Dialogue Management in a Home Machine Environment: Linguistic Components over an Agent Architecture

José F. Quesada, Federico García, Esther Sena, José Angel Bernal, Gabriel Amores
Grupo de Investigación Julietta
Universidad de Sevilla

Abstract This paper presents the main characteristics of an Agent–based Architecture for the design and implementation of a Spoken Dialogue System. From a theoretical point of view, the system is based on the Information State Update approach, in particular, the system aims at the management of Natural Command Language Dialogue Moves in a Home Machine Environment. Specifically, the paper is focused on the Natural Language Understanding and Dialogue Management Agents, and discusses their integration over a global agent architecture (which includes Action and Knowledge Managers, Speech Input/Output components and HomeSetup controllers). [1]

1 Introduction

Spoken Dialogue Systems have become one of the most prominent research areas in Computational Linguistics. Over the last few years, several projects all over the world have focused on this topic. This paper describes the main characteristics of a spoken dialogue system aimed at the control of a lightning scenario in a home machine environment.

From a functional point of view, the system may be described as a:

- Spoken Dialogue System, which acts as a natural language interface between users and
- Home Devices installed in a Home Machine Environment
- designed and implemented over an Agent Architecture (specifically, OAA - Open Agent Architecture [Martin *et al* 99]).

Section 2 describes the overall architecture and agents, mainly the HomeSetup, KnowledgeManager, ActionManager, Text-To-Speech and Dialogue Manager Agents.

One of the main goals of the Natural Language Understanding component of the system is both to detect the command and the device that the user wants to control and to set the level or percentage that the user specifies at a precise moment. For instance, the sentence *"Set the kitchen light at level 5"* , must be transformed into a semantic representation, where the main features will be:

- **Command**:Set
- **Device**: Light
- **Location**: Kitchen
- **Level**:5

Taking this information as its input, the Dialogue Manager must be able to detect if the semantic structure is:

- Functionally complete (Which light do you want to turn on if you have just received *"Switch on the light"*?)
- Pragmatically coherent and technically possible (you cannot switch on a door, for instance).

Besides, the Dialogue Manager must clarify ambiguous expressions, such as *"Switch on the living room light"*, when there are several lights in the living room.

In order to solve these problems, the Dialogue Manager is closely connected to both the Natural Language Understanding and the Knowledge Manager modules.

[1] This work has been supported by D'Homme (Dialogues in the Home Machine Environment), EC Project IST-2000-26280. The authors would like to thanks the rest of the project partners for their comments and suggestions on the work.

Next, we present the linguistic components of the architecture, that is, the Natural Language Understanding and the Dialogue Manager Agents. At this level, it is worth noting that the system allows the following functionality:

- Natural Command Languages
- Expectations
- Multiple Commands
- Default Interpretation

A detailed description of the kernel of the Dialogue Manager may be found in [Quesada *et al* 00, Quesada, Torre & Amores 2000], including the specification language for the description of dialogue systems and the strategies for the implementation of the previous functions, mainly the manipulation of expectations and the use of default interpretations.

Section 3 describes the Natural Language Understanding module, paying special attention to the grammatical level: a unification–based semantic–oriented grammar specification. The interface between the Natural Language Undertanding and the Dialogue Manager agents is based on the DTAC protocol ([Amores & Quesada 2000]). Section 4 includes an interesting set of examples incorporating for each example the DTAC representation of the sentence obtained by the grammar presented in section 3.

Finally, section 5 highlights one of the main challenges of the system: to obtain a natural language interface to the Home Machine Environment. This means that a user must be able to install, plug or unplug, and configure new or previous devices and to work and talk with them on the fly. This section includes some examples which illustrates the main functions supported by the system so far.

2 *Overall Architecture and Agents*

The system has been implemented using the Open Agent Architecture [Martin *et al* 99]. Figure 1 illustrates the main agents apart from the Facilitator, the central agent in charge of the communication between the rest of agents.

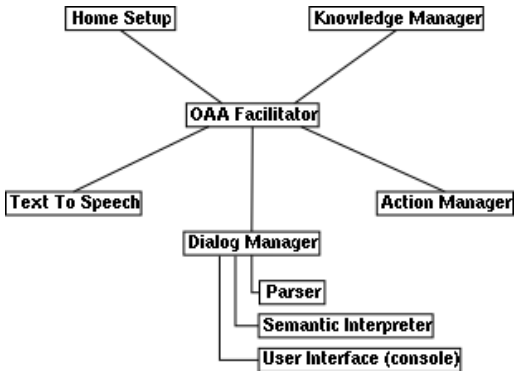

Figure 1: Overall agent architecture.

