Dialogue Management for multilingual communication through different channels

Meritxell Gonzàlez Bermúdez
`mgonzalez@lsi.upc.edu`

Director
Marta Gatius Vila

Memòria del DEA i Projecte de Tesi
Programa de Doctorat en Intel·ligència Artificial
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Juny 2007

Contents

Chapter 1

Introduction

1.1 Dialogue Systems and Dialogue Management

Nowadays, Dialogue Systems can handle friendly and collaborative communication. For this reason, they seem useful for accessing different types of applications. Dialogue Systems support several types of interactions, such as menus where the user is asked to choose an option, form-filling where specific information is asked to the user, commands for expressing user's orders, and complex questions that can even reference to previously evoked entities. For these reasons, Dialogue Systems are useful for many applications. Particularly, Spoken Dialogue Systems are specially appropriated for devices not allowing web browsing, such as telephone, or the use of hands, as vehicle GPS.

Practical spoken systems are mostly adapted to the functionality of a specific application because voice technology still presents recognition problems in open domains. The cost of developing an application-restricted Dialogue System is high because experts are needed. Moreover, these systems are difficult to adapt to new applications. For these reasons, the problem of developing Dialogue Systems reusable for different applications and languages has aimed many research works. There already are works on Dialogue Systems that support intelligent communication, use domain and dialogue models, and have reusable domain-independent components of discourse management and language/speech tools. However, the cost of adapting them to new applications is still high.

My work is particularly concerned with Dialogue Systems for guiding the user to access the web. Conversational Systems that guide the users to access web services can improve the usability and accessibility of web contents. Web interfaces need to be more interactive and adaptable to different types of users, services and channels since the need of enhancing the web usability is increasing. There already exists several works in this line, such as web voice interfaces, and collaborative agents that helps the user to interact with the web services. There is a well known standard formalism used in many Spoken Dialogue Systems, the VoiceXML language. The VoiceXML standard is appropriate to define the flow of system-initiative dialogues, where the system asks the users to give the specific information the service needs. However, VoiceXML systems only support very limited user-initiative (the user can only choose the order in which the information asked by the system is given) and they do not support complex dialogue phenomena, such as clarification. Furthermore, the dialogue flow in VoiceXML systems has to be defined for each service.

My work is focused on Dialogue Management, which is concerned mainly with conversational control and guidance, i.e. turn taking control, keeping track of the dialogue states and deciding the next system's actions. The Dialogue Management tasks are basically three: understand what the user's intention is, determine what should be accomplished in response to the user's utterances, and generate Natural Language sentences to achieve the selected goals. Early Dialogue Managers were simple state automata representing all the possible interactions. Dialogue Managers in recent research Dialogue Systems are complex models and use plans to deal with the user's goals. A proper design of the Dialogue Manager component improves communication and reduces the cost of adapting the system to a new service domain.

The main goal in commercial Dialogue Systems is robustness. For this reason, most of the Dialogue Managers only support system-initiative dialogues asking the user the information needed to accomplish the task. In such approaches the Dialogue Managers use a specific dialogue flow for each service in which there is an explicit description of the relation between the states and the participant's actions. However, more complex Dialogue Managers improve the communication and the engineering process of adapting the system to new services using explicit dialogue models and defining general strategies to decide the next system actions. The use of explicit dialogue models improve communication and portability of the Dialogue System.

1.2 The aim of this work

The work presented is concerned with the development of multilingual and multidomain Dialogue Systems, using a domain and language independent Dialogue Manager component, and the (semi)automatic generation of the system's responses. Particularly, it is concerned with the definition of strategies for improving communication and reducing the cost of adapting the system to other services.

The Dialogue Manager may support different types of dialogue initiatives. On one hand, the Dialogue Manager may drive the interaction with inexperienced users, asking them all the information the service needs. On the other hand, experienced users may take the initiative giving information not only about what the system asked for, but also about other aspects. The task of the Dialogue Manager become more complex when dealing with user-initiative dialogues. The Dialogue Manager should cope with the user's interventions that give more, less and/or different information than requested as well as with the user's interventions that ask for information not related to the last system's intervention.

The Natural Language Generator module may provide natural and efficient communication messages to the Dialogue Manager. Feedback strategies improve the communication. For this reason, the Dialogue Manager may deal with feedback strategies, and consequently the Natural Language Generator module may generate messages expressing this feedback. Feedback is related to implicit and explicit confirmation of values, and also with different types of mistakes, either caused by the user or by the system. For example, different messages should be generated for expressing misunderstanding of the user's sentence, an error accessing to the database, or a bad parameter value.

1.3 Overview of this Document

The rest of this thesis project is organized as follows: Chapter 2 gives a review of *the state of the art* of Dialogue Systems and the development of Dialogue Management and Natural Language Generation tasks. Chapter 3 presents our approach to Dialogue Management and the state of the work that has been carried on so far. Our experiments and their results are contained in Chapter 4. Chapter 5 presents the work plan for our oncoming research. Finally, Chapter 6 lists the publications that our work has produced so far, and locates them in the context of our project.

