Speaking with Google Calendar

Pedro Arez

Dissertação para obtenção do Grau de Mestre em Engenharia Informática e de Computadores

July 22, 2009
Agradecimentos

This master thesis suffered a long way until what it is today, it involved directly or indirectly many persons of many countries and it has been the middle of many arguments. First of all, I would like to thanks the patience, work and dedication of my dissertation coordinator, Nuno Mamede who always tried to accommodate and support my decisions even if they could imply radical changes to this master thesis or even its cancelation.

To my friend and colleague, Filipe Martins who in many occasions and many ways helped me accomplish this work, in despite of the thousands of emails and calls. Not forgetting, Ana Mendes, Tiago Luís, Joana Pardal, David Rodrigues, Carlos Mendes and all the other L²F colleagues, professors and personnel for the direct and indirect support throughout the development of this work. I extend this thanks to Professor R.J. Beun, of the University of Utrecht for supporting for more than one month the belief that I would be able to extend this research project to other university and finish my master in another country.

I cannot forget to thanks all the friends who put up with my problems and me more than once through out the past year and half, Hernâni Freitas, Duarte Almeida, Paulo Gomes, Simon Sørensen and Francisca da Silva. I want to give a special thanks to my friend Saverio Pasini, for patiently and without complaining helping me with the English review of this work.

I dedicate this work to both my parents and the rest of my family, for the blind faith towards all my academic studies.

Lisboa, July 22, 2009
Pedro Jorge Fino da Silva Arez
Resumo

Este projecto consiste num sistema de diálogo em português para interagir com um calendário online, o Google Calendar. Este sistema de diálogo usa a cadeira de processamento L^2F NLP Chain para reconhecer e normalizar expressões temporais. Esta dissertação regista todo o processo de desenvolvimento, incluindo: (i) o estudo de outros sistemas de diálogo no domínio dos calendários; (ii) o estudo de sistemas de diálogo desenvolvidos tendo como base a plataforma DIGA; (iii) o desenho de uma solução; (iv) a implementação e avaliação da solução definida. Neste trabalho a plataforma DIGA é usada numa perspectiva do cliente, isto foi feito de forma a ser possível identificar problemas relacionados com o desenho ou desenvolvimento da plataforma e providenciar sugestões para o seu contínuo desenvolvimento.

Este sistema de diálogo suporta quatro tipos de interacção: agendamento, listagem, remoção e procura de eventos num calendário associado ao utilizador. Adicionalmente, o suporte de um largo espectro de expressões temporais diminui as limitações de diálogo entre o sistema de diálogo e os seus utilizadores. Este trabalho teve duas fases de desenvolvimento onde foram obtidos resultados variáveis em termos da satisfação do utilizador. Este trabalho resultou num sistema de diálogo experimental, capaz de reconhecer com sucesso expressões temporais e interage eficazmente com o Google Calendar.
Abstract

This research project aims at developing a portuguese dialogue system to interact with an online calendar application, the Google Calendar. The dialogue system uses the L2F NLP Chain in order to be able to automatically recognize and normalize temporal expressions. This thesis traces the entire development process, including: (i) the study of other dialogue systems that deal with the calendar domain; (ii) the study of dialogue systems developed based on the DIGA Framework; (iii) the design of a solution; (iv) the implementation and evaluation of the devised solution. In this work the DIGA Framework is used on a client perspective, in order to identify possible design and development problems and suggest improvements to the development of the framework.

This dialogue system supports four types of interaction with the calendar: scheduling, listing, deleting and searching of events in an online calendar associated with the user. Additionally, the support of a wide range of temporal expressions diminishes the constrains over the dialogue between users and the dialogue system. This work went through two evaluation phases which showed mixed results towards the user satisfaction. This work resulted in a proof of concept dialogue system which successfully recognizes temporal expressions in portuguese and is able to interact effectively with Google Calendar.
Keywords

Dialogue systems
Calendar
DIGA Framework
Dialogue Manager
# Index

## 1 Introduction
1.1 Context ......................................................... 2
1.2 Objectives ......................................................... 3
1.3 Technology used ..................................................... 4
  1.3.1 Google Calendar ............................................. 4
    1.3.1.1 Google API ............................................. 6
  1.3.2 L²F Natural Language Processing Chain ....................... 7
  1.3.3 DIGA Framework ............................................. 8
    1.3.3.1 Dialogue Manager ....................................... 9
      1.3.3.1.1 Domains ........................................... 10
      1.3.3.1.2 Devices ........................................... 10
    1.3.3.2 Speech Recognizer: ...................................... 12
    1.3.3.3 Speech Synthesizer ..................................... 13
1.4 Thesis Outline .................................................. 13

## 2 State of the Art
2.1 DIGA based dialogue systems ....................................... 15
  2.1.1 Voice Control of a Hi-Fi .................................... 15
  2.1.2 Meteo: A telephone-based Portuguese Conversation System in Weather Domain .................................................. 15
  2.1.3 The Electronic Assistant ...................................... 17
  2.1.4 Ambrósio - The Virtual Butler ................................ 18
  2.1.5 Cooking Assistant ........................................... 18
2.1.6 Lisa, the Personal Assistant ....................................................... 19
  2.1.6.1 Architecture ................................................................. 20
2.1.7 Phone banking ................................................................. 20

2.2 Calendar Systems with Dialogue Interfaces .................................. 21
  2.2.1 SCHED - Simplified Calendar Handling through English Dialogue .... 21
  2.2.2 RavenCalendar: A Multimodal Dialog System for Managing a Personal Calendar 22
  2.2.3 You’ve Got Mail! Calendar, Weather and More: Customizable Phone Access to Personal Information ................................................................. 24
  2.2.4 SpeechActs PROJECT .......................................................... 26
  2.2.5 System and Features Comparison ............................................ 28

3 Google Calendar Voice Interface .................................................... 31
  3.1 System’s Architecture ............................................................ 31
  3.1.1 The integration with the Dialogue System Framework - DIGA ........... 31
      3.1.1.1 Developed Components .................................................. 31
          3.1.1.1.1 Google Calendar Device .......................................... 32
          3.1.1.1.2 Parsers ................................................................. 34
          3.1.1.1.3 Services definition ............................................... 34
      3.1.1.2 Module integration ....................................................... 36
  3.1.2 System Modules .................................................................... 36
      3.1.2.1 Temporal Expression Normalization Module .......................... 36
          3.1.2.1.1 L²F NLP Chain web service and connection layer ........... 38
          3.1.2.1.2 Post-Processing Layer .............................................. 38
      3.1.2.2 Temporal Expressions Processing Module ........................... 39
      3.1.2.3 Operations Module ....................................................... 40
  3.2 Developed Functionalities ......................................................... 40
      3.2.1 Creating Events ............................................................... 41
5 Conclusions and Future Work

5.1 Meeting the Objectives .................................................. 63

5.2 Problems ................................................................. 64

5.2.1 DIGA Framework ..................................................... 64

5.2.1.1 Dialog Manager ................................................... 64

5.2.1.1.1 Sentence generation .......................................... 64

5.2.1.1.2 Parser development .......................................... 64

5.2.1.1.3 Dialogue synchronization .................................... 65

5.2.1.1.4 Implicit confirmations ....................................... 65

5.2.1.2 Speech Recogniser ................................................ 65

5.2.1.2.1 Short words as no speech .................................... 66

5.2.1.2.2 Recognition problems ....................................... 66

5.2.2 L2F NLP Chain ......................................................... 66

5.2.2.0.3 Normalisation issues .......................................... 66

5.2.2.0.4 Performance issues ........................................... 67

5.3 Future Research and possible developments .......................... 67

5.3.1 User identification and authentication .............................. 67

5.3.2 Taking a step further ................................................ 68

5.3.3 L2F NLP chain web-service 2.0 ..................................... 68

5.3.4 Going Mobile .......................................................... 69

5.4 Final Remarks ............................................................ 70
It has been thirty years since George Heidorn (Heidorn, 1978) wrote about the development of computer systems designed to accomplish daily simple office tasks, like scheduling a meeting, booking a room or checking someone’s schedule. A system that is able to perform all these tasks would result in a huge breakthrough, not only because of the automation possibilities, but also from the reduction of the system learning curve. The concept “office of the future” was, and still is, of a considerable interest to the research world. There is a continuous search for the automation of the office domain tasks, with the objective of offloading office workers from redundant tasks that are currently processed manually.

The obsession on life planning turned the personal agenda into a very popular personal tool, from its paper version to the electronic one. The personal agenda is the everyday tool of any organised person. From meetings to birthdays, dinners parties, everything can be meticulously scheduled in an agenda, providing a daily, weekly or monthly plan of activities for its owner. The agenda concept has evolved considerably in the last two decades. However, it was the mass generalisation of the cell phone and the development of web2.0 applications, with the consequent integration of mobile/SMS capabilities, that resulted in the agenda we know today, an almost ubiquitous electronic agenda.

Google Calendar\(^1\) presents itself as one of the best web2.0 agenda/calendar applications on the market. This web application is able to provide all the typical agenda/calendar functionalities, as well as, sharing functionalities, SMS alerts and mobile support. Google Calendar can be accessed from any device that possesses an internet browser with internet access, or alternatively, can be synchronised with off-line calendar applications. However, Google Calendar fails to provide any kind of multi modal interface that allows users with disabilities to access its functionalities. Facing this, we considered developing a voice interface for Google Calendar, as the challenge of turning Google Calendar into a language enabled agenda and opening it to a broader range of devices and users. The integration of a voice interface, with Google Calendar, allows users with more demanding accessibility needs to use the most common of Google Calendar through other communication channels and environments. The integration of such voice interface with a phone system or a voip\(^2\) PBX\(^3\) would allow anyone, with or without disabilities to use the Google Calendar through a phone.

\(^1\)http://calendar.google.com
\(^2\)Voice over Internet Protocol (VoIP)
\(^3\)A private branch exchange (PBX) is a telephone exchange that serves a particular business or office


1.1 Context

The calendar concept has been explored in more than one generation of dialogue systems (Huang et al., 2001) (Perez-Quinones et al., 2004). Prototypes and proof of concepts were developed carrying voice functionalities for data input, functionality control or menu navigation. Most of these systems were developed in order to evaluate the current achievements or to find major problems and flaws.

The main functionalities of an electronic agenda are, generally speaking, scheduling, listing and deleting events, where editing, rescheduling and searching events are secondary functionalities. Facing this, we can assume that the minimal functionalities that should be present in any agenda system, are the options of scheduling, listing and deleting events. However, appointment scheduling applications are an example where only one of this functionalities is present. In scheduling applications, the user is only given the possibility to schedule an appointment, without the possibility to list the appointments that were already scheduled. Automated Appointment Scheduling (Fanty et al., 1995) is the example of an scheduling application that is briefly analysed in this thesis:

**Automated Appointment Scheduling** This spoken language system is capable of scheduling appointments over the phone. The user is able to call the system and through a natural flow of dialogue, the system tries to fulfils the user’s request according to the calendar of available service time slots. The system relies on a semantic parser to map the user’s input and in a basic temporal expression normalisation system to process the user’s input. The main objective of this project is to provide techniques for prevention, detection, recovery of dialogue errors or inconsistencies, and to overcome speech recognition mistakes, while handling the scheduling application.

The PDA\(^4\) can be seen as a platform for a voice operated agenda due to its multi modal support, however in most of the cases it still does not have the necessary capacity. In any multi-modal agenda, basic agenda functionalities are supported by the voice interface, but in a PDA, the voice functionalities are heavily constrained on the hardware characteristics available at the time of development. This means that usually, due to hardware restrictions, a PDA isn’t able to support a complete voice interface. We came across a multi-modal PDA project that supports agenda functionalities through voice interface. This project does not aim at using voice to fully control the agenda. Instead, voice is only used for the most text intensive tasks, leaving the PDA pen interface for the remaining tasks.

**MiPad: A Next Generation PDA Prototype** MiPad is a wireless Personal Digital Assistant that fully integrates continuous speech recognition and spoken language understanding. It enables users to accomplish many common tasks using a multimodal interface and wireless technologies. In this

---

\(^4\)Personal digital assistant (PDA)
project, the voice features (e.g. speech synthesis and recognition) are a complementary tool to the pen/tap input, meaning that the user cannot use voice as a single form of input. For example, the calendar functionality on the MiPad prototype allows the user to use the microphone to enter data regarding each event (e.g. "Meet with Peter on Friday"), but it does not allow the user to schedule the event, using exclusively the microphone. The main objective of this system is to improve the usability of the PDA, but since the hardware constrains of the voice processing were surpassed using an external full size server, the PDA gained new dependencies on external resources leading to a loss of mobility.

Most of calendar applications with voice interfaces are developed as a proof of concept in order to test theoretical approaches. However, some of this applications are able to provide calendar functionalities through a voice interface in near production environment. You’ve Got Mail is an example of a near production prototype.

G. Heidorn (Heidorn, 1978) is responsible for one of the first studies towards voice interface for online calendars. Despite being mainly a theoretical research, a prototype was developed and tested, resulting in some interesting conclusions. The prototype is able to successfully perform basic calendar operations, but there are consistent problems on how to deal with the numerous ways that date and time information can be expressed. This problem has already been addressed in the English language and is being currently addressed in the Portuguese language (Mamede et al., 2008) as well in this master thesis.

This thesis presents a proof of concept prototype based on DIGA framework. The DIGA framework is a speech dialogue system developed at the L2F Spoken Language Systems Lab in INESC-ID Lisboa. This framework was born with the DIGA project, which had as its main objective the development of a research testbed for communicative interaction, including the development of medium and large vocabulary continuous speech recognition modules, speech synthesis modules, natural language understanding and generation modules. Some prototypes have already been successfully developed with this platform. These will be briefly analysed further on this thesis.