- **HomeSetup**: This agent is in charge of the simulation of a Home Machine Environment. This agent mainly allows the installation of new devices and their configuration. As part of the configuration the user may instantiate the set of descriptors related to the device.
- **KnowledgeManager**: This agent contains both the static (structure of the house, general ontology, ...) and dynamic (installation, configuration and status of the devices) knowledge involved in the task.
- **ActionManager**: The current devices will support the actual functionality of the system. This functionality (turn on/off the device, ask the current status, change the level of a dimmerable device, ...) must be accessed through the HomeSetup agent, in charge of the simulation and monitoring of the scenario. Nevertheless the devices incorporate primitive functions to the system. The ActionManager incorporates new higher level functionality that will be translated into sequences of primitive actions.
- **Text-to-Speech**: In charge of the generation of the speech output. In order to do so we have developed an OAA-wrapper for Festival, allowing in this way the use of this system as part of the general architecture.
- **Dialogue Management**: This agent includes the User Interface (which controls at a high level the communication with the user), the Parser, the Semantic Interpreter and the Dialogue Manager components.

The Knowledge Manager is in charge of storing and organizing the information related to the devices installed in the house regarding ubication, characteristics and relations with other devices. The approach adopted for organizing this information is based on the conformation of a semantic network. The properties of this network are:

- Nodes organized in a *graph* topology
- Non- directed connections between nodes.
- Non- conexed.
- Non- cyclical.
- Parent-child relation defined in each conection between nodes.

Each node contains a list of components. A component consists of a pair of strings: a label and a value. We can describe thereby the structure of the network as having two relations:

- A component belongs to a node.

$$component \in node$$

- A node includes other node.

$$node1 \subset node2$$

  e.i. $node1$ is parent of $node2$.

Let see an example of semantic network:

```
ground < indoors
ground < outdoors

indoors < kitchen
indoors < den
indoors < corridor
indoors < living_room

outdoors < front
outdoors < back
```

where the symbol $\subset$ is represented by <. The figure 2 represents the graph conforming the network specified.

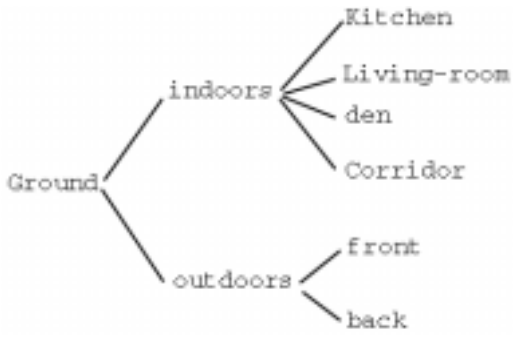

Figure 2: Graph representation of a semantic network example.

3 Natural Language Understanding

Dialogue systems enable users to interact with computer systems via natural and intelligent dialogues, as they would with human agents. Development of such systems requires a wide range of speech and language technologies, including natural language and dialogue processing, to determine the meanings and intentions of the utterances and to generate a text-to-text interaction between the user and the system.

As regards this document, what we have done is to develop an interface for both English and Spanish for the specification of different functions in an intelligent house scenario. In order to do so, we have made use of Natural Command Languages. Natural Command Languages comprise the set of input and output natural language expressions which are acceptable in a given application domain. The most outstanding characteristics of these NCLs are the use of natural language vocabulary and syntactic structure. Also, NCLs reflect natural syntactic order and more precisely the patterns of spoken language. In fact, they are a sub-language, with reasonably clear boundaries but also one in which new functionality can be added without altering the structural properties of the language.

Next subsection presents the main rules of the unification- based grammar that has been implemented. At this level it is worth noting that the grammar incorporates Non–terminal typification, and includes rules to construct a Natural Command Language Dialogue representation model, based on Dialogue Moves where DMOVE represents the dialogue move itself [Amores & Quesada 2000], TYPE the specific type of dialogue move, ARGS the

functional subcategorization model of the type of dialogue move and CONT the actual contents of terminal structures.