Chapter 2

State of the Art

This Chapter presents a brief summary of most relevant Dialogue Systems. In particular, it is focused on the Architecture and Functionalities of most relevant Dialogue Systems (Section 2.1), as well as most advanced works on Dialogue Management (Section 2.2) and the Generation of System's Messages (Section 2.3). A sketch of the Evaluation in Dialogue Systems is also presented (Section 2.4).

2.1 Architectures and Functionalities of the Dialogue Systems

One of the earlier works focused on improving *all* the required behaviours for having an efficient Dialogue System (DS) is that of Smith et al. [1995]. Most of the DSs developed before had been focused on improving several of these behaviours, but rarely all together. The required behaviours described in Smith et al. [1995] are the following:

1. The system may provide the information that the user requested. This stands for achieving the user's goal.
2. The system may manage the tasks that arise during conversation. The dialogue participants may initiate little tasks that help to solve the main goal. These tasks can be seen as subdialogues, and supporting them depends mainly on the dialogue structure.
3. The information appearing during communication may be structurally stored. User and system interchange information during communication, and this information is the data that guides the conversation as well as the common knowledge for both participants. The process of managing the dialogue depends mainly on the representation structure of the conversation information, including the application representation.
4. Different degree of initiative for different users is a worth. While unexperienced user may need that the system guides the dialogue (system-initiative), the experienced ones may prefer to have the initiative and guide the dialogue to the goal by the shortest way (user-initiative). Most of the research systems allows mixed-initiative, although most of the commercial systems use only system-initiative strategies. Few of them include mechanisms for adapting the initiative strategy to the user.
5. The system may predict the user's answer. Speech Recognizing (SR) is a hard task due to the current status of the speech technologies. If dialogue information is structurally stored, then in each state of the dialogue the system would know the focus of the conversation. The dialogue focus may direct the mechanisms to recognize and process the user utterance. But, as more directed is the recognition of the user's intervention, less degree of freedom has the user in his intervention. In consequence, the user loses initiative.

The implementation of DSs implies the collaboration of a variety of technologies. Figure 2.1 shows the generic diagram for conversational systems ([Zue, 1997]). There are other type of conversational systems that are not exactly DSs, as for example personalized recommendation systems ([Thompson et al., 2004]), collaboration agents ([Rich et al., 2001]). In all the conversational systems the Dialogue Manager (DM) may depend on the discourse context representation as well as the Natural Language Understanding (NLU) module performance and structure of results. In the same way, the Natural Language Generation (NLG) process depends on the context representation and the strategies and output of the DM. The whole system behaviour depends on the degree of collaboration between modules.

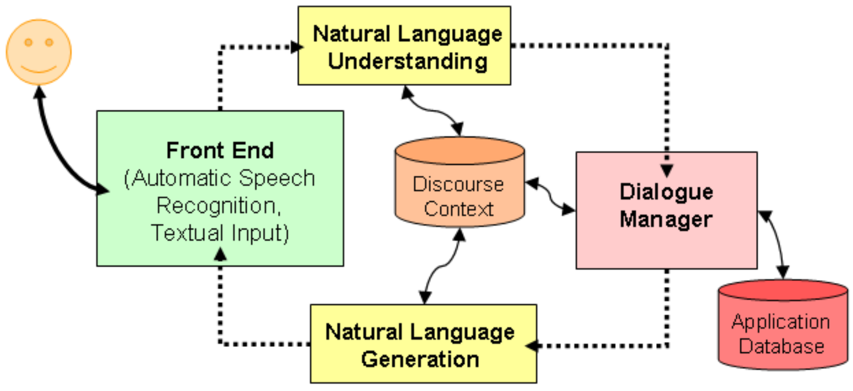

Natural Language Understanding
Front End (Automatic Speech Recognition, Textual Input)
Discourse Context
Dialogue Manager
Application Database
Natural Language Generation

Figure 2.1: Generic Diagram for Dialogue Systems

The DSs have evolved toward improvements in both the functionality and the engineering features of the development process. Several lines have been followed in this evolution. The most significant of these are the separation of the application and communication information modules and the adaptation of the communication process to applications and users. DSs adapted to the communication needs of the application improves performance because it reduces mistakes and ambiguities. However, application-restricted DSs are expensive to developed and difficult to reuse.

The problem of developing multilingual DS for different applications has focused many research works. Most of these works propose a separated representation of the different types of knowledge involved in communication (application knowledge, dialogue knowledge and modality and language-specific knowledge). In this line, one of most relevant work is that done by the Voice Browser and Multimodal Interaction ([W3C, b,a]) activity groups defining standard languages and architectures that facilitate the development of DSs for accessing the web contents. The most well-known of these languages is VoiceXML ([W3C, 2006]), the XML-based open standard explicitly dedicated to the development of Internet-powered telephone applications. There are many commercial systems implemented using VoiceXML-based platforms, and also research works, such as the aTTemps system developed at TALP ([Villarejo et al., 2003b]).

