO ALTARES PEREZ, etc.. Por favor, indique
        informacion adicional.
```

6.3.2 Multiple Command Utterances

One of the most interesting features the system incorporates is the ability to handle multiple commands at once. The system will be able to understand several commands, and remember to execute them. The order of execution is the order in which they were given by default, although the user may also specify which one they want done first.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.
In[1]: llama y desvia.
```

```
Out[3]: A quien quiere llamar por telefono?

In[2]: a juan.

Out[4]: He encontrado 64 entradas para JUAN en el directorio.
        Entre ellas: JUAN MANUEL ALONSO RODRIGO, JUAN JOSE ANDRES
        GUTIERREZ, JUAN JOSE APARICIO GARCIA, etc.. Por favor,
        indique informacion adicional.

In[3]: alonso.

Out[5]: Quiere llamar a JUAN MANUEL ALONSO RODRIGO. Es correcto?

In[4]: si.

Out[6]: Le pasamos a continuacion con JUAN MANUEL ALONSO RODRIGO

#### ACTION MANAGER:
####    Function: MakeCall

Out[7]: No es posible establecer la llamada con JUAN MANUEL
        ALONSO RODRIGO. Vuelva a llamar dentro de unos minutos.
        Mientras tanto puede usar el Operador Telefonico Automatico
        para realizar otras tareas.

Out[8]: Previamente indico que deseaba desviar sus llamadas,
        aunque no especifico el destino. Si desea realizar el
        desvio, por favor especifique un nombre o extension.
        En caso contrario indique una nueva funcion.

In[5]: laura.

Out[9]: Quiere desviar sus llamadas a LAURA MARIA ALCON DEL
        LAGO. Correcto?

In[7]: si.

Out[10]: A partir de este momento todas sus llamadas seran desviadas
         a LAURA MARIA ALCON DEL LAGO

#### ACTION MANAGER:
####    Function: CallTransfer

Out[11]: Que mas puedo hacer por usted?
```

The example above illustrates several features. In the first place, the system understands both commands, and given that no order has been specified, the first command uttered is executed first. Both commands are incomplete and require the system to prompt the user for additional information.

Note that once the call has been executed, the system also checks whether the call has been successfully established and if not, informs the user about the situation.

One more thing to take into account is the way the system resumes the execution of commands once the previous one is over. How the system does this will be explained in the next section.

There is no limit to the number of commands the system is able to remember, although it would be unnatural to expect more than three or even four in some far–fetched cases.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: haz una rellamada cancela mi desvio de llamadas y llama a paloma abad.

Out[3]: El destino de su ultima llamada fue:JUAN MANUEL ALONSO
        RODRIGO. Quiere volver a llamar?

In[2]: si.

Out[4]: Le pasamos a continuacion con JUAN MANUEL ALONSO RODRIGO

#### ACTION MANAGER:
####    Function: MakeCall

Out[5]: Su llamada con JUAN MANUEL ALONSO RODRIGO ha terminado.
        De nuevo esta usted al habla con su Operador Telefonico Automatico.


Out[6]: Previamente habia indicado que queria cancelar el desvio
        de llamadas. Sus llamadas estan actualmente desviadas
        a LAURA MARIA ALCON DEL LAGO. Desea cancelar el desvio?

In[3]: si.

#### ACTION MANAGER:
####    Function: CancelCallTransfer

Out[7]: Previamente indico que queria llamar a PALOMA ABAD.
        Desea realizar esa llamada a continuacion?
```

```
In[4]: si.

Out[8]: Le pasamos a continuacion con PALOMA ABAD BLAZQUEZ

#### ACTION MANAGER:
####    Function: MakeCall
```

6.3.3 Temporal Markers

Not only can the system handle several commands uttered at once, but it can also prioritize their execution given the user's instructions. There are certain temporal markers that speakers normally use to indicate that one action should be performed before or after the other, and the system takes them into account (DMS) to determine the order of execution.

```
In[1]: desvia mis llamadas a paloma abad pero primero llama a luis gallo.

Out[3]: Quiere llamar a LUIS GALLO ALONSO. Es correcto?

In[2]: si.

Out[4]: Le pasamos a continuacion con LUIS GALLO ALONSO

#### ACTION MANAGER:
####    Function: MakeCall

Out[5]: Su llamada con LUIS GALLO ALONSO ha terminado. De nuevo
        esta usted al habla con su Operador Telefonico Automatico.