3.1 Unification–based and Semantic–oriented Grammar

```
RootsOfGrammar:
        // Parameters
        Value
        Device
        // Commands
        Command
```

This is a configuration parameter used by the parser, in fact, it is the only one that the parser uses. On the other hand, the unifier is the one in charge of the rest of the parameters. By means of these RootsofGrammar, all grammar symbols that may be treated as roots are specified. This in turn let us have grammars with several root symbols.

```
NonTerminalTypes:
        // Parameters
        Device =
                    (DMOVE: specifyParameter,
                     TYPE: Device)
        Number =
                    (DMOVE: specifyParameter,
                     TYPE: Value,
                     SUBTYPE: Number)
        Level =
                    (DMOVE: specifyParameter,
                     TYPE: Value,
                     SUBTYPE: Level)
        Percentage =
                    (DMOVE: specifyParameter,
                     TYPE: Value,
                     SUBTYPE: Percentage)

        // Commands
        CommandOn  =
                    (DMOVE: specifyCommand,
                     TYPE: CommandOn,
                     ARGS: [Device])
        CommandOff =
                    (DMOVE: specifyCommand,
                     TYPE: CommandOff,
                     ARGS: [Device])
        CommandIst =
                    (DMOVE: specifyCommand,
                     TYPE: CommandISt,
                     ARGS: [Device])
        CommandSet =
                    (DMOVE: specifyCommand,
                     TYPE: CommandSet,
                     ARGS: [Device,Value])
        CommandInc =
                    (DMOVE: specifyCommand,
                     TYPE: CommandInc,
                     ARGS: [Device,Value])
        CommandDec =
                    (DMOVE: specifyCommand,
                     TYPE: CommandDec,
                     ARGS: [Device,Value])
        AskHelp =
                    (DMOVE: askHelp,
                     TYPE: AskHelp)
        Quit =
                    (DMOVE: quit,
                     TYPE: Quit)
```

Under the label NonTerminalTypes, we have both parameters and commands. Both types will be the ones that will appear in the DTAC structures that the Dialogue Manager will take as its input. This can be clearly noticed in section 4. As regards parameters, there are four, namely device, percentage, number and level. The label device may stand for words such as “kitchen”, the label number stands for a plain number such as “5”, and the word level along with percentage, indicate either a level such as “at level 5”, or a percentage such as “to 25 percent” respectively. These parameters will be instantiated as the arguments of the commands, but they will be enclosed under the label “value”, that is, this value together with the device, will be what will actually appear as the argument of the command. Therefore, percentage, number and level will be value subtypes. We have created this construct to avoid ambiguities in both the grammar and the semantic representation. Commands are divided into eight types. The first group of commands (CommandOn, CommandOff, and CommandIst) takes a device as its argument. At this point, the user may ask the system to switch on a light, to switch it off, or he may just be asking for the status of a light. The second set of commands are the ones used with dimmerable devices, therefore they take a device plus a value. They are used to set a light at any level, percentage or number, to increase it, or to decrease it. The third set consists of two commands, one of them for asking help, and the other one for quiting the system. As it can be noticed below, the do not require any parameters.

```
////////////    PARAMETERS
(A100 : AuxDevice -> LAuxDevice)
                { @up = @self-1; }
(A101 : AuxDevice -> AuxDevice LAuxDevice)
                { @up = @self-2; }
(P100 : Device -> LKeyDevice )
                { @up = @self-1; }
(P102 : Device ->AuxDevice LKeyDevice )
                { @up = @self-2; }
(L100 : AuxPerc -> LAuxPerc )
                { @up = @self-1; }
(L101 : AuxLevel -> LAuxLevel)
                { @up = @self-1; }
(L102 : Number -> LNumber)
                {@up = @self-1; }
(L104 : Level -> AuxLevel LNumber )
                { @up = @self-2; }
```

```
(L105 : Level -> LNumber AuxLevel )
                { @up = @self-1; }
(L106 : Percentage -> LNumber AuxPerc )
                { @up = @self-1; }
(L107 : Percentage -> AuxPerc LNumber )
                { @up = @self-2; }
(L110 : Value -> Number )
                { @up = @self-1; }
(L111 : Value -> Level )
                { @up = @self-1; }
(L112 : Value -> Percentage )
                { @up = @self-1; }
(L120 : Args -> Device )
                { @up.Device = @self-1; }
(L121 : Args -> Value )
                { @up.Value = @self-1; }
```