The development of large-scale DSs supporting rich interactions to different applications in several languages and channels still constitutes a challenge. For this reason, there is a lot of interest in research prototypes combining practical results from several research areas. These complex systems require a flexible and modular architecture model and adequate software for the integration modules. A distributed system constitutes the natural choice to develop a flexible and scalable software architecture, capable to integrate heterogeneous software modules.

A step can be taken toward the transportability of DSs and the reuse of their constituents by isolating and representing the conceptual knowledge involved in a specific domain or application in a declarative form. The representation of this information in an explicit and formal organization improves clarity, consistency and accessibility. The application knowledge model is used as the base for defining the semantics and the content of information exchanged by multiple components of the system. Increasingly, DSs incorporate ontologies to represent the application conceptual knowledge. The main advantages of organizing conceptual knowledge according to an ontology is that ontologies holds meta-level information about different types of entities appearing in a domain. Thus, the use of ontologies improves the extendibility and reusability. Several uses of ontologies in the DSs are presented at the end of this Chapter.

This Section revise several of the most relevant DSs developed in the last years, focusing on the description of the behaviours discussed above. First, the system developed by Smith et al. [1995] is presented. Then, there is a revision of the Galaxy architecture developed at MIT and the applications implemented there. Next, the systems developed at Rochester University and the application to give information about trains' timetable are described. The trains' timetable domain has been largely used in Spoken DSs, as the RailTel system developed at LIMSI, and the stochastic system of the DIHANA project.

- **The Zmod System**

The work carried on by Smith et al. [1995] is a DS for giving electrical aid to a worker. The aid and directions are given while the person is working, it consequently the worker has the hands occupied during communication. The Zmod architecture consists of specific modules for the domain processor and the knowledge information. The domain processor is organized as a part-of structure of subdialogues. It is an application-dependent process that returns suggestions for next goal to be undertaken. The knowledge base is a repository of task-oriented dialogues. It consists of the decomposition of actions into substeps, the theorems for proving goals, and the knowledge about expectations for the responses related to the actions. Moreover, there is a reasoner module specialized in theorem proven and problem solving. Given a set of goals to be proven, the general reasoner returns the proved theorems. Thus, the role of the DM is to perform the top-level goals formulation.

The reasoner module achieve the goals by theorem axioms prove, using prolog-based language (behaviour 1 of the list above). The role of this language is to supply missing axioms, while the role of the user is to enable the theorem proving process. The user capabilities are also specified as prolog rules (behaviour 3). In Zmod system, and similar ones, the user may not answer with the desired information, but with a clarification question, depending on the user skills and expertise. In Zmod the user model is represented using prolog axioms. The user model changes during conversation as the clarification subdialogues occur. In this situations, generally, the DSs can ignore the user or manage subdialogue tasks. Zmod system shift the control to another set of axioms (subdialogue) until the new goal of the subtask is achieved (behaviour 2). Then, the dialogue control returns to the previous goal. The system considers four degrees of initiative depending on the user model (behaviour 4). Thus, the system adapts the output statements to the initiative level. This levels are directive mode for most unexperienced users, suggestive mode, declarative mode and the passive mode, where the users have the complete control of the dialogue.
In order to improve the SR and NLU process (behaviour 5), the Zmod system produces a hierarchy of the expected user's intervention. The system expects that the user answer may be related to the current active subdialogues, or else that it may be related to a nearby active subdialogue. If nothing can be recognized, then the system supposes that the user answer may be related to a remote subdialogue.
Smith et al. [1995] work includes the description of the test performed and the evaluation analysis. The tests consisted in three sessions with each user. In the first session the purpose was to obtain general statistics on system performance and timing. In the second session the effects of the level of initiative adaptation were studied. In these tests the subjects attempted up to ten problems with the DS locked in directive or declarative mode. Finally, in the third session the focus of the study was to judge the human factors issues. In these sessions the subjects attempted up to ten

problems with more flexible than directive or declarative mode active. At the end of the tests three performance criteria were examined: the completion time, the number of utterances spoken, and the percentage of nontrivial utterances.

- **The Galaxy-II Architecture**

The Galaxy-II architecture ([Seneff et al., 1998, 1999]) consists of a hub of servers. The servers communicate each other using a token-based protocol. The information is represented using semantic frames, and the implementation is based on sequential application of rules. Each program in a server consists of a set of rules. Each rule is composed of a precondition, an input, an output and an operation.
The Galaxy-II architecture supports DSs with a variety of characteristics: multimodality, multichannelly, different domains and languages (even mandarin and japanese). This architecture is one of the most used systems for implementing practical dialogue applications, and also for research purposes.