1.2 Objectives

The main objective of this thesis is the creation of a voice interface for the Google Calendar service. This voice interface is based on the DIGA framework, and it will provide the ability to recognise a considerable variety of temporal expressions. The detailed objectives and research questions that this work tries to achieve are:

---

• Contextualize this research project with the dialog system framework and other tools and terms used in throughout research project;

• Produce an overview of the current approaches and dialogue systems that deal specifically with calendars;

• Produce an overview of dialog systems developed using DIGA framework;

• Build a domain model that support most common interactions with an online calendar;

• Develop an interface between the dialogue system framework DIGA and the Google Calendar using Google Calendar API;

• Develop an interface between the DIGA dialogue system framework and the L²F NLP chain;

• Normalise expressions regarding time intervals and time frequencies using the L²F NLP chain;

• Extend the dialogue system’s general grammar to the calendar domain;

• Perform evaluation tests to the prototype.

1.3 Technology used

1.3.1 Google Calendar

Google Calendar, previously known as “CL2”, is a web application offered by Google which provides contact- and time-management functionalities. It’s Graphical User Interface, exemplified in Figure 1.1 is still recognized as one of the most effective Graphical User Interface and one of the best examples of web 2.0 technology. Google Calendar allows its users to synchronise with Gmail, Outlook, PDA, cell-phone contacts and other off-line applications. All the information is stored online, meaning that the calendar and its events can be accessed through the Internet, cell phone or SMS.

Google Calendar provides the following operations:

• Add an event:
  – It is possible to quick add events by clicking in the spot of the calendar and write the event details;
  – A more detailed event creation is also possible using a form, with more details and options being available in the moment of the creation of the event;

---

6 http://calendar.google.com
7 http://calendar.google.com
– Is is also possible to import events from other programs, like Microsoft Outlook, Yahoo Calendar, or create events from events received in the user’s Gmail account.

• Invite and notify:
  – Invite other users, as guests or participants, to an event by adding their email address to the selected event;
  – Guests that were invited can comment an event, directly through their Google Calendar application or through their email;
  – The user can set up reminders that can be sent through email, sms or pop up window.

• Manage Multiple Calendars:
  – Google Calendar supports multiple calendars, so it’s users can have one calendar associated with each aspect of his life;
  – Google Calendar’s users can subscribe to public calendars, which can updated or visualised by other users.
  – It is possible to view Multiple calendars;
  – The user can also colour events or define colours to each event according to its importance or its type.

• Share a Calendar:
  – Its possible to share a calendar to a public group.
The development of an API \textsuperscript{8} turned Google Calendar into a landmark of online calendars. The API allows the integration with almost any software, being supported in many programming languages which results in less technical obstacles for the use of the API while interfacing the Google Calendar with the DIGA framework.

1.3.1.1 Google API

The Google Calendar API is available through a set of libraries. These libraries provide the developer with methods that, asynchronously, send and receive data from the Google Calendar service.

With the Google Calendar API it is possible to control almost all aspects of Google Calendar. After the necessary authentication using Google account credentials, it is possible to execute the following operations:

- Retrieving events
  - Retrieving events without query parameters;
  - Retrieving events for a specified date range;
  - Retrieving events matching a full text query.

- Creating/Editing events
  - Creating single-occurrence events;
  - Creating quick add events;
  - Creating Calendar Gadgets New/;
  - Creating recurring events;
  - Updating events;
  - Deleting events;
  - Performing multiple operations with a batch request;
  - Sharing calendars.
  - Retrieving access control lists;
  - Adding a user to an access control list;
  - Updating a user’s role in an access control list;
  - Removing a user from an access control list.

\textsuperscript{8}Application Programming Interface
• Retrieving calendar lists
  – Retrieving all calendars;
  – Retrieving only calendars that a user owns.

• Managing calendars
  – Creating new calendars;
  – Updating existing calendars;
  – Deleting calendars.

• Managing subscriptions to calendars
  – Adding new subscriptions;
  – Updating calendar subscriptions;
  – Deleting subscriptions.

1.3.2 L²F Natural Language Processing Chain

![Diagram of the L²F NLP chain]

Figure 1.2: Architecture of the L²F NLP chain.

The L²F NLP (Natural Language Processor) chain performs syntactic analysis, chunk identification and dependency extraction from each phrase that is submitted. This chain will be used to normalize the
temporal expressions in the user’s request to the dialog system. By doing this, the system will be able to accept several expressions that refer to the same time frame. The chain is capable of identifying and normalize temporal expressions in Portuguese.

As we can see in the Figure 1.2 the $L^2F$ NLP Chain is made of six modules and three converters, following a pipes & filters architecture.

**Tokenize** This module is responsible for the identification of tokens in the phrase that enters the workflow (e.g.: Email and HTTP addresses, Roman Numbers, Symbols..);

**Morphosyntactic Tagging - Palavroso** This module (Medeiros, n.d.) is responsible for the morphosyntactic tagging of the tokens identified by the former module. It uses a dictionary to classify the tokens, according to ten categories. (eg: Category, Sub-category, tense...);

**Phrasal Division Module** This module performs the division of the text received from the last module in individual phrases. To do this, it takes into account the phrasal terminators (e.g.: , . / or ?);

**Morphosyntactic Disambiguation Module - RuDriCo** This is an enhanced version of PAsMo, a rule-based module (Paulo & Mamede, 2004). The main objective of this module is to adapt the results produced by the morphologic analyser into the specific needs of each syntactic analyser. To achieve this, the module changes the segmentation performed by the morphologic analyser, performing corrections and changes to the output of the former modules.

**Statistical Morphosyntactic Disambiguation Module** This module is called MARV (Ribeiro et al., 2003). It receives as input the output of the former module, and it uses the Viterbi algorithm to select one of the morphosyntactic tags that are associated with each word. The selection process is made by taking only into account the category and sub-category that the word belongs to.

**XIP Syntactic Analyzer** This is the last module of the $L^2F$ NLP chain (AAAt-Mokhtar et al., 2001). This module was developed by Xerox, performs dependency analyses and creates chunks based on grammatical rules. This has the objective of providing lexical information, apply morphosyntactic disambiguation rules, local grammars, chunk segmentation and chunk dependency calculation over the information provided by the last module. It is configured through a set of local grammars and lexicons. These grammars, developed at $L^2F$, provide the developer with a mechanism to define what he wants the system to identify in the text input.

### 1.3.3 DIGA Framework

The main objective of the DIGA was the development of research testbeds for communicative interaction, giving birth to a domain independent dialogue system platform that we are using in this thesis,
which is also used on the Ambrósio (Mourão et al., 2004) and LISA (Martins, 2008) system as mentioned before. Figure 1.3 shows the a simplified description of the DIGA Framework architecture. The framework can be divided in three main modules described below: Dialog Manager, Speech Recognizer and Speech Synthesizer.

![DIGA Basic Architecture Diagram](image)

**Figure 1.3: DIGA Basic Architecture Diagram**

### 1.3.3.1 Dialogue Manager

The main function of the dialogue manager is to control the dialogue flow: recognise all objects in the user’s input, initiate the designated services and generate the necessary answers to the user.

The Dialogue Manager (Madeira et al., 2003) (Martins, Mendes, et al., 2008) is a complex part of the dialogue system and groups the modules described below. This is a brief description of the modules.

**Language Parser** This module is responsible for the identification of the key objects in the user’s input. At the moment, this module is just able to identify affirmations.

**Interpretation Manager** The interpretation manager is able to build an interpretation with the speech context available and the domain information. These interpretations are created when the language parser and the object recognizer identifies objects in the user’s input. All the interpretations are sent to the Discourse Context for context analysis and domain matching.

**Task Manager** The task manager is the interface between the Dialogue Manager and the Service Manager. It works basically as an interface conversion system, storing details on each transaction;
**Behavioral Agent** Responsible for context cleaning according to the speech timings and user’s input. This module is not fully developed and presents little functionalities for the dialogue system;

**Surface Generator** Given a speech obligation from the Behavioral Agent and information for the Discourse Context, this module generates the text to be read to the user as the system output, based on pre-defined rules available in the domain configuration file;

**Discourse Context** This module manages the speech information related to the conversation between the user and the system. The module stores the past conversation between the user and the system, and the active speech acts.

### 1.3.3.1 Domains

**Domain** Groups all the information that is relevant to a group of devices. The frame is defined inside the domain configuration file. A domain is defined by:

- Name and description;
- Frame: matrix with information slots. Represents the set of concepts to which we will associate objects (words or expressions) related with the domain in question. This concept allows the identification of any word that was previously associated with the domain;
- Composed Rules: rules used by the dialogue system that can be translated to phrases in order to allow the system to communicate its intentions to the user. These phrases are generated using pre-defined models that allow phrases to be built in execution time with the contextual information available.

Figure 1.4 shows a domain configuration file. This is the example of an extremely simple domain for an agenda. With this example domain configuration, the system can recognise the phrase (Ex: Agenda reunião para amanhã / Schedule meeting tomorrow), because the frame only has three information slots. One for an action (ex: marcar / schedule), one for the target (ex: reunião / meeting) and one for when it is going to happen (ex: amanhã / tomorrow). It is possible to have an undefined number of slots, mandatory or just optional. The matching of the slots with the user’s input is done using a word based grammar, or relying on external recognition methods.

### 1.3.3.2 Devices

A device is a dialog system object, which packs dialog system definitions and implementation related to a particular domain. A device is created by the developer of the dialogue system when using the DIGA framework.

There are four ways for a device to interact with the dialogue manager:
Figure 1.4: Example of an XML domain configuration file.

- sendInformation: sends a message by parameter for the user;
- sendSlotFilling: sends the result of frame slot attribution;
- sendValidationResults: sends the result of the parameter validation;
- sendResults: sends the result of a service invocation.

The representation of a device on the dialogue system is defined by:

- Domain name: References to the domain where the device in question belongs to. A device always belongs to a domain;
- State: Defines a working mode. The state of a device provides the ability to limit the set functionalities available to the user, this can be done by defining a small set of functionalities in different states;
- Services: Define execution rules of the services provided by the device in question. The execution of a service is determined by the combinations of values in the information slots;
- Object Recognition Rules: These rules are used to map objects (words or expressions) to information slots used in the descriptions of the domain. This mapping is fundamental to the execution of the service.

For setting up a device it is necessary to specify the service configuration, which is done by the following parameters:

- name: name of the service;
param: service parameter;
slotref: reference to the slot in the domain to which this device belongs to;
execute: information regarding the device implementation;
name: name for the method to be executed in the java class of the device;
success: name of the device state that the execution state should be in, if the service execution was successful;
failure: name of the device state that the execution should be if the service execution was unsuccessful.

Figure 1.5 shows an example of the configuration of a service, with the specification of each parameter.

<initialstate>online</initialstate>
<state name="online">
<prompt>
  <item>Estado definido como online. O que quer fazer agora?</item>
</prompt>
<service name="Booking">
  <label>Service used for booking a meeting</label>
  <params>
    <param slotref="action"></param>
    <param slotref="target"></param>
    <param slotref="when"></param>
  </params>
  <execute>
    <name>scheduleMeeting</name>
    <success></success>
    <failure></failure>
  </execute>
</service>
</state>

Figure 1.5: Example of an XML service configuration file

Additionally, it is required to specify the object recognition rules. Figure 1.6 shows an example configuration file for the object recognition rules. Looking at the example, it is possible to verify that, for each slot, it is necessary to define a rule for recognizing the object. The objects that are associated with an object recognition rule are lexically associated with that information slot. The attribute id identifies uniquely the information slot referred by the rule.

1.3.3.2 Speech Recognizer:

AUDIMUS (Portugu et al., 1999) (Meinedo et al., 2003) (Meinedo, 2008) is the name of a generic platform for an Automatic Speech Recognition System specifically tailored to the European Portuguese language.
1.3.3.3 Speech Synthesizer

DIXI (Oliveira et al., 1991) (Oliveira et al., 2001) is a concatenative-based text-to-speech synthesizer, based on Festival speech synthesizer developed at Edinburgh University (Black & Taylor, 1997). This framework supports several voices and two different types of units: fixed length units (such as diphones), and variable length units. Currently we are using an European Portuguese voice, also developed at L2F.

1.4 Thesis Outline

Chapter 1 This first chapter aims at contextualizing this research project, presenting its general objectives and introducing the technologies used throughout its development;

Chapter 2 The second chapter of this master thesis presents the dialogue systems based on the DIGA framework and the main technological approaches in the area of dialogue systems when dealing with calendar systems;

Chapter 3 This chapter explains the architecture of the system developed, its integration with third party applications and frameworks. It explains in a detailed way the functionalities provided by the voice interface, the problems found during its development and the solutions to overcome those problems;

Chapter 4 This chapter focuses on the evaluation of the dialogue system developed, presenting the evaluation methods used, its characteristics and results obtained.

Chapter 5 This chapter presents the conclusions of this research work, alongside with a review of the DIGA framework and a prospect of future work.
This section describes the evolution of dialogue systems in L²F labs, from the first dialogue systems developed, to the ones that are now in development, making a brief analysis of what has been done through time. The second part of this section analyses and compares dialogue systems that interact with calendars or scheduling applications, concluding with a comparison between the systems analysed.

2.1 DIGA based dialogue systems

In this section we are going to review most of the dialogue systems developed at L²F, taking special attention to its evolution and new characteristics. Additionally, we will do a brief description of each system developed, following a chronological order.