In this part of the grammar, we can see the rules with the functional equations associated to them. All of them are parameter rules. Words beginning with L are terminal categories. For instance: LAuxDevice, LAuxPerc,LNumber,LKeyDevice. We have words such as "level", "percentage" and "light", which belong to the corresponding categories "AuxLevel", "AuxPercentage" and " AuxDevice". Also, as device is a non-terminal type, "LkeyDevice" is the lexical category standing for words such as "kitchen". An Args can be either a value or a device, and a value can be either a level or a percentage. The creation of these constructs and the way in which the rules have been implemented, allows us both to have any kind of word order and to avoid grammatical ambiguity.

```
///////////    COMMANDS
(C200 : CommandOn -> LCommandOn)
                { @up = @self-1; }
(C201 : CommandOff -> LCommandOff)
                { @up = @self-1; }
(C213 : CommandIst -> LAuxIstB)
                { @up.TYPE =a CommandISt; }
(C214 : CommandIst -> LAuxIstA LAuxIstB)
                { @up.TYPE =a CommandISt; }
(C215 : CommandIst -> LAuxIstB LAuxIstC)
                { @up.TYPE =a CommandISt; }
(C202 : CommandSet -> LCommandSet)
                { @up = @self-1; }
(C203 : CommandInc -> LCommandInc)
                { @up = @self-1; }
(C204 : CommandDec -> LCommandDec)
                { @up = @self-1; }

(C211 : XCommand -> CommandOff )
                { @up = @self-1; }
(C212 : XCommand -> CommandIst )
                { @up = @self-1; }
(C213 : XCommand -> CommandOn )
                { @up = @self-1; }
(C220 : XCommand -> CommandSet )
                { @up = @self-1; }
(C221 : XCommand -> CommandInc )
                { @up = @self-1; }
(C222 : XCommand -> CommandDec )
                { @up = @self-1; }
(C223 : Command -> XCommand )
                { @up = @self-1; }
(C224 : Command -> XCommand Args)
                { @up = @self-1;
                  @up = @self-2; }
```

The way in which the command rules have been arranged allows us to have any combination of words, no matter the order in which they are either said or typed. This is a crucial point, specially if we have into account that this is intended to be spoken, and spoken language is by no means predictable or fixed as regards word order. If we start from the lexical categories, which are the ones beginning with L, we get the Command*. The * stands for On, Off,Set, Inc, Dec. As we can notice, all these are commands with arguments and they can be of the same type as the abovementioned parameters.They can be described as specific commands. We may have any combination of words, specifically when the user wants to ask for the status of a light. He may simply say "Estado de la cocina", or "en que estado se encuentra la cocina".In the later, the two keywords would be "estado" and "encuentra", and in the former, the keyword would be would be "estado". All the Command* are XCommands. These are commands without arguments. Finally, the XCommand is inserted within a more general command which is just a Command.

```
///////////    ERROR REPAIR

(N001 : Command ->  Command No Args)
    { @up = @self-1;
      @if (@self-3.Device) @then {
              @up.RepairedDevice = @self-1.Device;
              @up.Device =a @self-3.Device;
      }
      @if (@self-3.Value) @then {
              @up.RepairedValue = @self-1.Value;
              @up.Value =a @self-3.Value;
      }
    }

(N002 : Command ->  Command No Command)
    { @up = @self-3;
      @if (!(@self-3.Device) &&
            (@self-1.Device)) @then {
            @up.Device = @self-1.Device;
      }
      @if (!(@self-3.Value) &&
            (@self-1.Value)) @then {
            @up.Value = @self-1.Value;
      }
      @up.RepairedCommand = @self-1;
    }
```

These rules are used when the user makes an error and he inmediately corrects himself. He may be repairing an argument which may

be a level, percentage, or a device, he may be changing the command, that is to say, he may be asking the system to turn off the kitchen light when he has just realized that that light is already off, and he may be changing both the command and the argument. This is clearly exemplified in section 4, in examples 4, 5 and 6. In number 4, the user is changing the command, in number 5, he is changing both the device and the value, and in number 6, he is repairing the command and the value.