The [SLS] group has developed all the domain-independent modules needed for having an application running. SUMMIT is an Automatic Speech Recognition (ASR) that generates a ranked list of candidate sentences, TINA is a rule-based parser for converting sentences into a semantic frame, the D-Server is a frame-based DM, and GENESIS and ENVOICE are the NLG and speech synthesizer respectively. Moreover, the Galaxy-II architecture uses two separate servers for accessing the application backends and to track the discourse context.

The SLS group has implemented several applications on different domains using the Galaxy-II architecture. VOYAGER (1989) is a city guide system that gives tourist and travel information about the Greater Boston area. PEGASUS (1995) and MERCURY (1999) are airline flight systems. The former gives information about flight departure, gate and arrival for major cities in the U.S through telephone, and the latter is a flight browsing and booking. The JUPITER (1997) weather information system is one of the most famous applications since it gives weather forecast for cities worldwide through a toll free telephone number in North America. ORION (2000) is a personal agent that performs previously enrolled tasks, and the latest system developed is PENATES, a restaurant information system that has been used for demonstrating rapid prototyping strategies.

Regarding to the desirable behaviours described at the beginning of this Section, the Galaxy-II architecture is specially designed for achieving the user's goal (behaviour 1). There is a specific control strategy for each different domain, named the turn manager, and there is one server for the database. In general, the result of processing the user answer is a list of N-best frames that are evaluated in parallel. The last list of items displayed to the user are stored in the discourse context (turn manager history), and the parser can retrieve them in the next turn. The discourse history is updated in each turn based on the specification of the executed rules. The turn manager selects the best hypothesis (behaviour 2) and performs multiple and parallel requests to the database in order to get the pieces of information necessary to solve the queries. Then, the turn manager prepares its reply to the user. The information related to the conversation is represented using semantic frames, including the representation of the system reply. The semantic frame representing the system reply is also used for updating the discourse context. In special cases, the system can take the initiative in order to request for the missing pieces of informations from the user. In this sense, there are not strategies for adapting the initiative to the user expertise (behaviour 4). Dialogues are basically mixed-initiative except when the system uses a special system-initiative dialogue to aid in the next user's turn interpretation (behaviour 5).

Within the Galaxy-II platform, a separate evaluation server has been developed. This server performs comparisons and accumulative performance statistics. The idea is writing scripts for the hub for running multiple versions of the same server and compare the new versions against the old ones.

- **The Rochester Architecture**

The Conversational Interaction and Spoken Dialogue Research Group at Rochester ([URCS]) has developed four DSs since 1982. The first one was ARGOT, focused on examining and automatically understanding the intentions resulting from user's utterances. URCS developed the TRAINS system, that allows routing and scheduling of trains. It is one of the most relevant systems developed in last years. The first TRAINS version was in 1991, and newer versions were developed in the following years as improvements were applied. It has been possible to compile corpus in train routing and scheduling domain, and thus it has been possible to develop stochastic DSs based on this domain, one of the most used in the research of DSs. TRIPS (1998) is an improved version of TRAINS. An intelligent planning assistant was developed for TRIPS, and it has been used for implementing more applications in different domains, as airlift scheduling and underwater survey. CHESTER (2006), the last system developed by this group, consists in a personal medication advisor. This is not exactly a DS, but an intelligent assistant that uses natural spoken language to provide the users with information and advice regarding their prescribed medications ([Allen et al., 2006]).

The specific application, TRIPS, described in Allen et al. [2001] is a collaborative multimodal system for a rescue scenario emergency. TRIPS system architecture is agent-based, where a broker is used for accessing the databases. The architecture is separated into three areas of functionality (agents): interpretation, generation and behaviour. Each area includes a control and management module that coordinates the behaviour and shares information with the other managers. The shared information between the three agents consists of three modules: the discourse context, the reference manager and the task manager.

TRIPS is based on a generic system that uses plan-based and agent-based models for abstract problem-solving tasks. This generic system is specialized to the particular domain by integrating domain-specific information (behaviour 1). This information consist of the set of objectives for the emergency, represented by goals and subgoals, the possible solutions and their courses, the available and occupied resources and the state of the situation at the rescue scenario. The behaviour agent must decide how to handle the proposed acts of establishing a goal. Given the ongoing discourse context, the behavioural agent considers its own private goals and obligations. Subgoals (behaviour 2) are detected using intention recognition strategies in order to know what question is truly being asked. Intention recognition process is a two-stage process. First, a set of rules matches against the user's input and generates possible intentions. Then, this intentions are evaluated with respect to the on going problem solving state. Since this is a collaborative conversational system, neither the system nor the user has the control of the whole interaction. Initiative is shared (behaviour 4), and each participant contributes when they can better achieve the goals of the interaction.

- **The LIMSI RailTel System**

Other relevant system in the train timetable domain is the RailTel, for which has been implemented prototypes at LIMSI (France, [LIMSI]), CSELT (Italy) and CCIR (U.K). The description below is focused on the LIMSI system ([Lamel et al., 1997]), that provides telephonic access in French to static timetable information, and some information about services offered on the trains, fare-related restrictions and supplements.