2.1.1 Voice Control of a Hi-Fi

The Voice Control of a Hi-Fi system is one of the first dialogue systems developed at L²F (Jordão & Novais, 2001). It was developed as a proof of concept for the use of voice in controlling the functionalities of a household device. In this case, the control of a Hi-Fi using only the voice, the Hi-Fi was chosen because it is a rather complex device which possesses more options than the usual on/off. To complete this proof of concept, a speech recogniser which later turned into the Audimus based recogniser currently used in L²F. Besides the speech recogniser, the words spoken by the user were analysed by a simple interpreter module that would look, word by word, for key commands. When a key command was found in the sentence, the command associated to that word would be issued to the HI-FI system.

2.1.2 Meteo: A telephone-based Portuguese Conversation System in Weather Domain

Meteo is a dialogue system that was developed with the objective to provide a conversation system in a weather domain, applied to the European Portuguese language.

With this system (Cardoso et al., 2002) its possible to provide a spoken interface over the phone, to access meteorological information for the main cities in Portugal for a period of three days from the
current day. Information like maximum, minimum and current temperatures, air humidity and sky conditions. All meteorologic data is gathered to an internal database, which is updated regularly from online meteorology services.

As it will be shown later on, this dialogue system has many similarities with the Electronic Assistant as it uses the same speech synthesis and recognition engines and follows a similar architecture but with some interesting differences. Instead of a single hub architecture, in this system a dual HUB architecture was adopted, separating the Audio subsystem from the dialogue subsystem.

The audio subsystem is responsible for:

**Audio Server**  Consists of a phone interface that streams audio between the phone line and the Audio HUB;

**ASR**  Audio speech recogniser based in the Audimus system;

**Text-To-Speech**  Speech synthesiser based on the Dixi+ project (Oliveira et al., 2001);

**Log module**  Is a module that logs all the audio user input, in order to allow the developers to improve the speech recognition;

The dialogue subsystem is responsible for the following:

**Language Parser**  Module responsible for the keyword spotting, this module finds the city name to which a user refers to, as well as, which meteorological conditions and time of the day in the user input;

**Dialogue Manager**  Identical to the module available in the Electronic Assistant (Maia & Cassaca, 2002), works on a frame based system where the fields available are filled up with the keywords spotted by the language parser and context information from former queries;

**Natural Language Generator**  This module, together with the information from the dialogue manager, generates textual answers to be presented to the user, it fills up empty spaces pre-defined answer templates with the results from the user’s questions to the system.

This system had the purpose to show that, in the time being, there were already technologies mature enough to be used in a new type of applications. However, this system still lacked the naturalness that was only achievable with the readjustment based on the real use of the system.
2.1.3 The Electronic Assistant

The Electronic Assistant is one of the first big dialogue systems developed at L²F (Maia & Cassaca, 2002). It aimed at creating an electronic assistant that provides the user with the possibility to control his email, the lighting of the room, hi-fi or the air conditioner. This project was the starting point of the platform that today is still used to developed dialogue systems.

The main objective of this work was to develop a modular and extendable platform for dialogue systems. This would allow any developer, given the API created, to develop new services to the user.

This system takes a module architecture based on the Galaxy Communicator, divided in seven blocks:

Language Compression Module responsible for tagging the user’s input, after performing morphosyntactic processing of its content;

ASR This module allows the Electronic Assistant to recognise speech, it is able to convert the user’s speech in textual phrases. The ASR is based on the Audimus speech recognition system;

TTS DIXI based module that converts the system output in speech to the user;

GUI Basic graphical user interface that allows the user to use text as a form of input;

Language Generation This module will generate textual responses from the system to the user, based on information provided by the devices;

Service Manager This is the entry point between the external devices and the dialogue system, it is capable of selecting the services which can execute a specified task;

Dialogue Manager This module is responsible for keeping contextual information regarding the user’s input, and to match the information tagged with the actions the devices can perform through the use of a frame based system.

This architecture is supported by the existence of a group of servers/blocks that work together to reach a common goal. The communication and coordination between the blocks is supported by a hub. This hub is responsible for the underlying modular characteristics of this architecture, because it encapsulates all the information regarding the messages that are exchanged between each user and it serves as the central point of the system. The messages exchanged between each block have a set of pre-defined fields, which reduced the flexibility of this approach although they provided some extension mechanisms by defining extra fields. To avoid this, a XML format was adopted and used to encapsulate information inside the hub messages, providing additional flexibility to the information exchanged, but resulting at the same time in a decrease of performance.
This project was developed in a graduation thesis, reaching a prototype level and producing the foundations of the dialogue systems developed in L²F at this moment.

### 2.1.4 Ambrósio - The Virtual Butler

This prototype goes by the name of Ambrósio, a butler that is able to control home devices, such as lights, TV’s and acclimatisation systems through X10 electrical protocol or IrDA infrared protocol. This is a dialogue system that performs the user’s orders over a determined set of devices.

The idea behind this project was to apply the Dialogue Manager, responsible for domain independence, with the Service Manager, responsible for the functionality integration, in order to obtain a single generic dialogue system with the best usability possible. Both Dialogue Manager and Service Manager were developed in previous projects (Madeira et al., 2003) (Mourão et al., 2004), but independently. The objective here was to integrate the advantages of both components in order to obtain a generic dialogue system. This gave birth to what now is called the DIGA framework.

The architecture of the Ambrosio system is identical to the one in Electronic Assistant, differing on the strict separation between the Service Manager and the Dialogue Manager in the Ambrosio.

This project had two big iterations. The first, described above, where a generic dialogue system was created using the integration of previous knowledge developed in at L²F. And a second one, which is now the butler for the “House of the Future” at the Portuguese Telecommunications Museum (Mourão et al., 2004). Here, the system was heavily focused on the domotic features that were available in the house. However, the speech functionalities were user dependent due to major constrains on the speech recogniser, which in this case was not a major problem because there were only a few persons showing the system to the visitors of the museum. This dialogue system was also connected to a 3D face (Viveiros, 2004), also developed at L²F. This 3D face provided the user with a point of interaction and feedback with the system, with voice output synchronisation of the facial movements resulting in easier and more natural interaction with the system.

The Ambrosio prototype underwent further optimisations since it was deployed at the museum, most of them being a direct result from the continuous development of the DIGA framework which, among other things, resulted in an increase of execution speed and speech recognition efficiency.

### 2.1.5 Cooking Assistant

This dialogue, built over a task oriented dialogue system framework, system was a proof of concept (Martins, Pardal, et al., 2008) designed to assist during the cooking process. It was developed to prove that it is possible to use the DIGA, a task-oriented framework, to develop a question based system. The
prototype developed is able to help the user follow a cooking recipe, or change his car tire. In the second generation prototype, it is possible to have the system read domain information from XML files which gave the possibility to provide the system with one or more recipes by just adding new recipe files. Additionally, the system provides the option of playing multimedia files to the user in order to explain him some procedure. As a proof of concept prototype, this system presents several problems that are still unresolved, like the lack of a method to select the recipe that the user wants to do if we want to deal with a reasonable amount of recipes.

2.1.6 Lisa, the Personal Assistant

Lisa can be seen as the third generation of the dialogue systems developed in L²F. It is a significant evolution since the platform used in the development of the Ambrosio system, available at the “House of the Future”. The major developments were in the field of the audio input, were it is now possible to interface directly with PBX systems or web interfaces. As a virtual assistant, the system needs to be connected to a phone system in order to be able to receive and make calls. This characteristics are completely new to the DIGA framework and resulted in a considerable development in terms of functionalities and performance to the framework when compared to the one still in use by the Ambrosio system in the “House of the Future” at the Portuguese Telecommunications Museum.

This system gives to the user the following functionalities:

**Management of the availability** The user can manage its state of availability towards other users, whether he is busy, available or not in his office;

**Management of call redirection** The user is able to set where he wants to redirect all incoming calls;

**Contact Manager** The system is able to store and translate names in phone numbers or email addressees;

**Voice Dialling** The system establishes a call to a number or to a contact in the agenda;

**Voice mail** Permits the user to access common voice mail functionality, with the possibility to interact with the agenda;

**Agenda Management** The user is able to schedule and search events in a personal agenda.

**Messaging Services** The user is able to write or to dictate messages and have them sent through more than one channel (email, sms, mms, fax.).
2.1.6.1 Architecture

The Lisa dialogue system was the first dialogue system to introduce new architectural and design changes, this changes resulted in the connection of Dialogue System framework to a Business Module and a PBX system, both from third party developers. To allow this, the dialogue system framework was adapted with SOAP interfaces in order to support the connection to third party modules. The architecture developed still does not allow the system support more than one user at a time. This can be done by issuing more than one instance of the dialogue manager to handle the speech processing, not being necessary to have more than one instance of speech recogniser and speech synthesiser running at the same time.

2.1.7 Phone banking

The phone banking dialogue system is a system that provides bank and trading services to the user. The innovation in this system is related to the fact that the system avoids the use of DTMF\(^1\) to guide the dialogue system, letting the user navigate all the options just using his voice.

This system presents to the user a flat menu, which allows him to perform the following usual banking operations:

- Check the balance;
- Check account number;
- Check transactions;
- Check overall position;
- Check credit card information;
- Request new ATM cards;
- Cancel lost credit cards.

Regarding the stock exchange operations, the following operations are available:

- Check Stock prices;
- Check Exchange rates;
- Check Interest rates;

\(^1\)Dual-tone multi-frequency
• Acquire new stocks;
• Sell stocks.

This system follows the same architecture as the dialogue system developed for Lisa.

2.2 Calendar Systems with Dialogue Interfaces

In this section we are going to analyse four systems that make use of a voice interface to interact with calendar applications and present examples of possible user interaction with some of this systems while taking special attention to temporal expressions and how they are processed in each system.

2.2.1 SCHED - Simplified Calendar Handling through English Dialogue

SCHED (Simplified Calendar Handling through English Dialogue) is one of the first dialogue systems dedicated to meeting scheduling in an online calendar application. This analysis focuses on the feasibility of developing a dialogue system that accomplishes typical office tasks by means of voice communications with the user. The development of this project aimed at creating a framework for the scheduling of office activities.

This project started at IBM Research, following a task based approach, where each task is simplified as much as possible until it is achievable to have some of the pretended functionality working. To reach its objectives, the project relied on defined system characteristics:

• Dialogue context, supported by a knowledge network improved the speech generation and understanding;
• Assumption of common natural language;
• Communication between the user and the system takes the form of a dialogue;
• Purpose for the communication, which means, there is something the user tries to achieve with the communication.

The dialogue fragment in the Figure 2.1, is an example where the phrases in upper case refer to the dialogue system sentences and the phrases in lowercase refer to the user’s input. This example provides a very segmented interaction between the user and the system, which means that the user needs more than just a few interactions to schedule an event. However, according to the system specification, it should be possible to provide more information to the system in each interaction with more complex
sentences. The connection with the calendar system is made through the issue of uninterpreted character strings commands to another system that stores them on a database. This system was developed from scratch to cope with the calendar domain, which allowed the developers to build a knowledge network fully adapted to the domain in question. This knowledge network holds information about the things that can be referred to in a dialogue between the system and the user, each node representing a particular person, office or a collection of entities. The information held in the knowledge network is restricted to dates, times, places, people or activities. All other information regarding possible understanding of concepts, like meetings, is disregarded.

One of the most relevant aspects of this research is the importance given to the need of thorough analysis of the way date and time information is stated. The interpretation of phrases like, "same time" and "the next day" relies on information, provided by the user, earlier in the dialogue. Due to the broad way of defining and expressing date and time information in English, the system presents several limitations regarding the processing of temporal expressions since it just supports a fixed number of temporal expressions defined in a rule based system.

2.2.2 RavenCalendar: A Multimodal Dialog System for Managing a Personal Calendar

RavenCalendar(Stenchikova et al., 2007) is a multimodal dialogue system built around Google Calendar\(^2\) and Google Maps\(^3\). As the dialogue system developed in this master thesis, RavenCalendar is a system that also relies on generic framework for dialogue systems, RavenClaw.

RavenClaw(Bohus & Rudnicky, 2003) is a domain independent dialog management framework, developed at CMU, for task-oriented spoken dialog systems. The RavenClaw framework provides all the speech processing functionalities for RavenCalendar.

The RavenClaw dialog management framework is a generic framework for the development of dialogue systems. It provides a separation between the domain-specific and the domain-independent aspects of the dialog control logic. To develop a new dialogue system using this framework, system developers just need to focus on defining the domain-specific aspects. The RavenClaw Dialog Engine provides generic conversational behaviours (error-handling, timing and turn-taking, and universal dialog mechanisms). Full spoken language interface capabilities (speech recognition, language understanding, language generation, speech synthesis) are usually provided by the Olympus (Bohus et al., 2007) dialog system architecture, which provides all these components, as well as the means for connecting them. RavenClaw and Olympus are connected through a Galaxy Hub, in a TRIPS (Allen et al., 2000) alike

\(^2\)http://calendar.google.com

\(^3\)http://maps.google.com
1. I want to schedule two meetings and a seminar.
2. WHEN WILL THE FIRST MEETING BE?
3. Thursday morning.
4. WHAT TIME IN THE MORNING WILL THE FIRST MEETING START?
5. 10.
6. FOR HOW LONG?
7. I don't know
8. WITH WHOM?
9. Lance.
10. WHERE?
11. In his office.
12. WHAT ABOUT?
13. "Progress report".
14. WHEN WILL THE SECOND MEETING BE?
15. Same time.
16. WHICH DAY?
17. The next day.
18. FOR HOW LONG?
19. About one hour.
20. WITH WHOM?
22. WHAT ABOUT?
23. "Experiments".
24. WHEN WILL THE SEMINAR BE?
25. At 2:30 tomorrow, for 90 minutes.
26. WHERE?
27. The first conference room on the ground floor.
28. WHO WILL BE THE SPEAKER?
29. Tom Bever, but I do not know the topic.
30. YOU HAVE A MEETING WITH LANCE MILLER AT 10:00 AM ON THURSDAY IN HIS OFFICE ABOUT PROGRESS REPORT.
YOU ALSO HAVE A ONE-HOUR MEETING WITH JOHN THOMAS AND JACK CARROLL FROM 10:00 AM UNTIL 11:00 AM ON FRIDAY IN THE PSYCH LAB ABOUT EXPERIMENTS.
YOU ALSO HAVE A 90-MINUTE SEMINAR ON AN UNSPECIFIED TOPIC BY TOM BEVER FROM 2:30 PM UNTIL 4:00 PM ON TUESDAY IN THE FIRST CONFERENCE DINING ROOM.
31. SHALL I ISSUE THE FOLLOWING COMMANDS TO THE TIME MANAGEMENT SYSTEM?
Z.S. +THU 10:00 MTG - MILLER, HIS OFFICE, PROGRESS REPORT
Z.S. +FRI 10:00 MTG - THOMAS AND CARROLL, PSYCH LAB, EXPERIMENTS
Z.S. +TUE 14:30 90 SEM - BEVER, FIRST CONFERENCE DINING ROOM
32. Yes.
33. DONE.