Out[6]: Previamente habia indicado la activacion del desvio
        de llamadas a PALOMA ABAD. Desea ejecutarlo?

In[3]: si.

Out[7]: A partir de este momento todas sus llamadas seran desviadas
        a PALOMA ABAD BLAZQUEZ

#### ACTION MANAGER:
####    Function: CallTransfer

Out[8]: Que mas puedo hacer por usted?
```

The example above illustrates how the system understands non–default order of execution. Nonetheless, the system has been endowed with a more sophisticated algorithm that will be able to handle the situation, even when where more than one temporal marker is present.

```
Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: Primero llama aunque antes desvia.

Out[3]: Indiqueme a donde le gustaria desviar sus llamadas,
        por favor.

In[2]: a paloma abad.

Out[4]: Quiere desviar sus llamadas a PALOMA ABAD BLAZQUEZ.
        Correcto?

In[3]: si.

Out[5]: A partir de este momento todas sus llamadas seran desviadas
        a PALOMA ABAD BLAZQUEZ

#### ACTION MANAGER:
####    Function: CallTransfer

Out[6]: Anteriormente menciono que queria hacer una llamada.
        Si desea realizarla, por favor indique a quien quiere
        llamar. En caso contrario indique una nueva funcion.

In[4]: a luis gallo.

Out[7]: Quiere llamar a LUIS GALLO ALONSO. Es correcto?

In[5]: si.

Out[8]: Le pasamos a continuacion con LUIS GALLO ALONSO

#### ACTION MANAGER:
####    Function: MakeCall
```

6.3.4 Recovery Actions

All the examples given so far with multiple commands also illustrate the system ability to recover a command that was given and has not yet been executed. This is very important in order to avoid abrupt changes that may confuse the user. Once the current command has been executed, the system will bring up the next command in the list and confirm with the user whether they still want to go on with it. If that is the case and more information is needed, the system will proceed to gather that information normally.

6.3.5 Task Accommodation

In order to recreate as closely as possible human–human interaction in this domain, the system incorporates the ability to presume the command, given a parameter. This has been done for the case of phone calls.

If the user starts the conversation with just a name, the system will deduce they want to call that person. This is done by means of a default state that awaits in the background in case this happens.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: con carlos garcia martinez.

Out[3]: Quiere llamar a CARLOS GARCIA MARTINEZ. Es correcto?

In[2]: si.

Out[4]: Le pasamos a continuacion con CARLOS GARCIA MARTINEZ

#### ACTION MANAGER:
####    Function: MakeCall
```

6.3.6 Task Re–Accommodation

This feature is similar to the former one. The difference is that task accommodation occurs at the beginning of the dialogue, whereas task re–accommodation will happen during the dialogue. The strategy is simple but very efficient and covers the cases in which the user wants to perform the same command, changing one of several of the parameters.

```
Out[1]: Esta usted al habla con su operador telefonico automatico.

Out[2]: Por favor, indique la funcion que desee realizar.

In[1]: llama a luis gallo.

Out[3]: Quiere llamar a LUIS GALLO ALONSO. Es correcto?

In[2]: si.

Out[4]: Le pasamos a continuacion con LUIS GALLO ALONSO
```

```
#### ACTION MANAGER:
####    Function: MakeCall

Out[5]: Su llamada con LUIS GALLO ALONSO ha terminado. De nuevo
        esta usted al habla con su Operador Telefonico Automatico.


Out[6]: Alguna otra funcion con la que desee continuar?

In[3]: ahora a paloma abad.

Out[7]: Quiere llamar a PALOMA ABAD BLAZQUEZ. Es correcto?

In[4]: si.

Out[8]: Le pasamos a continuacion con PALOMA ABAD BLAZQUEZ

#### ACTION MANAGER:
####    Function: MakeCall
```