The RailTel architecture consists of six modules: the ASR, the NLU, the DM, the information retrieval, the NLG and the speech synthesizer. Two particularities of this system is that the

NLG is based on grammars, where the non-terminals are conditioned by the discourse context, and that the system never gives a negative response to the user, but relaxes the constraints provided in order to propose a solution.

The structure of the dialogue (behaviour 2) in RailTel consist of a hierarchy of subdialogues with a particular functional task to be accomplished. Formally, the dialogue is modeled (behaviour 1) combining formal grammars and speech acts theory. The result is a set of rules where the non-terminals of the grammar correspond to subdialogues, and the terminals correspond to dialogue acts (DAs). Each rule in the dialogue generates a DA which controls the opening, closing and message generation of the subdialogue. The information from the participants (behaviour 3) is represented using semantic frames. The DM maintains a discourse context history of those frames in order to avoid repeating itself. The ASR returns a list of N-best utterances. All the utterances are analyzed by the NLU module, and the context understanding process (behaviour 5) interprets the utterances in the context of the ongoing dialogue.

The three RailTel prototype systems were evaluated within the same framework. A global evaluation consisted in countering the number of calls, the turns per call, the duration and the success rate. Then the SR and understanding performance were evaluated, as well as the DM. Finally, usability profiles and completion task rate were studied as a function of scenario types, and overall results were presented as a function of the qualitative measurement of the user's satisfaction.

- **The Adaptive Place Advisor System**

The Adaptive Place Advisor ([Thompson et al., 2004]) is not properly a DS for seeking information, but a personalized one giving recommendations to select a restaurant that meets user's preferences. Adaptive Place Advisor uses a novel model of the information related to the users. It stores information acquired for beeing used in future conversations with the same user.

Adaptive Place Advisor architecture is similar to the typical DSs. It is composed of five modules: the NLG, the ASR, the DM, the retrieval engine and the particular user modeling system. There is no a goal to achieve (behaviour 1), but a recommendation to give. The solutions provided are based on a constraint-satisfaction solving problem, implemented as a case-based reasoning system. The conversational interaction consists of a sequence of questions by both participants, the system and the user. The user interventions answering system's questions are not used for ranking items, but for eliminating alternatives. The system considers two phases in the conversation: constraint-satisfaction and item presentation. During constraint-satisfaction, the dialogue state determines the system's utterances (behaviour 5) and the range of responses expected at each point. During the item presentations, the user mainly can accept or reject the proposition.
The DM follows a frame-based approach (behaviour 2) where the user fills in attributes related or in addition to the ones suggested by the system. The DM updates the query after each user interaction, and the search phase is ended only when a small number of items match the constraints and are highly similar to the user's preferences.
The user model (behaviour 3) in the system uses probabilistic, short/long-term and individualized structures that contain information about user's preferences for items and their characteristics. The user model is acquired by inferring user preferences examining online behaviour. Thus, there is not an explicit strategy for adaptable initiative (behaviour 4), but the dialogues should become more efficient over time, as the system knows better the user and viceversa.

Thompson et al. [2004] work also describes the evaluation of the Adaptive Place Advisor system. The evaluation consisted in two groups of people doing 15 trials each person. Moreover, trials of the same user were separated by some hours when possible. The variables considered in the evaluation were the following: the average number of interactions to find a restaurant accepted by the user, the time taken for each conversation, the number of system rejections, the speech misrecognition errors, the rejection and hit rate, and a subjective questionnaire provided to users.

- **The COLLAGEN System**

COLLAGEN is a multimodal (speech, text and menus) and multitask collaborative interface agent for conventional graphical user interfaces ([Rich et al., 2001]). Its main interest is that it is based on the theories of discourse ([Grosz and Sidner, 1986, Grosz and Kraus, 1996]). It uses Sidner's artificial discourse language for representing discourse interpretation and data structure. COLLAGEN collaborative agent has been added to several implemented applications already using conventional GUIs, such as: air travel planning, email access, video recorder program, symbol editor graphical interface, student teacher and thermostat manipulation.

The COLLAGEN collaborative system architecture consists of two main modules: the discourse interpretation that updates the discourse state, and the discourse generation that outputs an agenda. The state of the discourse is stored using plan tree structures. It contains a segmented interaction history (behaviour 3) that represents the past, the present and the future. The structure of the dialogue (behaviour 2) is represented in a stack of goals. A plan tree for each goal is stored in the stack. Each plan tree encodes a partial shared plan between the user and the agent. There is a plan recognition process that allows unexpected focus shift to other plans, which are push on the stack of goals. Thus, the interface is improved by facilitating intelligent help features, clarification and answer prediction.