4 Semantic representation of dialogue moves

Next, we present some examples which illustrate the use of the previous grammar. These results have been obtained with the Episteme NLP tool ([Quesada & Amores 2000]).

The goal of this section is to illustrate the generality and simplicity of the representation formalism with a wide range of phenomena: incomplete commands, commands with different arguments, different types of arguments (numbers, descriptors, ...) and error repairs, both of commands and parameters:

1. **Turn on**

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & SwdCommandOn \\ ARGS & : & [SwdDevice] \end{bmatrix}$$

2. **Turn on the kitchen light**

$$\begin{bmatrix} DMOVE: & : & specifyCommand \\ TYPE: & : & SwdCommandOn \\ ARGS & : & [SwdDevice] \\ CONT & : & \\ SwdDevice & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & SwdDevice \\ CONT & : & kitchen \end{bmatrix} \end{bmatrix}$$

3. **Dim the kitchen light down to 25 percent**

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & SwdCommandDec \\ ARGS & : & [SwdDevice], [SwdValue] \\ CONT & : & \\ SwdDevice & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & SwdDevice \\ CONT & : & kitchen \end{bmatrix} \\ SwdValue & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & SwdValue \\ CONT & : & 25 \\ SUBTYPE & : & SwdPercentage \end{bmatrix} \end{bmatrix}$$

4. **Increase the kitchen lamp no no decrease it**

$$\begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & SwdCommandDec \\ ARGS & : & [SwdDevice], [SwdValue] \\ CONT & : & \\ SwdDevice & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & SwdDevice \\ CONT & : & Kitchen \end{bmatrix} \\ ErrorCommand & : & \begin{bmatrix} DMOVE & : & specifyCommand \\ TYPE & : & SwdCommandInc \\ ARGS & : & [SwdDevice], [SwdValue] \\ CONT & : & \\ SwdDevice & : & \begin{bmatrix} DMOVE & : & specifyParameter \\ TYPE & : & SwdDevice \\ CONT & : & Kitchen \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

5. **Increase the lamp in the kitchen to level 10 no no no the light in the dinning room to 25 percent**

$$\left[\begin{array}{lcl}
DMOVE & : & specifyCommand \\
TYPE & : & SwdCommandInc \\
ARGS & : & [SwdDevice], [SwdValue] \\
CONT & : & \\
SwdDevice & : & \left[\begin{array}{lcl} DMOVE & : & specifyParameter \\ TYPE & : & SwdDevice \\ CONT & : & living_room \end{array}\right] \\
SwdValue & : & \left[\begin{array}{lcl} DMOVE & : & specifyParameter \\ TYPE & : & SwdValue \\ CONT & : & 25 \\ SUBTYPE & : & SwdPercentage \end{array}\right] \\
ErrorSwdDevice & : & \left[\begin{array}{lcl} DMOVE & : & specifyParameter \\ TYPE & : & SwdDevice \\ CONT & : & Kitchen \end{array}\right] \\
ErrorSwdValue & : & \left[\begin{array}{lcl} DMOVE & : & specifyParameter \\ TYPE & : & SwdValue \\ CONT & : & 10 \\ SUBTYPE & : & SwdLevel \end{array}\right]
\end{array}\right]$$

6. **Set the light in the kitchen at level 5 no no no increase it to 25 percent.**

$$\left[\begin{array}{lcl}
DMOVE & : & specifyCommand \\
TYPE & : & SwdCommandInc \\
ARGS & : & [SwdDevice], [SwdValue] \\
CONT & : & \\
SwdValue & : & \left[\begin{array}{lcl} DMOVE & : & specifyParameter \\ TYPE & : & SwdValue \\ CONT & : & 25 \\ SUBTYPE & : & SwdPercentage \end{array}\right] \\
SwdDevice & : & \left[\begin{array}{lcl} DMOVE & : & specifyParameter, \\ TYPE & : & SwdDevice \\ CONT & : & kitchen \end{array}\right] \\
ErrorCommand & : & \left[\begin{array}{lcl} DMOVE & : & specifyCommand, \\ TYPE & : & SwdCommandSet \\ ARGS & : & [SwdDevice, SwdValue] \\ CONT & : & \\ SwdDevice & : & \left[\begin{array}{lcl} DMOVE & : & specifyParameter \\ TYPE & : & SwdDevice \\ CONT & : & kitchen \end{array}\right] \\ SwdValue & : & \left[\begin{array}{lcl} DMOVE & : & specifyParameter \\ TYPE & : & SwdValue \\ CONT & : & 5 \\ SUBTYPE & : & SwdLevel \end{array}\right] \end{array}\right]
\end{array}\right]$$