Figure 2.1: Fragment of a dialogue interaction from the SCHED system.
architecture, which is quite common in dialogue systems architecture. The TRIPS architecture provides
the ability to use other system besides Olympus, in order to obtain full spoken language capabilities
(speech recognition and synthesis).

By reviewing the dialogue presented in Figure 2.2, we could infer that the system supports simple
tasks (adding and listing events) and it is capable of dealing with some temporal expressions (e.g. add
a meeting for Monday). However no information is provided in that way.

| S: Welcome to RavenCalendar, the automated dialog calendar system. |
| S: What can I do for you today? |
| U: add a meeting for Monday |
| S: For what time do you want to add the meeting? |
| U: from ten until eleven |
| S: The event was added type meeting |
| for tomorrow starting at ten a.m. |
| U: List all events for the next week |

Figure 2.2: sample dialogue from a RavenCalendar interaction

All the interaction between RavenCalendar and Google Calendar are done through a back-end
server. This means that RavenCalendar requires a third-party server to provide the connectivity between
Google Calendar service and the calendar functionalities RavenCalendar provides. The multimodal
aspect is developed further with the connection to the Google Maps application, which allows the user
to pin point in a map the location for a certain event that is being scheduled in Google Calendar.

Unfortunately it was not possible to find precise information about the system capabilities regarding
the processing of temporal expressions. Looking at the example dialogue in Figure 2.2 it is possible
to assume temporal expression recognition is supported, although it is not possible to know in which
terms this processing is done and to what extent it is supported by the system analyzed.

2.2.3 You’ve Got Mail! Calendar, Weather and More: Customizable Phone Access to Personal Information

You’ve Got Mail (Perez-Quinones et al., 2004) is a system that provides access to calendar, email,
weather, and news over a phone using a VoiceXML interface. The main objectives of this system are:

1. to provide ubiquitous access to information while away from office/home without any kind of
   synchronization. The PDA offers a ubiquitous access to information when the user away from the
desktop, but a previous synchronization with a desktop PC is required in order to achieve that.
   You’ve Got Mail attempts to provide a centralised calendar system with more than one interface
   using an everyday appliance, the phone;
2. to provide the possibility for the user to pre-program and customise the system in order to minimise the interactivity required to address core functionalities. The dialogue systems currently available to the general public, like home banking applications, do not provided any kind of customisation in order to speed up the interactions. This means that if the user performs always the same actions in the system, it will have to go through the same menus and sub-menus every time he wants to perform that action. The possibility of having a customised menu system associated with each user’s phone number would result in a great increase in efficiency and consequently time reduction, because when the user would dial-in from his usual cellphone he will be presented with his self-custumized option menu.

The You’ve Got Mail dialogue system does not fully support the functionality of a personal calendar system with all the possible interactions of the calendar through the voice interface. Instead, it focuses only on supporting interactions that are crucial, leaving complex and nonessential tasks for other modes of interaction. Betting in fewer functionalities is an important aspect that differentiates this system from the other systems analysed, as well as the system we want to produce in this master thesis.

For nine months, the prototype of the calendar system was used by in house developers. Despite being an insufficient test, it provided useful information regarding to where and when the access to the calendar though this dialogue system has been essential:

- Refresh memory of following day’s activities, upon return from a trip;
- Handling unexpected events that requirers cancelling of remaining events of that day (e.g.: I’ve got a flat tyre in the morning, therefore I have to cancel all my meetings for that morning).

This system started as an interface for a calendar system, supporting some functionality in the initial prototype through a telephone number. Other domains, as the email and weather domain were added later on as the result of the success of the initial prototype. The system relies on iCal calendar application for the calendar functionalities. The ubiquitously is provided by exporting all the calendar information from this application through a WebDav (E. James Whitehead, 1997) server that maintains the scheduling information in a single point. This means that there is a synchronization process, but it is completely hidden from the user.

The system provides the possibility of customising the voice interface through a webpage. This allows the user to previously tailor the service to his needs and reduces the time it takes in achieving his goal.

Contrary to the other examples presented in this master thesis, where voice interfaces were created over proprietary dialogue frameworks, this application relies on a voiceXML\textsuperscript{4} interface, which allows

\textsuperscript{4}http://www.voicexml.org/
the use of a generic speech synthesiser and recogniser when they support voiceXML. For example, in Figure 2.3 we can see an example of a voiceXML file that results in the “Hello World” phrase being synthesised by the speech synthesiser.

```xml
<?xml version="1.0"?>
<vxml version="2.0" xmlns="http://www.w3c.org/2000/vxml">
  <form>
    <block>
      <prompt>
        Hello world!
      </prompt>
    </block>
  </form>
</vxml>
```

Figure 2.3: Reduced example grammar for the birthday service

VoiceXML is a markup language created by a consortium of companies in the communications and computer area. It is a XML-based\(^5\) language that includes tags specifically for building voice interfaces and support telephony applications, which combines the advantages of a XML-based language with the unique features of voiceXML, resulting in a standard for phone-based applications.

The adoption of VoiceXML when developing a dialogue system results in many advantages for the application developer. It allows the application to be connected to any speech recogniser and speech synthesiser, as long as both speech recogniser and synthesiser support voiceXML. With this, any dialogue system can easily use other speech synthesizer/recogniser systems without any extra work regarding their API. On the other hand, adopting a VoiceXML results in extra work for the developer, because the dialogue system has to fully support VoiceXML API. When using a non-VoiceXML architecture, the dialogue system only uses and supports the features that it needs from both speech synthesizer/recognizer. In the short term this results in less work for the developer, but in the long run, it creates a strong dependance between the dialogue application and the speech synthesizer/recognizer. This happens because the dialogue system is not compatible with any standard so it has a strong connection with the speech synthesiser and recognition engines that it is using.

2.2.4 SpeechActs PROJECT

Accessing remotely to the information on a local computer has proven to be a research subject. The SpeechActs is a research project that aimed at providing a flexible and natural approach to access a computer remotely. This research studies one way to access a stand alone calendar application through a voice interface.

\(^5\)http://www.w3.org/XML/
The SpeechActs project (Yankelovich, 1994), shows a different approach to the development of voice applications that interact with a calendar, as in this case, we are dealing with a standalone calendar application on the user’s computer, and not an online calendar. The SpeechActs project intends to provide a framework to ease the development of voice interfaces for accessing applications remotely. The developed prototype supports basic calendar functionality.

The first prototype created was based on the SpeechAct framework interfaced with Sun’s Calendar Manager application. Before developing a full-blown interface, with all the characteristics, a survey was performed in order to detect what were the major problems with the calendar application in question. Issues regarding remote access were detected by recording calls made to the user personal assistant, which in the time being, provided a “remote interface” for the calendar application to each user. By doing this, it was possible to determine which were the most used functionalities while away from the application, and how the users used these functionalities.

Looking at the sample dialog in Figure 2.4, it is interesting to observe that this dialog system takes a pre-emptive initiative in providing the next events to the user after its identification.

```
S: Welcome to RavenCalendar, the automated dialog calendar system.
S: What can I do for you today?
U: add a meeting for Monday
S: For what time do you want to add the meeting?
U: from ten until eleven
S: The event was added type meeting
    for tomorrow starting at ten a.m.
U: List all events for the next week
```

Figure 2.4: sample dialog from the SpeechActs system

The speech interface was designed with the intent to mimic the same kind of interaction that the users had when they called their personal assistant. This was done with the objective of providing the most natural interaction as possible to the user of the application. It is important to notice that only a small subset of functionalities are supported by the voice interface, like browsing the user’s calendar and other calendars the user had access to.

The development of this prototype helped the research team to define the major problems within the developed speech user interface. At the end of the development, a number of questions were still unanswered:

- How to make the user aware of the boundaries of the functionality provided by the current application;
- How to design an effective speech interface taking into consideration the limitations of the speech recognition technology;
• How to characterise the error recovery behaviour necessary to compensate for the flaws in the speech recognition technology.

2.2.5 System and Features Comparison

In this section, we analyzed several systems that provide calendar functionalities through a voice interfaces. Now we compare the main features of each system analyzed and their architectural differences and similarities.

Table 2.1 shows comparison of the date when each prototype was developed, the main functionalities they provided, the calendar system they used and the modes of interaction supported. In our opinion, it is interesting to see the evolution in terms of functionalities and modes of interaction provided throughout the years. It is interesting to see that the older systems were standalone systems, while the more modern one, including the one in development on this master thesis, were all developed over some kind of framework for dialogue systems.

<table>
<thead>
<tr>
<th>Systems</th>
<th>Year</th>
<th>Functionalities</th>
<th>Type of Calendar</th>
<th>Multimodal Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHED</td>
<td>1978</td>
<td>calendar</td>
<td>mock-up calendar</td>
<td>voice/text interface</td>
</tr>
<tr>
<td>SpeechActs</td>
<td>1994</td>
<td>calendar</td>
<td>sun’s calendar manager</td>
<td>voice interface</td>
</tr>
<tr>
<td>You’ve Got Mail!</td>
<td>2004</td>
<td>calendar, e-Mail,news</td>
<td>iCal based</td>
<td>voice interface</td>
</tr>
<tr>
<td>RavenCalendar</td>
<td>2007</td>
<td>calendar</td>
<td>GoogleCalendar</td>
<td>voice/text and map</td>
</tr>
</tbody>
</table>

Table 2.2 shows a comparison between the features provided by each system. Unfortunately, each system presented particular characteristics that did not allow to perform a side by side comparison. However, it is important to point to the fact that only one of the systems analyzed supported VoiceXml and dialogue customization.

<table>
<thead>
<tr>
<th>Systems</th>
<th>VoiceXML</th>
<th>Norm.Temp.Express</th>
<th>Custom Dialogue</th>
<th>Other Data Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHED</td>
<td>not supported</td>
<td>crudely supported</td>
<td>not supported</td>
<td>no</td>
</tr>
<tr>
<td>SpeechActs</td>
<td>not supported</td>
<td>information n/a</td>
<td>not supported</td>
<td>calendar app.</td>
</tr>
<tr>
<td>You’ve Got Mail!</td>
<td>supported</td>
<td>information n/a</td>
<td>web-interface</td>
<td>Ical app.</td>
</tr>
<tr>
<td>RavenCalendar</td>
<td>not supported</td>
<td>information n/a</td>
<td>not supported</td>
<td>GoogleCalendar</td>
</tr>
</tbody>
</table>

Taking into account that the information available on each of the systems analyzed is limited, it is rather difficult to produce a comparison between systems analyze since it is not possible to access the information about the same characteristics on each system. From the systems analyzed, its important to notice that the SCHED system was the only one that was designed to support calendar functionalities.
since the beginning, with a custom made knowledge network built-in within its dialog manager.

The remaining systems analyzed, were either built on top of full featured dialogue system framework (RavenCalendar), or relied on third party natural language components. The option of using a generic framework when building a dialogue system provides considerable reduction in development time when prototyping a dialog system application. RavenCalendar is the system that presents most similarities with the system in development on this master thesis. However it was not possible to find more information about this system, its characteristics or the recognition of temporal expressions.
3.1 System’s Architecture

This section explains how the Voice dialog system was developed and integrated with the DIGA framework. Additionally we are going to give brief examples of the configuration files on both domain and service configuration files, explaining some of the options taken. The voice interface system is divided in three modules: a temporal expressions normalization module, a temporal expressions processing module and an operations module. The first uses the L2F NLP Chain to process the user’s input in order to find any temporal expressions and replace them in the user’s request; the second module processes the normalised temporal information from the first module into programming language objects, it checks for missing date values and calculates default dates when necessary; the third module is responsible for connecting directly to google calendar and performing the requested operations using the temporal expressions identified by the first module and processed in the second module. These modules are integrated within the dialogue system framework used in this project.

3.1.1 The integration with the Dialogue System Framework - DIGA

During the development of this master thesis, we always tried to use the Dialogue System Framework in the least evasive way. This means that we tried to adapt the objectives and respective solutions around the system specifications and limits, and not the other way around. It would have been less troublesome to implement additional features in the dialogue system framework that would facilitate the development of new functionalities. Not to mention the fact that the dialogue system framework was going through a period of reengineering and a new development cycle (Martins, Mendes, et al., 2008). By not changing the framework, we aimed at testing the adaptivity of the DIGA framework to a new domain and additional requirements, and to be able to point what can be developed further in order to allow better integration of new features without the need to change parts of the framework.