2.1.1 Dialogue Systems and VoiceXML

'The VoiceXML language is used to specify what the system will say, what the user is allowed to say, and how to proceed from one dialogue state to another'. [Wilcock and Jokinen, 2003]

As noted at the beginning of this Section, the VoiceXML standard has brought a radical change in spoken commercial applications. Representing separately the application, dialogue, modality and language-specific knowledge, makes speech applications easy to implement. Moreover, using standard markup languages facilitates reusability and technology integration, and improves flexibility. One of most interesting characteristics of VoiceXML is that it allows creating a voice application separating the content and business logic from the presentation layer. Thus, developing a voice application becomes easy, the programmer can write it without having to know anything about the voice hardware on which the application runs. Standard markup languages to support multimodality have also been defined, such as the Extensible Multimodal Annotation (EMMA) markup language and the Synchronized Multimedia Integration Language (SMIL), for multimedia presentations which integrate streaming audio with video images.

However, the capabilities of VoiceXML systems are often limited. Although most of the efficient problems of VoiceXML systems are originated by the limitations in ASR engines, some of them are also caused by a poor dialogue design. VoiceXML has been proven efficient in certain type of applications. In these applications the dialogues are well defined, they can be represented using graphs and the system usually must guide the user, user-initiative is not required. If the whole Dialogue is well designed to best complete the tasks, and error recovery and help strategies are efficient, the system gets a high degree of success. The meteorological information system aTTemps ([Villarejo et al., 2003b]) is a good demonstration of those characteristics.

Different groups are carrying on investigations on the opportunities for the integration of intelligent technologies with VoiceXML. In Winiwarter [2002] several improvements by applying natural language techniques in VoiceXML applications are suggested, being one of them the generation of system's messages and prompts.

There are several DSs combining natural language techniques and VoiceXML, such as the GEMINI system. In the GEMINI project ([Hamerich et al., 2003, 2004, L.F. dHaro, 2004] research has focused on developing VoiceXML compatible DS application. The aim of this particular project was to generate the dialogue applications semiautomatically using a database description.

- **The aTTemps System**

aTTemps is a weather real-time information system for Catalunya areas, and it sends warning messages for a given data. Its architecture is based on the VoiceXML framework ([Villarejo et al., 2003b, 2006]). The platform consists of a web server, a DM, a NLG module and the databases. The ASR is specific for the telephony hardware. The aTTemps system guides the user through the dialogue, but the dialogue flow has been designed to allow naturalness in the communication process.

The user's utterances are processed by an ABNF parser for grammatical constructions that performs a transcription to an internal format. Since each page represents a specific state of the dialogue, the scope of the turn is defined in such a way that there is no possible ambiguity processing the user's utterance. The module performing the data acquisition uses two different databases, one for meteorological information and other for storing users' data. Finally, the NLG module uses templates that are dynamically filled.

The web server contains VoiceXML documents and application logic. The DS follows a state-based approach where the dialogues for the two services are a set of VoiceXML documents stored in the web server. Each page represents a different state (behaviour 2) and loads the grammar that must be used for recognizing the user's intervention. The VoiceXML interpreter determines which grammars must be active in each interaction. In terms of VoiceXML mixed-initiative (behaviour 5), the user can give more information than the required when the grammars recognizing such information are active. In the aTTemps system all the grammars are active allowing mixed-initiative except in that states where the grammar is too complex, i.e. the grammar for recognizing the municipality name.

aTTemps dialogue management strategies are better described in Villarejo et al. [2003a]. One of them is to give the user feedback of the information interpreted (behaviour 4). Most of the information is confirmed implicitly, but the critical information, as telephone number for message warning, is confirmed explicitly. After either the implicit or the explicit confirmation, the user can correct the last intervention. Moreover, error recovering, correction and helping features are strategies implemented in the aTTemps system. Help messages are only provided when the user explicitly asks for help. At the beginning of the conversation the system explains the user how to get help. Moreover, the VoiceXML standard establishes the mechanisms for detecting when the user is experiencing problems: noinput, nomatch and count. This mechanisms are used in order to provide the user with adaptable help messages according to the specific troubles.

2.1.2 Dialogue Systems and Ontologies

There are DSs dealing with Natural Language techniques that have already adopt some type of ontologies for giving more abstract level of the information involved. The Bateman [1991] work differentiates three types of ontologies depending on the approach used for representing the world knowledge: conceptual, mixed and interface ontologies. Some Natural Language Processing systems use conceptual ontologies to organize the concepts in the domain. Mixed-ontologies are abstract semantic-conceptual representations of the world knowledge used as semantics for grammars and lexicons. In such approach the lexical entries directly contain categories of the ontology,

which are categories of the real-world knowledge. Interface ontologies are abstract organizations that act as interfaces between world knowledge and grammar and lexicon. Their purpose is to establish a linguistically motivated organization of objects and relations in which application specification could be represented.
This Section revise several DSs using ontologies for improving the communication with the user, such as for representing the domain structure or the user model.