5 Natural Language Dialogues

Finally, we present some dialogue examples which illustrate the main functions of the system from a linguistic point of view:

5.1 Natural Command Languages and Expectations

Scenario: Two lights: one in the kitchen and the other in the corridor. Both are turned off at the beginning

Dialogue 1

- User: Turn on.
- System: What device would you like to turn on?
- U: The kitchen light.
- S: The light in the kitchen has just been turned on.

In this example, the system detects that in order to turn on a device, the user must supply the descriptor of the actual device. This functionality is implemented as a expectation of the dialogue manager ([Quesada, Torre & Amores 2000]).

5.2 Multiple Commands

Dialogue 2

- U: Turn off the kitchen and switch the corridor on.
- S: The light in the kitchen has just been turned off.
- S: The light in the corridor has just been turned on.

Althought the output generated by the system must be improved in order to obtain just one sentence, this example illustrates one of the most complex phenomena in dialogue management: the user asks the system to execute several commands in one utterance.

5.3 Expectations, previous commands and default interpretation

Dialogue 3

- U: Turn on the corridor.
- S: The light in the corridor is already on.
- U: Sorry, the kitchen light.
- S: The light in the kitchen has just been turned on.

This example shows the integration of the dialogue manager with the knowledge manager, which has detected that the corridor light was already on. Besides, the example illustrates an ambitious dialogue strategy: the default interpretation of a parameter as part of a previous command. In this scenario, the specification of the parameter is analised as an expectation of the previous command.

5.4 Plug & Play

Dialogue 4

- U: What is the current state of the living room light?
- S: The light in the living room is not available at the moment.
- *Using the HomeSetup Agent, the user installs, configures and plugs a new light at the living room.*
- U: Now, turn the living room on.
- S: The light in the living room has just been turned on.

Taking into account that the 4 previous subdialogues belong in fact to just one dialogue obtained from the real system, it is worth noting the ability to support plug & play capabilities without restarting the dialogue manager nor the rest of agents.

6 Conclusion

A dialogue system designed to support natural and flexible dialogues intended to control a lightning system in a home machine environment has been presented. The paper has concentrated on the linguistic components of the agent–based architecture: the grammar and the semantic representation of complex sentences. A set of dialogue examples illustrating the linguistic capabilities of the system have been presented and described.

References

[Amores and Quesada 1997] Amores, J.G. and J.F. Quesada. 1997. Episteme. *Procesamiento del Lenguaje Natural* **21**. pp. 1-16.

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

[Fernández & Quesada 99] Fernández, G. & Quesada, J. F. 1999. Delfos: Un Modelo Basado en Unificación para la Representación y el Razonamiento en Sistemas de Gestión de Diálogo. *Procesamiento del Lenguaje Natural*, 67–74.

[Kamp & Reyle 93] Kamp, H. & Reyle, U. 1993. *From Discourse to Logic*. Dordrecht: Kluwer.

[Kirchner 90] Kirchner, C. ed. 1990. *Unification*. San Diego, California: Academic Press Inc.

[López & Quesada 99] 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.

[Noord *et al* 98] 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.

[Martin *et al* 99] Martin, D. L., Cheyer, A. J., Moran, D. B. (1999). The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, vol 13, pp. 91–128, January–March 1999.

[Quesada *et al* 00] Quesada, J.F., Amores, J.G., Fernández, G., Bernal, J.A., and 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 & Amores 2000] Quesada, J. F. & Amores, J. G. (2000). *Diseño e Implementación de Sistemas de Traducción Automática*. Universidad de Sevilla: Secretariado de Publicaciones.

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

[Rozenberg & Salomaa 97] Rozenberg, G. & A. Salomaa. eds. 1997. *The Handbook of Formal Languages*. Berlin: Springer Verlag.

[Smolka and Aït–Kaci 1990] Smolka, G. and H. Aït–Kaci. 1990. Inheritance Hierarchies: Semantics and Unification. In [Kirchner 90], pages 489–516.