3.1.1.1 Developed Components

The development of this dialog system and consequent integration within the DIGA framework requires the following steps to be taken:
1. develop a device with the domain configuration files, service specification and correspondent implementation;

2. develop a service specification and correspondent implementations;

3. implement the necessary parsers and associate them with the correspondent slot types.

3.1.1.1 Google Calendar Device  A device represents a group of service configurations and correspondent implementation related to the same domain in the dialog manager. In the beginning of the development of this master thesis, we started by creating a device for each group of functionalities, scheduling, searching, listing and deleting. Figure 3.1 shows the first architecture of the Operations Module, which resulted in a considerable replication of code regarding conversions and translation of temporal information. The one device per function design resulted in the necessity to provide the dialog system with the action and target before any other input. This was necessary in order to the dialog system identify which device was responsible for the action requested (Figure 3.2). Facing the limitation of the first solution, we concluded that the user should be able to give more information to the system in a single utterance, not only to support advanced users but also to use the dialog system capabilities. Considering this, a second solution was developed and the system was redesigned into a single device system architecture, grouping all functionalities together and providing additional flexibility to the user.

Figure 3.1: Architecture of the Operations Module on the first system solution, it supports a device for each functionality

Figure 3.3 shows a simplified XML configuration file of the domain specification used in the calendar application. This file specifies all the information slots used by all the services built for the calendar domain and its correspondent types. The information type used will define which parser will be run when the system is waiting/asking for that slot.
Figure 3.2: Example of a dialog with the first solution developed. This dialog system would not accept more than the user request in the first interaction disregarding any temporal information

```xml
<slot type="unknown">action</slot>
<slot type="unknown">target</slot>
<slot type="DATE">date</slot>
<slot type="TIME">hour</slot>
<slot type="TEXT">title</slot>
<slot type="unknown">listPeriod</slot>
<slot type="unknown">recurrency</slot>
<slot type="unknown">confirmation</slot>
<slot type="TEXT">endDate</slot>
<slot type="unknown">searchType</slot>
<slot type="NUMBER">num2delete</slot>
```

Figure 3.3: Simplified domain configuration of the calendar application

Table 3.1 shows which parsers are used by the dialog manager to identify the objects that can be associated to each slot. The Unknown slot type is the generic slot type, used for words that are present in the device grammar. Figure 3.4 shows a small example of the grammar used for both action and target information slots. The object identification is performed by the generic parser, this is achieved by comparing if the word present in the users input is also in the grammar associated with that information slot. By definition, the generic parser is executed for each sentence. However, in order to be able to detect temporal expressions in any of user’s input the date parser is also executed, if this was not the case, the dialog system would only detect a temporal expression when it was asking for one. The dialog system only executes non standard parsers when it is trying to fill a specific slot which has that parser associated to that slot type. Both Date and Time slot types use the date parser which classifies the objects with the tag date, or the tag time depending if it is a calendar date or an hour. The text slot type is a special slot type which changes the dialog system’s default behaviour, halting the executing of the generic parser, meaning that this parser is only executed when the dialog system is asking for the text slot type. Although by default the text parser are used for special input (e.g, names, titles of events, etc..) that does not require parsing, in this dialog system they are also use it for duration. It may seem strange at first sight to use this information slot type for duration, but this was a necessary workaround in order to avoid mismatches between the identification of duration and other temporal expressions, keeping the recognition of duration apart from the recognition of dates. This means that duration are
only recognized if the dialog system is asking for one.

Table 3.1: Parsers used in each slot type

<table>
<thead>
<tr>
<th></th>
<th>Generic Parser</th>
<th>Text input parser</th>
<th>Date/Time Parser</th>
<th>Duration Parser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.4: Reduced example grammar for the birthday service

3.1.1.1.2 Parsers Two parsers were developed, one that deals with duration and another one that deals with expressions referring calendar dates or hours. The L²F NLP Chain gives an output similar to the one in Figure 3.5 when processing the user’s input with a temporal expression. Even though this example does not show the full output, it is possible to observe what kind of information can be obtained from the chain analysis. From the dependency analysis in this example, it is possible to infer that the action associated to the verb (marcar/schedule) happens on the 25th of May, because the expression “25 de Maio/ 25th of May” is set as a dependency of the verb.

Table 3.2: Services defined for each functionality

<table>
<thead>
<tr>
<th>Scheduling</th>
<th>Listing</th>
<th>Searching</th>
<th>Deleting</th>
<th>Auxiliary Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal Event</td>
<td>List by Period</td>
<td>Search by Title</td>
<td>Delete by Date</td>
<td>Confirm Duplicate Events</td>
</tr>
<tr>
<td>Recurrent Event</td>
<td>List Next Event</td>
<td>Search by Date</td>
<td>Delete</td>
<td>New Event Date</td>
</tr>
<tr>
<td>Birthday</td>
<td></td>
<td>List by Date</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.1.1.1.3 Services definition Services represent functionalities that the dialog system will provide to the user. It is in the service definition that it is specified which methods from the Operations Module are going to be executed when the dialogue manager has all the necessary information. Figure 3.6 shows an example of the service definition for the birthday service, the service responsible for the scheduling of birthdays in calendar. When the dialog manager has the slots referring the date, the action and the title...
Figure 3.5: Cut down version of the L²F NLP Chain output for the sentence “marcar evento para o dia 25 de Maio” / “Schedule an event to the 25th of May”

filled with the information provided by the user, the method createBirthday in the Operations Module will be executed. Table 3.2 shows the services implemented for each kind of functionality of the dialog system, plus the auxiliary services implemented. The Confirm Duplicate Events and New Event Date are services used to confirm with the user if he wants to continue booking an event on top of an existing event, or to provide a new date to the event.

Figure 3.6: Example of the XML configuration configuration for the birthday service
3.1.1.2 Module integration

In Figure 3.7 it is possible to observe where each module of the voice interface system resides within the Dialogue System Framework. The user’s input is passed to the dialog manager by the speech recognition module (e.g., “marcar evento para o dia 20 de Janeiro” / “schedule an event for the 20th of January”), the dialog manager runs the necessary parsers according to what was asked to the user (e.g., generic and date parser if it was the first interaction). Assuming the generic parser identified the action (e.g., “marcar” / “schedule”), the target (e.g., “evento” / “evento”) and the date (e.g., “20/01/- - -”) it will acknowledge that the user wants to schedule an event and will ask the user with the missing information slots in order to execute that event, time of the event, duration and title.

Using the utterance generation rules defined in the domain configuration file, the dialog manager generates a question in order to obtain information for one of the missing information slots. This question will also display what the dialog system acknowledge already from the previous user input, working as an implicit confirmation (e.g., “Para marcar um evento para 20/01/2010 é necessário que indique uma hora para o evento” / “To schedule an evento to 20/01/2010 it is necessary to define an hour of the event”). In order to convert the date input into the one used for question generation it is first necessary to validate it using the validation methods in the Temporal Expression Processing Module and then convert it to a non textual dependent format and define the year value. After asking the user for time values, durations and the event title, verifying if the values are possible in the case of the duration or the event time, the dialog system has all the information to schedule the event in the calendar. Facing this, the dialog manager executes the event scheduling method associated to the event scheduling service in the Operations Module, the method is executed with the information on the information slots as arguments.

3.1.2 System Modules

3.1.2.1 Temporal Expression Normalization Module

Figure 3.8 shows a conceptual module capable of identifying and normalizing several temporal expressions, with or without precise temporal information (hours and minutes), and durations. It uses the L2F NLP Chain to analyse the text that enters the dialogue system through the speech recogniser or the text input, and identify temporal expressions present in the text. This process is made through the dialogue system parsers, specifically the date and duration parser, a post-processing layer and a connection layer. This module is responsible for the identification and post-processing of all the user’s input regarding temporal expressions.
Figure 3.7: Overall Dialog System Architecture.
3.1.2.1.1 L²F NLP Chain web service and connection layer The L²F NLP chain consists of a script which runs individual tools in a chain sequence that is capable of identify and normalise temporal expressions. The requirement of a standalone system left us with the need to integrate the chain with the dialog system. The following solutions are considered:

- include the chain inside the dialog system should be one of the most efficient solution. however it is not a viable solution due to performance and copyright issues;
- install the chain in the same system of the dialog system. This is the easiest solution, however it creates a hard dependency on the machine and diminishes the portability of the dialog system;
- implement a web-service that functions as an access point to the chain, maintaining exactly the same interface. Although it is the best solution found, it is not a perfect one. It is considerably slower to use the chain through the web-service, resulting in the system taking up to 5 seconds to answer the user.

To minimise this problem, we decided to create a small php/mysql cache inside the web-service in order to speed up the processing of repetitive input, by caching all output of the L²F NLP Chain and using the cache every time a repetitive input reaches the web-service. This is a simple solution which can to decrease the response time in 2 to 4 seconds in repetitive situations.

3.1.2.1.2 Post-Processing Layer The post processing layer is responsible for converting the output of the L²F NLP Chain to object that the dialog manager can handle.

**Durations** When dealing with durations, the post-processing layer transforms the `A0M0S0D0H0M0S0` expression into a `HH:MM` and passes it to the dialog system. This is a sim-
plification used to reduced the need of additional calculations regarding temporal expressions in other parts of the service implementation. The conversion is done by converting all durations into seconds and then back to hours and minutes, the seconds are disregarded because the notion of seconds is rarely used in the calendar domain. After performing this conversion, the resulting temporal expression is added to the user input inside the \textit{duration} tag. Figure 3.9 shows an example of the duration normalization process.

\begin{verbatim}
input: O evento tem uma duração de duas horas.
output: <NORMALIZATION sent ="0" num ="36" start="27" end="39" value="A0M0D0H2M0S0"/>
post-processed output: O evento tem uma duração de duas horas. <hour> 2:00 </hour>
\end{verbatim}

Figure 3.9: Duration normalization process with the correspondent result from the post-processing layer.

**Dates and Hours** When dealing with dates or hours, the post processing layer divides the full expression in two, date and hour, formatting it according to the normal standards. This results in a \textit{YYYY/MM/DD} or \textit{HH:MM} expression, which is added to the user input inside the \textit{date} or \textit{hour} tag.

3.1.2.2 Temporal Expressions Processing Module

The temporal expressions processing module (Figure 3.10) is a conceptual module made of a group of methods that process the output of the Temporal Expression Normalization Module. These methods have the responsibility to check the consistency of the temporal expressions, calculate and complete expressions with default values when there are missing values, validate possible inconsistencies and, together with the Operations Module, confirm if the user wants to schedule more than one event at the same time.

These methods can be divided in the following categories:

**Validation Methods** Methods that perform validation and acceptance checks on the dates provided by the user, after the normalization process;

**Date Processing Methods** These methods create default temporal values (e.g, 2008), if any value is missing in the normalised date, and turn the normalised version of the dates into programming language level objects;

**Time Processing Methods** These methods have the same behaviour of the Date Processing Methods, but with regard only to hours and minutes;

**Duration Processing Methods** As it happens with dates and expressions regarding hours and minutes, durations have to be processed in programming language level objects. Here durations are
converted from a duration expression based in hours and minutes to a precise date and time in the programming language object level. This is done taking into account the system current time as a reference point.

Figure 3.10: Internal architecture of the Temporal Expressions Processing Module.

### 3.1.2.3 Operations Module

Similarly structured like the Temporal Expressions Normalization Module, the Operations Module in Figure 3.11, has a connection layer to the Google Calendar through Google’s Calendar API, and several operations methods specialised for each device, which are grouped by the type of operation provided.

**Event Scheduling Methods** These methods are responsible for the scheduling of simple events, recurrent events and birthdays on the Google Calendar application;

**Event Listing Methods** Listing is far more simple than scheduling, which means that the only difference between the listing methods available is the type of temporal interval used for listing the events which reduces the number of core methods to one.

**Event Search Methods** These methods are responsible for searching the calendar for an event given its title;

**Event Deleting Methods** This Method is responsible for deleting an event in the calendar giving its title.

### 3.2 Developed Functionalities

This section describes in a detailed way the calendar functionalities that the dialog system provides to the user.
3.2.1 Creating Events

Creating an event is the process of adding a new event to the calendar application. To schedule an event, except in the case of a birthday, the user needs to provide at least the type of the event between a predefined set (generic event, dinner, lunch, meeting), the day and hour of the event, its duration and the title of the event. Additionally, in the case of a recurrent event, the user also needs to provide the end date for the event and its frequency (daily, weekly, monthly or yearly). The request to create an event can use different methods according to the type of event. However, the final request to Google Calendar relies solely on a core method that is used by all the event creation methods. This core method belongs to the Google Calendar API. It allows the creation of all kinds of events supported by Google Calendar.

3.2.1.1 Creating Simple Events

A simple event is an event that is characterised by type, date, hour, duration and title. The definition of a simple event is necessary in order to differentiate single occurrence events from recurrent events, or birthdays. Ideally the user should be able to schedule a simple event only in one sentence, however, this is not possible due to the difficulty of separating the input of a request and the title of an event. So, for the user to schedule an event he needs from three to six steps depending on how much information he gives the system in each step.
3.2.1.2 Creating Recurrent Events

A recurrent event is an event that occurs more than once in a defined period of time. To schedule a recurrent event the user needs to provide additional information to the system, when compared with single events. It is necessary to provide the system with the frequency of the event and its end date. The frequency of the event can be one of the following:

- daily;
- weekly;
- monthly;
- yearly.

The implementation of a recurrent event is more complex when compared with single events. Google Calendar relies on the iCal\(^1\) calendar syntax, as shown in Figure 3.12, to define the recurrence of an event. This requires the construction of a string with the information regarding the event before submitting it to Google Calendar. It is possible to observe an example of an iCal recurrent event in Figure . The iCal format for recurrent events allows the definition of a very broad type of event frequencies of an event. However, it’s not feasible to support all those possibilities with the voice interface system. Supporting this would result in an avoidable and unproductive complexity in the moment of scheduling a recurrent event, which would be easily done by the web interface.