- **The SmartKom System**

SmartKom is a multimodal and collaborative system, and has been integrated into 11 different application. An ontology represents the domain and application knowledge. SmartKom system uses an homogeneous world model common for various modules. This model provides not only a top level ontology with a structured view of the world, but also a domain dependent knowledge ([Wahlster, 2003, Porzel et al., 2003]).

In the SmartKom system all the interface recognizers (voice, text, graphical, emotion) produce time-stamped hypotheses. There is no user's actions prediction based on the discourse context (behaviour 5), but it is applied a modality fusion process that consists in the reduction of the overall uncertainty and the mutual disambiguation of the various analysis results for each modality channel. Then, the intention recognizer module ranks the set of interpretation hypotheses and selects the most likely one to the ongoing context. Finally, the hypothesis selected is passed to the action planner module.
The DM uses plans of actions describing the user goals (behaviour 1). The plan models the actions that may be performed to access each service supporting the system. Actions could be either communicating with the user or with the application modules. Since services are organized in the ontology, any application module can use the rest of the services. The operations in the processes and roles of the participants are also encoded in the ontology.
Dialogues are structured in 43 types of complex tasks (behaviour 2). The discourse context (behaviour 3) uses a three-tiered representation: domain layer, discourse layer and modality layer. The items in each layer can be linked with items in the other ones. Moreover, the domain layer links discourse objects with instances of the ontology domain model.
System outputs are generated by a presentation planner module. The input to the presentation planner is a presentation goal encoded in M3L as a modality-free representation of the system's intended communicative act. This M3L structure is generated by either the action planner or the dynamic help component, which can initiate clarification subdialogues (behaviour 4). In this sense communication is basically mixed-initiative since the user delegates a task to a virtual communication assistant. In certain situations the system can direct the communication in order to obtain clarification's answers. The M3L is a XML-based language for describing interfaces data. The presentation planner specifies presentation goals for the test generator, graphics generator and animation generator.

- **The GISE System**

The GISE system was designed to provide Natural Language access to expert systems although it could also be adapted to other types of applications. i.e. the system answers properly the requests involving the knowledge in a collection of domain-restricted web sites. GISE uses ontologies for modeling both the application domain and the expert system performance ([Gatius and Rodríguez, 1996, 2002]). The domain and system's tasks are represented using a task-based specification methodology in a declarative way. This methodology is composed of the model specification and the process specification.

In GISE, a Conceptual Ontology (CO) represents domain concepts and communication tasks, while a Linguistic Ontology (LO) represents general language knowledge. The CO is organized

in three independent taxonomies: domain concepts, attributes describing the concepts and operations on the domain concepts. Operations are classified as simple or complex. The latter provide inferential and reasoning capabilities to answer complex questions. The system uses wrappers for obtaining the information from heterogeneous sources in the web. The extracted data from the web pages is represented in the CO.
A parser processes the user input, generating a set of possible semantic interpretations. Each interpretation includes the operation and concept identifiers as well as the operation parameters expressed in the intervention. Then, the DM completes the semantic interpretation using the information in the CO and the dialogue history. The operation that is being performed, jointly with its parameters are the *focus of attention*. For disambiguating the semantic interpretations, the DM considers the definition of the operation, as well as the context, and the focus of attention of the previous interventions.
When all the parameters for an operation are obtained, the operation is executed over the instances of the CO. If no satisfactory answer is obtained, then the DM extracts the information accessing the application (on the web pages).

- **The Cancer Diseases Diagnosis System**

The work described in Milward and Beveridge [2003] uses a rich medical ontology about terminologies and semantic information. This ontology is also used for guiding the user towards the diagnosis. Since the dialogues supported are mixed-initiative, the ontology allows the DM to deal with the user's utterances that do not fit exactly with what was expected from the user, as hypernyms and hyponyms (is-a), or part-whole relationships (not-is-a).
When the user's answer is related to the expected answer with an is-a link, the DM may discover the relation between the expected answer and the actual one. In case the answer is an hypernym, the system answers with a clarification question. In case the answer is an hyponym, the dialogue must find a more general term that answers the actual question, but also it must store the actual answer for avoiding possible future questions. When the answer does not fit the actual question but it is related to the expected answer with a not-is-a relation, the DM may discover whether the provided term is more or less specific than the expected one according to the relation of the concepts in the ontology. Once the relation is established, the system behaviour is similar to that of the is-a relations.

The DM achieves more natural interactions clustering questions semantically following the user *focus of attention*. Thus, the questions that are related to the terms appeared in the previous answer are set immediately in the next system's intervention. This strategy is named *dialogue coherence*, and it is implemented in the conversational game in which the DM is based on.

The ontology drives a high-level dialogue specification that is used by the move engine in each intervention to generate the sequence of DAs that can be made by the participants given the discourse context. Since all the domain knowledge is isolated within the domain ontology, the DM is fully generic.