This is not only in the case of phone calls, but for any of the commands available.

```
Out[1]: Hola, le atiende su operador telefonico automatico.

Out[2]: Que funcion desea realizar?

In[1]: desvia mis llamadas a carlos garcia martinez.

Out[3]: Quiere desviar sus llamadas a CARLOS GARCIA MARTINEZ.
        Correcto?

In[2]: si.

Out[4]: A partir de este momento todas sus llamadas seran desviadas
        a CARLOS GARCIA MARTINEZ

#### ACTION MANAGER:
####    Function: CallTransfer

Out[5]: Alguna otra funcion con la que desee continuar?

In[3]: y ahora a paloma abad.

Out[6]: Quiere desviar sus llamadas a PALOMA ABAD BLAZQUEZ.
```

```
        Correcto?

In[4]: si.

Out[7]: A partir de este momento todas sus llamadas seran desviadas
        a PALOMA ABAD BLAZQUEZ

#### ACTION MANAGER:
####    Function: CallTransfer
```

Bibliography

[Alexandersson *et al* 1998] Alexandersson, J., B. Buschbeck-Wolf, T. Fujinami, M. Kipp, S. Koch, E. Maier, N. Reithinger, B. Schmitz & M. Siegel. 1998. *Dialogue Acts in VERBMOBIL-2.* DFKI Saarbrcken and TU Berlin, Report 226, July 1998.

[Allen *et al* 2001] Allen, J., Ferguson, G. & Stent, A. 2001. An Architecture for More Realistic Conversational Systems. *Proceedings of Intelligent User Interfaces 2001.* Available on http://www.cs.rochester.edu/research/trips/papers.

[Alvarez *et al* 1997] Alvarez, J., Tapias, D., Crespo, C., Cortázar, I., & Martínez. 1997. Development and Evaluation of the ATOS Spontaneous Speech Conversational System. *International Conference on Acoustics, Speech and Signal Processing*, 1139–1142.

[Amores & Quesada 2000] Amores, J. G., Quesada, J. F. 2000. *Dialogue Moves in Natural Command Languages.* Deliverable 1.1. Siridus project.

[Andry *et al* 1990] Andry, F., E. Bilange, F. Charpentier, K. Choukri, M. Ponamalé, & S. Soudoplatoff. 1990. *Computerised simulation tools for the design of an oral dialogue system.* Selected Publications, 1988-1990, SUNDIAL Project (Esprit P2218). Commission of the European Communities.

[Aust & Oerder 1995] Aust, H. & M. Oerder. 1995. Dialogue Control in Automatic Inquiry Systems. In Andernach, J. A., S. P. van der Burgt & G. F. van der Hoeven. eds. *Proceedings of the 9th Twente Workshop on Language Technology.* University of Twente, Netherlands.

[Aust *et al* 1995] Aust, H., M. Oerder, F. Seide & V. Steinbiss. 1995. The Philips automatic train timetable information system. *Speech Communication*, 17, 249-262.

[Austin 1962] Austin, J. L. 1962. *How to Do Things with Words.* Oxford: Oxford University Press.

[Barwise & Perry 1983] Barwise, J. & Perry, J. 1983. *Situations and attitudes.* Cambridge, Mass: The MIT Press.

[Bresnan 1982] Bresnan, J. ed. 1982. *The Mental Representation of Grammatical Relations.* Cambridge, MA: The MIT Press.

[Bunt & Tomita 1996] Bunt, H. & Tomita, M. eds. 1996. *Recent Advances in Parsing Technology.* Kluwer Academic Publishers.

[Carberry 1990] Carberry, S. 1990. *Plan recognition in natural language dialogue.* ACL-MIT Press Series in Natural Language Processing. Bradford Books, MIT Press.

[Carbonell & Hayes 1983] Carbonell, J., Hayes, P. 1983. Recovery Strategies for Parsing Extragrammatical Language *American Journal of Computational Linguistics*, **9**(3-4): 123-146.

[Castejón *et al* 1994] Castejón, F., Escalada, J.G., Monzón, L., Rodríguez, M.A., & Sanz, P. 1994 Un conversor texto voz para español. *Comunicaciones de Telefónica I+D*, vol. 5 no. 2

[Cooper *et al* 1999] Cooper, R., S. Larsson, C. Matheson, M. Poesio & D. Traum. 1999. *Coding Instructional Dialogue for Information States.* TRINDI (LE4-8314) Deliverable D1.1, February 1999.

[Dalrymple *et al* 1995] Dalrymple, M., Kaplan, R , Maxwell III, J. M. & Zaenen, A. eds. 1995. *Formal Issues in Lexical–Functional Grammar.* CSLI Lecture Notes 47. Stanford, California: Center for the Study of Language and Information.