3.2.1.3 Creating Birthdays

A birthday is a recurrent event, which happens every year and has no duration and no hour since we normally consider the whole day has being associated with that event. This is the definition of birthday in the calendar domain, and this is why birthdays are treated in a special way. It is the simplest event a user can schedule in the voice interface system, the user just needs to provide the date and title to schedule someone’s birthday and the Voice Interface System will schedule an annual event with the title provided.

\(^{1}\text{http://www.apple.com/macosx/features/300.html#ical)}\)
3.2.2 Listing Events

Listing an event is an essential functionality of the dialog system, which allows the user to search and list the events for a specific temporal interval. The user has at his disposal the option to list events for a specific hour, day, week, month or year, with the possibility of filtering the type of event the user wants to list. Due to usability requirements, the listing of the events available is limited to a pre-configured number of events that are read by the system to the user in a chronological order. This means that if the user has too many events in a specified temporal interval, he needs to limit the interval to a narrower interval in order to list all the events in the specified interval.

To list the events scheduled within a temporal interval, the user has three options:

- Listing the events in a particular hour;
- Listing the events in a temporal interval;
- Listing the next scheduled events that the user has scheduled.

3.2.3 Deleting Events

Despite not being a vital functionality in a voice interface, it makes sense to support it since the system already supports adding new events and listing them. Deleting an event is an action very similar to listing an event, as the user needs to provide the specific date of the event he wants to delete and confirm the operation. The idea of supporting additional deleting functionalities, like deleting all the events for a single day would result in an unnecessary complexity for the user, resulting in a decrease of system usability. To delete an event the user has two available options, using the dialog system:

**by title** To delete the event by title, the system asks the user its title and enumerates all future events with similar titles. After that, the system asks the user which event he wants to delete, deleting the event associated with the number the user has given.

**by date** To delete the event by date, the system asks the user for the day of the event he wants to delete and enumerates all the events that are scheduled for that day.

In both situations the user is only able to delete one event per interaction. Giving the user the option to delete additional events would introduce an additional level of interaction each time he wants to delete an event.
3.3 **Implementation**

3.3.1 **Normalizing Temporal Expressions**

Most of the normalization process is performed in the $L^2F$ NLP Chain. However, for the dialog system to be able to make use of this information, it is necessary to take several steps. This section describes the steps taken from the speech recogniser to the output of the normalization process, explaining how the dialog system makes use of this output.

3.3.1.1 **Object Identification**

The identification of the user intentions by the dialogue system is performed by the parser system. The Parser Mechanism within the dialogue system is responsible for parsing the user’s input by running a pre-defined set of parses that perform object identification over the dialog.

The extension mechanism provided by the parser system of the dialogue system framework allows the developer to add new parsers that process the user’s input searching for new objects. It is through the development of two additional parsers that, connecting to the $L^2F$ NLP Chain, it is possible to pinpoint temporal expressions and provide their normalised version to the dialog manager.

From the user’s input there are three kind of objects that these new parsers are able to identify:

- Durations;
- Calendar dates, with or without reference regarding the year;
- Time of the day, with or without reference regarding the minutes;

These objects are identified by two parsers which are able to process the same kind of objects. The reason of having two parsers doing the same thing falls over a design decision towards the behaviour of the dialog system regarding the process of the temporal information. To accept a time interval (e.g, “in ten days”) as a date we had to isolate the processing of the durations from the processing of dates in order to minimise clarification requests by the dialog system during a dialog. On the other hand, by imposing this limitation, the user won’t be able to provide the date for an event when the system is asking him for a duration because the system will be only waiting for a duration.

As shown in Figure 3.13, after identifying an object, the parsers add the resulting normalised temporal expression to the user’s input and sends it back to the dialog manager. The dialog manager will then associate the text inside the tags with the correspondent information slot.
3.3.2 The Normalization

The result of the L^2F NLP Chain provides several information regarding the text submitted. Figure 3.14 shows the example of the output from the chain currently used by the dialogue system. Facing this, the first step is to retrieve the necessary information which is associated with the value field of the.NORMALIZATION tag. This field takes different output formats depending on the object identified:

- **Date** Dates are normalised to the YYYYMMDD correspondent expression, where YYYY stand for years, MM for months and DD for days;

- **Time of the day** The time of the day is normalised to the HH:MM correspondent expression, where HH stand for hours and MM for minutes;

- **Duration** Durations are normalised to the A0M0S0D0HM0S0 correspondent expression, where A,M,S,D,H,M,S stand for years, months, weeks, days, hours, minutes and seconds followed by its quantity in numerical form.

After the first step of the normalization process is complete, the Post Processing Layer performs additional normalization on durations, reducing everything into an hourly based form. This reduces the complexity of expressions the Operations Module needs to be able to handle.

3.3.3 Processing Time

Despite the straightforward interpretation of the temporal expressions that result from the normalization process, some level of contextual processing is performed in the Temporal Expressions Processing Module. This is necessary to cover aspects that are not covered directly by the L^2F NLP Chain.

The following temporal expressions processed by this module:

- normalize special temporal expressions that are not directly supported by the L^2F NLP Chain;
replace with default values missing fields in temporal expressions;

convert weekdays to calendar dates;

The following expressions are normalised in this module:

**Current and following day** Any reference to “today” or “tomorrow” is converted to the correspondent calendar date (e.g., “amanhã”/“tomorrow” is converted to 17/09/2009 if the current time is 16/09/2009);

**Weekdays** Any reference to a weekday is converted to the first calendar date correspondent to that weekday (e.g., “Segunda-Feira”/“Monday” is converted to 21/09/2009 if the current time is 16/09/2009);

**next week** Any reference to the following week is converted to the current calendar date plus seven days (e.g., “next week” is converted to 23/09/2009 if the current time is 16/09/2009);

The following situations are subject to verification to reduce the number of necessary utterances between the user and the dialog system:

**Incomplete dates** If the year is missing in the user’s input, the current year is assumed by the system (e.g., 10/09/- - - - is converted to 10/09/2009);

**Incomplete hour** If the hour or minutes field in the hour expression is not mentioned by the user, the field value is truncated to 00 (e.g., 17:- - converted to 17:00);

**Temporal interval as a date** A temporal interval is sometimes used as a date. When that happens, it is translated into a calendar date taking as reference the current system time (e.g., if today is 10/09/2009 the expression “in ten days” is converted to 20/09/2009).

### 3.3.4 Temporal expression replacement for implicit validations

During one of the validations phases, the system was criticised for the lack of implicit validations during the dialog with the user. The dialog system supports implicit validations by defining answer generation templates referring to the information slots, being these references replaced in real time. Figure 3.15 is an example for an answer generation rule used by the dialog system to produce a sentence with information to the user. The problem of adding implicit validations is to replace the information slot reference used by the dialog system during the dialog, with the result of the temporal expression normalization process. This process works seamlessly with non temporal information slot, but not with dates, hours or durations. This happens because different expressions can be used to reference the same time. The
answer generation rules do not allow any kind of dynamic behaviour, meaning the developer cannot choose which sentence the system is going to use when there are more than one sentence with the same information slot variables. Figure 3.16 shows the example of a situation where the normal implicit validation system would not work. In this example, the user is scheduling a meeting in ten days. A logical implicit confirmation would be “[action] [target] in [date]”. However, if if the user specified a calendar date, the same sentence would not make any sense. The solution was to replace the temporal expressions for their normalised expression on the fly, which means that the answer generation must use the normalised version of the expressions. This workaround was developed using the validation system of the dialog system. Using the validation features, the dialog system replaces in real time the expression that is filling the slot by the correspondent normalised version.

3.3.5 Validating time

The validation of the information exchanged between the user and the dialog system is performed in order to avoid incorrect requests. The Temporal Expression processing module, making use of the val-
idation mechanism provided by the dialog manager, responsible for the contextual validation. Each validation is associated with one or more information slot that belongs to each service. This means that each service has associated its own validations methods according to its specification.

There are three types of validations that can be performed during the execution of a service:

**Syntax consistency** Durations are checked against its expected format, when the dialogue system asks for a duration, if the user replies with a date, then it is necessary to ask for the user to resubmit a duration;

**Semantic consistency** The L²F NLP Chain does not perform any semantic consistency check on the temporal expressions. This opens the door for the normalization of expressions that do not make sense (e.g. “...40th of January” converted to 40/01/2010). To avoid it, the dialog system performs a semantic validation to every date and hour, notifying the user to submit a new date or hour if the current one is invalid;

**Temporal consistency** If the user wants to schedule an event, it does not make any sense to schedule it in the past. Also in this case, the dialog system compares each date against the current system time to be sure it is a future date;

**Event Duplication** This is a special validation method created to verify if the user is about to schedule an event on top of another, this validation is used to warn the user about the double scheduling and to give him the option to redefine the date of the event.

Table 3.3 shows the validations performed for each type of functionality that Google Calendar provides.

<table>
<thead>
<tr>
<th>Validations</th>
<th>Scheduling</th>
<th>Recurrent Scheduling</th>
<th>Birthdays</th>
<th>Listing</th>
<th>Searching</th>
<th>Deleting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validate Durations</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validate Dates</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Validate Hours</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare Dates</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duplicate Verification</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation

Evaluation is an important and necessary part of any development project. It allows project developers to know what was achieved from the proposed objectives and what should be further developed on other iterations. The evaluation of spoken language dialog systems (SLDS) has become a respected research subject which is given critical importance. SLDS evaluation (Bernsen & Dybkjae, 2007) can be decomposed into (i) technical evaluation of systems and their components, (ii) usability evaluation of the system, and (iii) customer evaluation of the system and components. However other aspects can be taken into account in the evaluation process of a SLDS, like cost and platform compatibility which have little to do with technical perfection or user satisfaction.

4.1 Evaluation Objectives

For the evaluation of this dialog system we present the following objectives:

- identify the objectives achieved;
- identify possible design problems;
- identify performance problems;
- suggest new ways to increase overall performance;

The dialog manager is the main target of this evaluation since it is responsible for the execution of all functionalities on Google Calendar. Both speech recognition system and speech synthesiser can introduce additional problems, facing this, it is necessary to test the dialog manager independently in order to understand how much entropy is introduced by the voice components of the dialog system. However, this does not mean that the voice components are going to be given less importance during the evaluation process.

4.2 Evaluation Methodologies

Several studies have been conducted regarding the evaluation of dialog systems, these gave origin to two major frameworks for the evaluation of SLDS:
• Best Practice in Spoken Language Dialogue Systems (DISC) (Bernsen et al., 1999) (Bernsen & Dybkjaer, 2000) (Bernsen & Dybkjaer, 1994);

• PARADISE (PARAdigm for Dialogue System Evaluation) (Walker et al., 1997) (Charfuelan M. and Lopez C. and Gril J. and Rodriguez, C. and Gomez, L., 2000);

In this master thesis we will analyse both evaluation frameworks and choose one of them as the grounds for the final evaluation process. Done that, we will define an evaluation procedure for the final evaluation phase.

4.2.1 DISC

The European Esprit Long-Term Research project on “Spoken Language Dialogue Systems and Components, Best practice in development and evaluation (DISC)” was initiated by ELSNET1 on 1 June 1997, lasting until 31 December 1998 but immediately continued on the DISC-2 project until 31 December 1999. The first project had the objective to develop the first dialogue engineering best practice methodology for the evaluation of dialog systems. The second phase of DISC, called DISC-2, focused on testing the validity and the usability of the draft best practice scheme, the concepts, tools, integration, packaging and dissemination of the final best practice scheme.

During the development of the DISC project, it became clear that SLDS evaluation deserved more attention and effort than originally planned. In the DISC approach, an SLDS has six components that should be taken into account: speech recognition, speech generation, natural language understanding and generation, dialogue management, human factors, and systems integration. In simple systems, the natural language understanding and generation aspect may be non-existent or not relevant but the five other aspects should be present for the system to be a full featured SLDS.

To efficiently perform an evaluation procedure on an SLDS, the DISC project developed an evaluation template to support the how of the evaluation process. This template is a model of what the developer needs to know to evaluate a particular property of an SLDS (e.g, noise models used by the recogniser). This knowledge is specified by the template, with ten entries numbered from 1 to 10. Depending on the objective of the template, there are three different versions of the template including different information:

1. The basic template presents and defines the ten entries. The basic template is not meant to be filled with specific information but to support the understanding of how to use the template for evaluation purposes;

1http://www.elsnet.org/
2. The empty template includes the ten entries of the DISC evaluation model. The empty template is filled with information for specifying particular evaluation criteria. An empty template has to be filled for each property;

3. A filled template specifies a particular evaluation criterion.

The basic template is made of ten entries described below:

**What is being evaluated** This entry describes the property or properties of an SLDS or component that is being evaluated (e.g., speech recognition success rate). When dealing with generic properties, the evaluators using the template will have to define the appropriate additional specifications of the properties which they want to evaluate;

**System part evaluated** This entry describes which components of an SLDS are being evaluated (e.g., parser system, speech generation component, the system as a whole);

**Type of evaluation** This entry describes the type of evaluation. The evaluation can be quantitative, qualitative or subjective and if the evaluation is comparative. It is preferable to obtain quantitative results from the evaluation, allowing performance and progress measuring, and comparison with other SLDS evaluated. However, many characteristics cannot be subject to quantification;

**Method(s) of evaluation** This entry describes the evaluation methods used in the various stages of the evaluation process. In early stages of the evaluation process the evaluation tends to be more conceptual than based in real data. However, in later development stages the data analyses and capture is takes much more relevance in the evaluation process;

**Symptoms to look for** This entry describes what the evaluator will look for in the evaluation data during the evaluation process (e.g., lack of understanding by the system);

**Life-cycle phase** This entry describes the life-cycle phases where each property starts to be evaluated;

**Importance of evaluation** This entry describes the importance of evaluating a specific property. The importance of evaluating some aspect of a dialog system may depend on its relevancy and on the importance for the system’s objectives. This description can be described as low, medium or of high importance;

**Difficulty of evaluation** This entry describes the difficulty in performing the whole evaluation process. This entry depends on various forms of complexity, such as task complexity, user’s input complexity, dialog manager complexity or overall system complexity. This entry also depends on whether the system suffers of unsolved problems that interfere in the evaluation process;
Cost of evaluation  This entry describes the cost of the evaluation process. Cost can be defined in terms of money, time, manpower or skilled labour. It can be difficult to measure and evaluate dialogue system, mainly because of the quantity of data involved or the complexity of the dialog system;

Tools  This entry describes the software tools and utilities used in aiding and performing the evaluation process.

Although still in development, the DISC filled template supports various types of evaluations criteria. However, an evaluation criteria represented exclusively by a filled template does not prove to be useful since the template as is, is not self explanatory due mostly to terminology issues.

4.2.2 Paradise

PARADISE is a general framework for spoken dialogue system evaluation. This framework was created taking into account the difficulty in evaluating and comparing dialog systems with different dialog strategies. The Paradise framework separates task requirements from dialog behaviour, supports comparisons between different dialogue strategies, performance and task complexity measurement. Performance is seen as the result from a weighted function that takes into account a task based success measurement and dialogue based cost calculation, where weights are computed by correlating satisfaction with performance.

Paradise’s performance model is based on decision theory to combine disparate performance measurements into a single performance evaluation function. This requires the previous specification of objectives, decision problems and measures in order to operationalize the objectives of the evaluation.

The framework takes as its objective to maximise the user satisfaction, considering this as the primary goal of the evaluation procedure. The framework defines a structure of objectives based in this goal:

Maximise task success  Using the Kappa coefficient\(^2\) it is possible to operationalize task success. This value makes use of linear regression to quantify the relative contribution of the success and cost contributions to the user satisfaction calculated;

Minimize cost  Relating efficiency measures (i.e. number of utterances, dialogue time) with qualitative measures (i.e. agent response delay, repair ratio, etc..) it is possible to define and understand what is potential relevant to user satisfaction.

\(^2\)http://www.linguatca.pt/aval_conjunta/Faro2002/HTML/Renata_Vieira/tsld007.htm
Additionally, the framework uses an attribute value matrix to represent the dialogue tasks and compare them with each other. This AVM (attribute value matrix) records the information that must be exchanged between the dialogue system and the user during a dialog. This is represented as a set of ordered pairs of attributes and their possible values.

In a short description the Paradise methodology requires the developer to perform the following steps:

1. define the tasks and scenarios to be evaluated;
2. specify the attribute value matrices according to the tasks defined in the first step;
3. experiment the tasks defined with different dialog agents/dialog systems in order to have comparative results;
4. calculate user satisfaction using surveys;
5. calculate task success using the Kappa value;
6. calculate dialogue cost and efficiency using qualitative measures;
7. use linear regression, user satisfaction, dialog cost and the Kappa value to estimate the performance function;
8. compare the values obtained with other tasks in order to determine which factors are general in the evaluation;
9. refine the performance model with the values obtained.

A set of quantitative and qualitative metrics are defined by the Paradise framework on the evaluation of dialog systems:

**Quantitative measures:**

- Ratio of concluded tasks;
- Ratio of correct answers;
- Number of utterances;
- Dialog duration;
- User’s average answering time;
- System’s average answering time;
• Ratio of confirmations;
• Dialog size.

**Qualitative measures:**

• Ratio of sub-dialogues created for implicit dialog recovery;
• Ratio of sub-dialogs created for explicit dialog recovery;
• Ratio of system messages contextually inappropriate;
• Ratio of correct or partially correct answers;
• Ratio of incorrect or out of context answers.

This are the measures that the Paradise framework takes into account when evaluating a dialog system using the tools defined above (i.e. attribute value matrix, Kappa coefficient, etc...).

### 4.3 Evaluation

Comparing both evaluations frameworks it is a complex task since each evaluation framework takes a different approach at the dialog system evaluation process. The DISC approach looks the evaluation process since the beginning of the development process while the Paradise can be used at later development stages. Facing this, we think that Paradise is the evaluation process with more advantages, mainly due to its evaluation flexibility (i.e. sub-dialog evaluation, task complexity is taken into account in the evaluation process, performance measurement combines both qualitative and qualitative aspects). However, its task based success measure does not reflect if some solutions are better than others because the evaluation does not consider the dialog system as a whole. The Paradise framework presents a considerably more defined set of rules than the DISC process. This facilitates the use of the former evaluation process at late stages of development.

#### 4.3.1 Evaluation phases

To perform the full evaluation of this dialog system we decided to divide this evaluation in two distinct phases:

**preliminary evaluation** The first evaluation phase that the project is subject to. It is performed over a small user group and aims at identifying early development and design problems, resulting in easier correction and redesigning;
The final evaluation process that the project is subject to. This phase is done with a considerable amount of users and requires a more complex evaluation process. Contrarily to the preliminary evaluation phase, this phase focus on identifying if the project objectives were met and evaluate the performance of the dialogue system following a more formal evaluation process.

4.3.1.1 Preliminary Evaluation

The preliminary evaluation phase was performed over a group of six users. Some of these users were already familiarised with the dialog system framework, which probably helped them achieve the objectives of the test cases easily. The evaluation phase did not follow any particular evaluation framework, however the evaluation tasks given to the user were taken from the evaluation process of other dialog systems based on the DIGA framework.

This evaluation process consisted on five tasks which the user had to perform over the dialog system. Since the system was still under development, the tasks target exclusively two main functionalities, scheduling and listing.

4.3.1.1.1 Evaluation Tasks and Process In this evaluation phase the users are asked to perform the following tasks over the dialog system:

- Schedule a meeting for the 20th of April at 7 PM, with an arbitrary length;
- Schedule a diner for tomorrow at 8 PM, with an arbitrary length;
- Schedule a meeting that takes place in 10 days for an arbitrary hour with an arbitrary length;
- Schedule a daily meeting, starting at 15th of June, 3 PM and ending on the 16 of September, with an arbitrary length;
- Schedule your own birthday;
- Find which is the next event scheduled;

While performing the tasks, for each task, a developer tracked the user progress, making records of the following aspects of the evaluation task:

- Time taken to execute a task;
- Number of interactions;
- Number of errors;
- Number of times the user asked the developer for help;
This measures allowed us to understand how well the system is performing, independently of the capacity of the user to adapt to the dialog system.

After performing the tasks the user is asked to fill a small questioner about the dialog system and the evaluation process. This questioner covered the following aspects of the evaluation process:

**User Satisfaction** The user satisfaction while executing the given tasks;

**Understanding the dialog system** User understanding and interpretation over the feedback the dialog system gave;

**Task execution** How difficult was for the user to execute each task;

### 4.3.1.2 Final Evaluation

The final evaluation phase followed the directives of the Paradise evaluation framework. This framework was chosen because it allows the evaluation of the dialog system independently of the steps taken during the development process. Additionally, the paradise framework directives on the evaluation of dialog systems provided a step by step evaluation process which facilitate the evaluation of this dialog system.

#### 4.3.1.2.1 Evaluation Process

Like in the first evaluation phase, the user will be asked to perform a series of tasks on the dialog system and after that to fill a small questioner. The user performance will be recorded on video and audio to allow an additional analysis of his performance.

**Evaluation Focus** This evaluation process will focus on the following characteristics:

**System perception over the user’s intention** Analysing the system perception over the user’s intention aims at evaluating how well the system is understanding what the user is saying and what the user wants. In one hand, this evaluation characteristic aims at evaluating the speech recogniser capabilities: how many words are well recognized; how many recognition errors; how many recognition errors affect the task; how is the system recovering from these errors. On the other hand, this characteristic also aims at evaluating how the dialog system is interpreting the user intentions.

**User’s perception over the dialog system** Understanding what the dialog system is saying or asking to the user is as important as the system recognizing what the user is saying. This characteristic aims at evaluating specifically the speech synthesiser, its speed, user perception and influence on overall user experience.

**User Satisfaction** Satisfying the user is the prime goal of any system, if the user satisfaction is not positive, the system won’t have any user acceptance. This characteristic is evaluated with the use of
questionnaires. The user satisfaction is related not only with the system performance and capabilities, but also with the user expectations over the system and previous background experience. Evaluating user satisfaction can be rather subjective depending on the user group which is evaluating the system.

**Reaction time** The time the system takes from recognising the user’s input. Many things can influence this, the system needs to interpret the user’s intentions, process them and generate an answer, any of this phases can affect considerably the user’s satisfaction and the system usability if they take too long to complete. Additionally, using the voice to access the calendar should not take much more time that using the correspondent web interface.

**User experience with dialog systems** The user’s experience with dialog systems can influence deeply his performance. How to adjust the voice, or speak slowly can greatly improve the performance and consequently affect the user satisfaction.

### 4.3.1.2.2 Evaluation Tasks and Process

**Evaluation Tasks** In this evaluation phase the users are asked to perform the following or similar tasks over the dialog system:

1. Schedule a meeting for the 14th of July of 2009, at 20PM, that lasts five hour;
2. Schedule a meeting for the 25th of June, that lasts one hour and thirty minutes;
3. Schedule a meeting in two weeks, at 2 PM, with a duration of five hours;
4. Schedule a daily meeting, which takes place in five days and ends on the 5th of September at 11AM with a duration of two hours;
5. Schedule a dinner for the 14th of July at 8PM that lasts three hours;
6. Schedule a meeting for tomorrow, 12AM, for four hours;
7. Schedule a meeting for the next monday, 9AM, lasting three hours;
8. Schedule a in three hours, lasting thirty minutes;
9. Schedule your own birthday;
10. List the next event in the agenda;
11. List all events for today;
12. List the events for the 14th of July;
13. List events for next Wednesday;

14. Search for the first event you scheduled using its title;

15. Delete one event on the 14th of July;

**Evaluation Measures**  Quantitative and qualitative measures will be used to evaluate the user performance. The following measures will be used during this evaluation phase in order to evaluate the evaluation focus previously defined, each of this measure will be applied for each task independently.

**Time taken to fulfil the task**  Time the user took to complete successfully or unsuccessfully the respective task;

**Task Completed**  If the user was able to complete the task in the designated time frame;

**Number of utterances used by the user**  Number of utterance the user exchanges with the dialog system in order to complete the task proposed;

**Number of utterances used by the system**  Number of utterance the system generated in order to complete the task proposed;

**Number of words used by the user**  Number of words the user takes to make his requests;

**Number of words generated by the system**  Number of words the system uses to communicate with the user;

**Number of errors due to lack of understanding**  Number of errors occurred due to an incomplete speech recognition;

**Number of errors due to total miss understanding**  Number of errors occurred due to a total failure in the speech recognition system;

**Number of fatal errors**  Number of errors that required the assistance of a developer;

**Number of assistance requests**  Number of requests for assistance;

**Number of situations with unexpected behaviour**  Number of occasions where the system performed unexpectedly;

**Number of answers with errors**  Number of answers given by the dialog system which present errors regarding what it was asked by the user;
**Questionnaire**  In the end of the evaluation process the user is invited to fill in a questionnaire after performing the tasks of the evaluation exercise. This questionnaire is similar to the one used in the preliminary evaluation phase, following the proper updates regarding the number of questions and the additional focus and objectives of this evaluation phase, maintaining the questionnaire main objective of evaluating the user satisfaction.

Questions to test overall user satisfaction:

- Did you like using the Google Calendar Voice Interface?
- Did you find it difficult to use the system?
- Would you consider in using this application?
- Did the system met your expectations?
- Did you felt lost, at anytime, in the conversation without knowing what to say?

Questions to understand the user perception on the voice capabilities of the system:

- Did you understood what the system was saying to you?
- Did the system understood your requests?

Questions to understand how the performance is affecting the user satisfaction and what could be improved;

- Are you satisfied with the dialog system performance?
- Did you find the conversation rhythm appropriate?
- What would you improve from the following characteristics: speed; understanding; dictation; functionalities?

4.4 **Results**

The evaluation proved to be quite hard mainly to the long list of tasks devised for the final evaluation. The speech recogniser proved unstable during the evaluation tests which directly affected the user satisfaction and the results. Besides this, one of the users drop the evaluation phase due to frustration regarding voice recognition problems.
4.4.1 Analysis

The results from this evaluation phase were rather mixed depending on the user, meaning, that there were users who performed well above average. The results using the speech recogniser reached disappointing levels in some situations. However, since we were not able to reach our target evaluation in terms of users the results tend to be not so statically meaningful.

The user’s background with similar dialog systems proved to have some influence while performing the required tasks among the testers. This was even more noticeable if the user was using the voice interface.

Figure 4.1 shows a comparison between the time taken to complete a task using the voice interface and using the text interface. As it is possible to observe, the difference between the time taken can be quite significant in some tasks. This difference happened for two main reasons: the time taken by the user to hear and the time taken in recovering from recognition errors. Alongside to the increase of the time to complete a task when the voice interface is used, is the increase of the number of iterations. This happened due to extra iterations for error recoveries. Figure 4.2 shows a comparison between the number of iterations to complete a task using both interfaces. It is interesting to see that in some situations the text interface took more interactions than the voice interface, taking however less time to complete the same task.