[Eijck & Kamp 1997] Eijck, J. van & Kamp, H.. 1997. Representing Discourse in Context. In van Benthem, J. & Meulen, A. ter. eds. *Handbook of Logic and Language.* Elsevier Science. pp. 179-237.

[Ferguson *et al* 1996] Ferguson, G., Allen, J.F., Miller, B.W., & Ringger, E.K. 1996 *The Design and Implementation of the TRAINS-96 System: A Prototype Mixed-Initiative Planning Assistant*, TRAINS Technical Note 96-5. Available at http://www.cs.rochester.edu:80/u/ringger/research/tn96-5.html. University of Rochester, Computer Science.

[Finnin *et al* 1993] Finnin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzon, R., McKay, D., McGuire, J., Pelavin, R., Shapiro, S., & Beck, C. 1993. *Specification of the KQML Agent-Communication Language.* Draft 15 June 1993. Available at http://www.cs.umbc.edu/kqml. University of Maryland Baltimore County.

[Freedman 2000] Freedman, R. 2000. A Reactive Approach to Dialogue Planning in an Intelligent User Interface. *Workshop on Using Plans in Intelligent User Interfaces, International Conference on Intelligent User Interfaces (IUI 2000)*, New Orleans.

[Giachin & McGlashan 1997] Giachin, E., McGlashan, S. 1997. Spoken Language Dialogue Systems. In Young, S., Bloothooft (eds.) *Corpus-based methods in language and speech processing.* Dordrech: Kluwer Academic Publishers, pp. 69-117.

[Gibbon *et al* 1997] Gibbon, D., Moore, R., Winski, R. (eds.) 1997. *Handbook of Standards and Resources for Spoken Language Systems.* New York: Mouton de Gruyter.

[Grosz & Kraus 1993] Grosz, B., Kraus, S. 1993. Collaborative plans for group activities. *Proceedings of IJCAI-93*, volumen 1, pages 367-373.

[Heeman 1997] Heeman, P. A. 1997. *Speech Repairs, Intonational Boundaries and Discourse Markers: Modeling Speakers' Utterances in Spoken Dialog* Ph.D. Dissertation. Department of Computer Science, University of Rochester, New York.

[Hobbs *et al* 1992] Hobbs, J., Appelt, D., Bear, J., Tyson, M. & Magerman, D. 1992. Robust Processing of Real-World Natural-Language Texts In P. Jacobs. ed. *Text Based Intelligent Systems*, 13-33. Lawrence Erlbaum Associates, Hillsdale, NJ.

[Hodgson 1991] Hodgson, J.P.E. 1991. *Knowledge Representation and Language in AI.* Chichester, England: Ellis Horwood.

[Kamp 1981] Kamp, H. 1981. A theory of truth and discourse representation. In Groenendijk, J., Jansen, T. & Stockhof, M. eds. *Formal methods in the study of language.* Amsterdam: Mathematical Centre tracts 135.

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

[Kaplan & Bresnan 1982] Kaplan, R. M. & Bresnan, J. 1982. Lexical–Functional Grammar: A Formal System for Grammatical Representation. Functional Uncertainty. In [Bresnan 1982], 173–281. Also in [Dalrymple *et al* 1995], 29–130.

[Kaplan & Zaenen 1989] Kaplan, R. M. & Zaenen, A. 1989. Long-distance Dependencies, Constituent Structure, and Functional Uncertainty. In [Dalrymple *et al* 1995], 137–165.

[Kay 1980] Kay, M. 1980. Algorithm Schemata and Data Structures in Syntactic Processing. *Report CSL-80-12*, Xerox Palo Alto Research Center, Palo Alto, CA. También en Grosz+al:82, 35–70.

[Kirchner 1990] Kirchner, C. ed. 1990. *Unification.* San Diego, California: Academic Press Inc.

[Koskenniemi 1984] Koskenniemi, K. 1984. Morphology with Two Level Rules and Negative Rule Features. *Proceedings of COLING-88*, Budapest, Hungary.

[Kowtko *et al* 1992] Kowtko, J. C., S. D. Isard & G. M. Doherty. 1992. *Conversational Games within Dialogue.* Research Paper HCRC/RP-31, Human Communication Research Centre.