Although when using the voice recogniser the system tends to be more prone to errors, it does not mean that errors won’t happen the text interface is used. Since the voice recogniser grammars work as a filter for spelling errors, it shields the user from spelling mistakes, whereas they can happen often when using the text interface.

The fact that the voice recogniser failed to recognise single words (e.g, “hoje” in task 8) resulted in an error which affected negatively the test results in this and other simpler tasks. Additionally, this problem increased the number of iterations used to complete the task since most of the test subjects repeated the word at least one more time before moving to different expression.

Despite of the high number of recognition errors in some of the tasks, as shown in Figure 4.3, there was an user that was able to fill most of the tasks with hardly any recognition problem. Although with little statistical ground to prove, we can assume that an experienced user would be able to use the system with a much higher success rate.

The dialog system revealed some lack of stability during the test phase, specially when using the speech functionalities. In total there were fifteen failures which require developer intervention during the test phase. From this failures, fourteen happened during tasks with the voice interface and only one during the use of the text interface.
Figure 4.1: Average time for task completion in seconds, blue describes the tasks performed with the text interface and red with voice interface.

Figure 4.2: Number of iterations to complete a task, blue describes the tasks performed with the text interface and red with voice interface.

Figure 4.3: Average number of speech recognition errors per task.
The user satisfaction was tested using questionnaires handed out to the users after they completed the tasks. This questionnaires show that the users who used the voice interface were less satisfied with the system and considered it harder to use when comparing to the ones who used the text interface. However, it is interesting to see that the users complaint over the same tasks disregarding if they were performed by voice or by text.

The major complaints of the users fall over two categories, speed and perception. The users who tested the text interface complaint over the time the system takes to answer. However, the users who tested the voice interface prefer a better perception of their speech to an increase of the dialog speed.

During the evaluation we noticed two major mistakes in this evaluation process. The long task lists, which demoralised the users during the test phase and greatly affected the number of test subjects and the long welcome messages. The later one, proved to be rather annoying for the users who were testing the voice interface.
In this master thesis we addressed a series of research objectives regarding the development of a dialog system that makes the bridge between the user and the web interface. This research and the achievement of the proposed goals was met through a focused bibliographic research on similar dialog systems as well as the analysis of dialog systems based on the same platform. Followed by the design and development of a solution with correspondent evaluation with a test group.

5.1 Meeting the Objectives

In the beginning of this research we proposed to meet the following research objectives:

1. Contextualize this research project, the dialog system framework used and other tools and terms applied in this research;
2. Produce an overview of the current approaches and dialogue systems that deal specifically with calendars;
3. Produce an overview of dialog systems developed with the DIGA framework;
4. Build a domain model that support most common interactions with an online calendar;
5. Develop an interface between the dialogue system framework DIGA and the Google Calendar\textsuperscript{1} using Google Calendar API;
6. Develop an interface between the DIGA dialogue system framework and the L\textsuperscript{2}F NLP chain;
7. Normalize expressions regarding time intervals and time frequencies using the L\textsuperscript{2}F NLP chain;
8. Extend the dialogue system’s general grammar to the calendar domain;
9. Perform evaluation tests to the prototype.

Chapter 1 addresses the first goal introducing the context behind this research project in the first section, specifics and the tools used are described in the remaining sections. Chapter 2 focus on the

\textsuperscript{1}http://calendar.google.com
second and third goal, concerns with the review of the main dialog systems developed with similar purposes alongside the review of dialog systems based on DIGA framework. Chapter 3 addresses most of the goals defined for this project, leaving only the evaluation goal for chapter 4. Section 3.1.1 explains how the new functionalities were developed using the DIGA framework and the correspondent domain model. Section 3.1.2 explains how the L²F NLP chain was integrated with the DIGA framework and integrated with the dialog system to normalize temporal expressions. Finally, section 4.3 describes the results of the evaluation performed with this dialog system.

5.2 Problems

5.2.1 DIGA Framework

During this master thesis, we tried to be mere clients of this dialog system framework. This allowed us to review the framework in the perspective of a potential client, pointing what went wrong during the process of developing a dialog system using this framework.

5.2.1.1 Dialog Manager

Regarding the dialog manager there were problems which introduced additional complexity in the development of this dialog system.

5.2.1.1.1 Sentence generation The phrase generation template does not allow the developer to define which phrase he wants the system to use from the available templates, this means that, if the system has two phrase templates with the same slots he will use one arbitrarily. Some customisation would allow the developer to create phrases which could be more natural according to the situation in question. Figure 5.1 shows an example of a phrase generation, in this case, if the expression “next Monday” was not translated in dialog time to its normalised version, the question to the user would make no sense. If it was possible to dynamically the answer the system was going to say from the ones available according to the input, then it wouldn’t be required to convert the user expressions in the correspondent normalised versions during the dialog.

5.2.1.1.2 Parser development The necessity to identify new objects in the user’s input which are not natively supported by the dialog system framework resulted in the need to develop additional parsers. During this process, we identified the following problems:

Slot Type definition It is not possible to define custom slot types using just configuration files. The dialog manager does not offer any possibility to support new slot types without defining them in the source code. In the case of this dialog system, the slot type Time and Date were already
Figura 5.1: Example of phrase generation.

supported by the dialog manager, however there is no slot type for Durations. This is an additional reason to be using the slot type Text when dealing with durations.

5.2.1.3 Dialogue synchronization The system presented synchronization problems between the dialogue system and the voice components. This happened because the system only starts hearing the user when the voice syntethizer ended speaking. However, if there is a double output from the system, the voice syntethizer will play both outputs, but the end of the first one makes the voice recognition system start trying to hear the user. Since the user is still hearing the system, he will wait for the system to stop in order to start speaking. As the user does not say anything, the system will assume a timeout and will ask the user to repeat. This generates a miss synchronization between the dialog manager and the voice components of the dialog system.

5.2.1.4 Implicit confirmations Implicit confirmations was the chosen confirmation method used in this dialog system. However, this implies that, for each interaction the user needs to hear much more information from the dialog manager when comparing with other confirmations.

The problem with the implicit confirmation in the dialog manager is that it is not possible to force the dialog system just to confirm one slot once. This results in situations where the user hears the date of his event up to three times, which can be annoying and consequently damage users satisfaction and system usability.

The dialog system framework allows the developer to fully control how the phrases are generated by the dialog system and thus allowing the use of implicit confirmations. However it does not allow to specify how many types can that slot be presented to the user during an interaction. A possible way to solve this situation. Allowing the developer to specify how many times a slot could be used for phrase generation, or when it could be used, could result in an effective solution to this problem.

5.2.1.2 Speech Recogniser

The speech recogniser presented several problems during the development of this master thesis.
5.2.1.2.1 Short words as no speech  
The speech recogniser had problems to recognized single words when they were too short. If in a previous interaction the system missed a word related to a day of the week, in the next interaction the user would have to repeat everything. If not, the system would not recognise it since it was a single word. This problem was detected during the evaluation process, where users were trying to recover from recognition errors by saying key words (e.g., Monday), and the speech recogniser was treating the single word as no speech.

5.2.1.2.2 Recognition problems  
During the development and tuning of the speech recogniser, we faced several problems. Despite the different results obtain with different users, which is expected, there were problems regarding the recogniser behaviour. The speech recogniser failed to follow some of the specifications in the grammar, which in some cases caused additional recognition errors during the evaluation process. Figure 5.2 shows the example of the grammar used for date recognition by the speech recogniser, facing this example, the user should be able to say a date, or an hour plus date. However and despite of the low weight given to the hour and date option, the recogniser would always choose the option in detriment to recognising a single date.

```
<rule id="data" scope="public">
  <one-of>
    <item><ruleref uri="&tasksModelsDir;\dates.xml#date" /></item>
    <item weight="0.05">do</item>
    <item><ruleref uri="&tasksModelsDir;\hours.xml#hours" /></item>
  </one-of>
</rule>
```

Figure 5.2: Example of grammar used by the speech recognizer for date input.

5.2.2 L²F NLP Chain

During the development of this project we came in touch with several issues regarding the normalisation of temporal expressions.

5.2.2.0.3 Normalisation issues  
Although very rare, there were some normalisation issues in particular situations which affected the normal behaviour of the dialog system. Some of this issues are described below:

19:30 - 18:30  
If the user asked to schedule an event to 19:30, the normalisation of the hour would be shifted one hour before to 18:30;

Encoding problems  
This was a re-incident issue during the integration of the chain with the dialog system, mainly due to the portuguese language. During the development phase we came across
an internal problem of the chain when dealing with the word three in portuguese and other accented words, like March (e.g três, Março);

Both of this problems were solved during the development phase by the developers behind the L²F NLP Chain. Since we were dealing with a proof of concept system, both problems were not critical since it did not affect directly the user experience or system evaluation;

5.2.2.0.4 Performance issues As mentioned before, the performance of the L²F NLP Chain is an important issue which was not completely addressed until the end of this master thesis. The fact that it was not possible to identify if the user was saying any temporal expression is one major draw back which requires the use of the chain for every input, slowing down the whole dialog process. The use of the cache previously mentioned only proved successful in two occasions:

- development phase were the use of the same expression was common;
- small commands with no temporal expressions (e.g schedule meeting);

5.3 Future Research and possible developments

Taking into account the tools and systems used during the development of this research project we can consider different approaches of future work according to the tools in question or the combination of them:

- DIGA Framework;
- L²F NLP chain web-service;

5.3.1 User identification and authentication

This project was developed over a single google calendar account. This means that, at the moment there is no mechanism to support user selection or authentication. A production version of this project would required some kind of authentication mechanism to allow multiple users. There are several possibilities to allow user authentication:

phone id association Relying in associating google account details with a phone number id should be one of the easiest way to authenticate a user. However being the easiest way, it would be also one of the most unsafe ways of performing user authentication since it is not that hard to fake a phone id number;
**pin control** Associating a pin with a google account details with a digit sequence should be one of the simplest ways to introduce user authentication;

**phone id association with pin control** Associating a pin code would add an additional security layer to the authentication process, however the user would be limited to check his calendar by a limited set of phone numbers;

Additionally, a configuration application should allow the possibility to add additional google accounts to the dialog system.

### 5.3.2 Taking a step further

The $L^2F$ NLP chain allows advance dialog analyses and could replace the role of the dialog manager in the interpretation of the user intentions. This would allow the recognition of speech acts instead of just key words for the identification of the user intentions. A speech act is an utterance that serves a function in communication (e.g. A greeting... ).

In our opinion this could be done in two ways:

- replacing the generic parser, with a custom made parser that would connect to the chain web-service, and together with a domain specific grammar on the chain side could be possible to identify the user iterations regarding that domain. This would be the less evasive approach and would still leave some responsibility to the dialog manager in terms of user input processing, however, it will require additional effort in synchronising both components;

- integrating the chain directly with the dialog manager would require a complete bypass of the parser system and a total dependency of this new tool in order to identify objects on the user input. This approach would definitely present a better performance, although it would have a harder implementation that the other approach.

Additionally, transferring this responsibility to the chain would required the developer to rethink on how to integrate the chain with the dialog system, mainly due to performance issues. The use of the chain in its current model, can present serious performance issues which would make it unusable in a production environment.

### 5.3.3 $L^2F$ NLP chain web-service 2.0

This web-service was created to allow the development of this master thesis. It was created with a proof of concept in mind, disregarding performance issues. The cache used is a workaround to speed up the
debugging process due to repetitive static commands (e.g. schedule, schedule a meeting). Currently, the chain has several performance issues due to how the tool is being executed. For each processing request, each part of the tools is executed independently. This means that, each component and respective libraries are loaded into memory and executed for each call to the chain. This could be avoided by having each component of the chain to function as services that would be always online, answering to independent requests. This could prove to be a much more scalable solution than the current one, especially for production environments, where the chain would be processing more than one request at a time.

5.3.4 Going Mobile

Until now, it has been possible to access dialog systems on the move by calling phone numbers and interact with them using the voice connection. This solution has the advantage that all processing is made on the server side which is running the dialog system, meaning we can access the dialog system using a cheap out of the shelf cell phone. However, it is necessary to pay the call, and it is necessary to have a server that is capable to handle a considerable amount of users concurrently. Using the dialog system directly on a PDA or cell phone, would necessarily require a power cell phone and internet connection, but it would not require the user to spend money on the phone call nor a server to handle all the users as long as it was an internet enabled phone.

The MiPad PDA dialog system is device which supports a mobile dialog system. This PDA dialog system was heavily constrained by the use of an external server. This server was responsible for its speech functionalities, meaning voice recognition of the user commands and text input was done in the server. Continuous developments in mobile hardware has resulted in considerable improvements both in processing power and memory capacity. With the recent developments on mobile versions of speech tools, like speech recognition systems (Miranda, 2008), it becomes feasibility to develop stand alone mobile dialog systems in the near future. The possibility of porting Google Voice Interface to a mobile device is an open possibility, we suggest three ways to achieve that:

**Using a phone line** Setting up the dialog system behind a PBX would make it accessible by calling a phone number, since the DIGA framework already supports this I/O system;

**Using the PDA as a voice entry point** Porting both speech recognition and synthethizer to the PDA and leaving all speech processing in an external server web service would require the PDA to handle just with the speech recognition and synthesis;

**Porting the whole dialog system** This would require that all components of the DIGA framework to be ported to a mobile environment, which means the PDA would have to be considerably powerful.
5.4 Final Remarks

In this dissertation we have addressed a series of research objectives regarding the development of a dialogue system that interacts with an online calendar. These objectives were met through an analysis of dialog systems with similar objectives, followed by the identification of the tools used and by the design and implementation of a solution. The project was subject of two evaluation phases which aimed at identifying design problems and determine user acceptance