[Labrou & Finnin 1996] Labrou, Y. & Finnin, T. 1996 A Proposal for a New KQML Specification. Available at http://www.cs.umbc.edu/kqml. University of Maryland Baltimore County.

[Levinson 1981] Levinson, S. 1981. Some pre-observations on the modelling of dialogue. *Discourse Processes*, 4(1), 93-116.

[Levinson 1983] Levinson, S. 1983. *Pragmatics.* Cambridge: Cambridge University Press.

[Lewin *et al* 2000] Lewin, I., Rupp, C. J., Hieronymus, J., Milward, D., Larsson, S., Bermann, A. 2000. *Siridus System Architecture and Interface Report (Baseline)* Deliverable 6.1. Siridus project.

[Lochbaum 1994] Lochbaum, K. E. 1994. *Using Collaborative Plans to Model the Intentional Structure of Discourse.* PhD thesis, Harvard University.

[López & Quesada 1998] López, M. T. & Quesada, J. F.. 1998. Spoken Language Parsing Strategies in a Conversational System. In *Proceedings of ECAI'98: XIII European Conference on Artificial Intelligence*, 203-204.

[López & Quesada 1999] López, M. T. & Quesada, J. F.. 1999. Error Detection and Error Recovery from Speech Recognition: Language Engineering Strategies. *XIV International Congress of Phonetic Sciences.* San Francisco, CA.

[Luperfoy 1998] Luperfoy, S. (ed.) 1998. *Automated Spoken Dialog Systems.* Cambridge: MA: MIT Press.

[Maier *et al* 1997] Maier, E., Mast, M., Luperfoy, S. (eds) 1997. *Dialogue Processing in Spoken Language Systems.* Springer-Verlag.

[Martin *et al* 1996] Martin, P., Crabbe, F., Adams, S., Baatz, E., Yankelovich, N. 1996. SpeechActs: A Spoken Language Framework. *IEEE Computer*, **29(7)**.

[Noord *et al* 1998] Noord, G. van, G. Bourna, R. Koeling & M. J. Nederhof. 1998. Robust Grammatical Analysis for Spoken Dialogue Systems. *Natural Language Engineering*, 1(1), 1-48.

[Poesio *et al* 1999] Poesio, M., R. Cooper, S. Larsson, D. Traum & C. Matheson. 1999. Annotating Conversations for Information State Updates. In *Proceedings of the Amstelogue99 Workshop on the Semantics and Pragmatics of Dialogue.* Amsterdam University, May 1999.

[Quesada *et al* 2000] Quesada, J. F., Torre, D., Amores, G. 2000. *Design of a Natural Command Language Dialogue System* Deliverable 3.2. Siridus project.

[Quesada & Amores 1995] Quesada, J. F. & Amores, J. G. 1995. A Computational Model for the Efficient Retrieval of Very Large Structure–Based Knowledge Bases. In *Proceedings of the Knowledge Representation, Use and Storage for Efficiency (KRUSE'95) Symposium,* 86–96. Santa Cruz, California.

[Quesada 1997] Quesada, J. F. 1997. *El Algoritmo SCP de Análisis Sintáctico Mediante Propagación de Restricciones.* [The SCP Parsing Algorithm based on Syntactic Constraint Propagation]. PhD Thesis. Universidad de Sevilla.

[Quesada 1998] Quesada, J. F. 1998. The Lexical Object Theory: Specification Level. *Grammars*, 1, 57-84.

[Quesada 2000] Quesada, J. F. 2000. A Uniform Model for the Specification, Representation and Unification of Linguistic Information: the L^3U and *CU* Unification Algoritms. In Nepomuceno, A., Quesada, J. F., & Salguero, F. eds. 2000. *Logic, Language and Information: Proceedings of the First Workshop on Logic and Language*, ILLI, University of Seville, pp. 253–262.

[Quesada *et al* 2000] Quesada, J.F., Amores, J.G., Fernández, G., Bernal, J.A., & López, M.T. 2000. Design constratins and representation for dialogue management in the automatic telephone operator scenario. In Poesio, M., and Traum, D. *Proceddings of Götalog 2000*, Gothenburg Papers in Computational Linguistics 00-5, pp. 137–142.

[Quesada *et al* 2001] Quesada, J. F., Amores, J. G., Bos, J., Ericsson, S., Gorrell, G., Knight, S., Lewin, I., Milward, D., Rayner, M. 2001. *Configuring Linguistic Components in a Plug and Play Environment.* Dhomme project. Deliverable 3.1.(http://www.ling.gu.se/projekt/dhomme/)

[Rickel *et al* 2000] Rickel, J., Ganeshan, R., Rich, C., Sidner, C.L., Lesh, N. 2000. Task-Oriented Tutorial Dialogue: Issues and Agents. *AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications.* Technical Report FS-00-01, November 2000, pp 52-57.

[Rozenberg & Salomaa 1997] Rozenberg, G. & A. Salomaa. eds. 1997. *The Handbook of Formal Languages.* Berlin: Springer Verlag.

[Sadek 1991] Sadek, D. 1991. Dialogue acts are rational plans. *Proceedings of the ESCA/ETRW Workshop on the Structure of Multimodal Dialogue*, Maratea.

[Schank & Abelson 1977] Schank, R. and R. Abelson. 1977. *Scripts, Plans, Goals and Understanding.* Hillsdale, New Jersey: Lawrence Erlbaum.

[Scott & Kamp 1995] Scott, D., Kamp, H. 1995. Dialogue Modelling. In Cole, R. A., Mariani, J., Uszkoreit, H., Zaenen, A., Zue, V., Varile, G., Zampolli, A. (eds.). *Survey of the State of the Art in Human Language Technology.* http://cslu.cse.ogi.edu/HLTsurvey/

[Searle 1969] Searle. J. R. 1969. *Speech Acts.* Cambridge: Cambridge University Press.

[Seneff 1992] Seneff, S. 1992. A Relaxation Method for Understanding Spontaneous Speech Utterances. In *Proceedings of the Speech and Natural Language Workshop*, 299-304.

[Seneff *et al* 1998] Seneff S., Hurley, E., Lau, R., Pao, C., Schmid, P., & Zue V., 1998. Galaxy-II: A Reference Architecture for Conversational System Development. Proc. ICSLP 1998, Sydney, Australia, November 1998. Available at http://www.sls.lcs.mit.edu/sls/publications/index.html.

[Seneff *et al* 1999] Seneff, S., Lau, R., & Polifroni J. 1999. Organization, Communication and Control in the GALAXY-II Conversational System. *Proc. Eurospeech 99*, Budapest, Hungary, September 1999. Available at http://www.sls.lcs.mit.edu/sls/publications/index.html.

[Shieber 1986] Shieber, S. M. 1986. *An Introduction to Unification–based Approaches to Grammar.* CSLI Lecture Notes 4. Stanford, California: Center for the Study of Language and Information.

[Sutton *et al* 1996] Sutton S., Vermeulen, P., de Villiers, J., Schalkwyk, J., Fanty, M., Novick, D., Cole, R. 1996. *Technical Specification of the CSLU Toolkit.* Technical Report No. CSLU-013-96, Center For Spoken Language Understanding, Oregon Graduate Institute of Science & Technology.

[Torre, Amores & Quesada 2000] Torre, D., J. G. Amores & J. F. Quesada. 2000. *User Requirements on a Natural Command Language Dialogue System.* Deliverable 3.1. Siridus project.

[Trains Web Page] TRAINS project web page, http://www.cs.rochester.edu/research/trains. University of Rochester, Computer Science.

[Traum *et al* 1999] Traum, D., Bos, J., Cooper, R., Larsson, S., Lewin, I., Matheson, C., Poesio, M.. 1999. *A model of dialogue moves and information state revision.* TRINDI (LE4-8314) Deliverable D2.1, November 1999.

[Trips Web Page] TRIPS project home page, http://www.cs.rochester.edu/research/trips. University of Rochester, Computer Science.

[Xu & Rudnicky 2000] Xu, W., Rudnicky, A. 2000. Task-based dialog management using an agenda. *ANLP / NAACL 2000 Workshop on Conversational Systems*, pp. 42-47.

[Zue & Glass 200000] Zue, V., Glass, J. 2000. Conversational Interfaces: Advances and Challenges. Invited Paper, *Special issue on Spoken Language Processing, Proc. IEEE*, **88(8)**, pp. 1166-1180.