2.2 Dialogue Management

The dialogue is a particular kind of discourse. There are three main characteristics in dialogues: there are turns, participants have a common ground knowledge, and there appear new phenomenons not present in other types of discourse, as pauses, errors, confirmations and false beginnings. In the intention-based dialogue models, each intervention is called speech act, and is then related to Dialogue Acts (DA). Many works has addressed the classification of DAs. An important one was the developed by Allen ([Allen and Core, 1997]), which was the base for the next study carried on by Jurafsky ([Jurafsky et al., 1997a]). In this study Jurafsky gets 42 final different clusters for DAs. In Jurafsky et al. [1998b] work and study about a specific type of these DAs is included: the backchannels. Backchannels are an important set of the sentences that appear in a dialogue. Backchannels include continuers, assessments, yes-answers, agreements and incipient-speakership. Other most significant DAs classified by Jurafsky are Statements, Questions, Exit, Answers and Confirmations ([Jurafsky et al., 1998a]).

Dialogue management techniques deal with the use of dialogue in furthering the user's objectives. The dialogue management process is concerned mainly with conversational control and guidance: who can speak at each state of the communication and what they can speak about. It is also related to the type of interaction allowed, as well as the underlying task and domain. This Section revise the underlying theory about dialogue management and dialogue models, and the functionalities and implementation of several DM approaches.

In Walton [1995] we can found a study of several human-human dialogue types: quarrel, deliberation, negotiation, inquiry and information-seeking. Walton's work also analyzes a philosophical point of view of the concept of **commitment**. In Walton [1995] a *propositional commitment* depends upon a context of the dialogue, and each type of dialogue has different kinds of rules for the management of commitments. The structure of the dialogues proposed in Walton [1995] are viewed as dialogue games where two participants take turns making moves, and each participant has its own goals and strategies. The main goal of a type of dialogue is distinguished from the aims of the participants of a dialogue. In Walton's work the dialogue is considered a rule-based two-person game where one participant defends a proposition against the other participant. In these rule-based DSs, commitments should be incurred or retracted as a function of the following: the kind of move, the type of dialogue, the goal of the dialogue, the speakers role and the appropriate rules for the dialogue type.

The DMs developed for most existing DSs follow one of these two main approaches: structural dialogue state and plan-based. Structural dialogue state approaches, such as finite state models, are still used in many spoken systems. For example, in VoiceXML, the standard language widely adopted for providing telephone access to web content, follows a finite state model. In structural dialogue approaches each dialogue state represents the results of an action performed in a previous state. These approaches are efficient for system-initiative dialogues, but they are not appropriate for mixed-initiative ones.
On the other side, plan-based dialogue management theories support collaborative communication but are difficult to implement in practical DSs. Plan-based models of dialogue consider utterances as communication actions that are part of a plan. In these approaches general planners and plan recognizers are used to produce cooperative behaviour.
In order to combine the main advantages of the two approaches, efficiency and flexibility, new theories were defined, such as the **Information State Update theory** (ISU) of language modeling ([Traum et al., 1999]). The ISU theory provides the tools to formalize the type of information the dialogue participants consider and how this information is updated during the communication.

In the literature we can find hand-crafted dialogue strategies, as that of Larsson [2002] based on a set of rules managing the dialogue state, or mathematical models for controlling the dialogue, as Louloudis et al. [2005]. In the latter, it is stated that having the graph topology and the characteristics of each node, one can predict the system's behaviour, that is mainly the number of interactions required for negotiate an appropriate action, the task completion rate.

In Allen et al. [2001] dialogues are classified by the technique used and ordered by the task complexity. The most simple approaches are those that use finite-state scripts. Next in task complexity are frame-based systems, where users ask questions and the system performs simple clarifications. Sets of contexts are those systems where several topics are integrated into the same service, such as travel agencies that, at least, deal with hotels and transportation. The systems for more complex tasks are plan-based approaches and agent-based ones.

In Xu et al. [2002] work dialogue management is usually guided by different principles, strategies and objectives. This work proposes a classification of the most common dialogue models and dialogue management approaches.

This work states that the *Model of the Dialogue* could be classified in two categories:

- Pattern-based. This approach models regularities (patterns) in dialogues at the speech act level.
- Plan-based. This approach relates speech acts in utterances to Dialogue Acts (DAs) in the plans. In plan-based approaches the DAs classification and its link to the utterances becomes a strong problem to solve, since from this relation may depend on the dialogue management.

The speech acts recognition is performed at SR level. Those speech acts may be related to the DAs. A relevant research in this line is the statistical approach for modeling DAs in conversational speech described in Jurafsky et al. [1998b], Stolcke et al. [2000]. These works also present a framework for automatic classification of DAs. The corpus used was the task free Switchboard of human-human conversations. The initial tag set used was the DAMSL ([Allen and Core, 1997]), but obtaining a domain-independent tag set motivated several modifications. The results was the SWB-DAMSL tag set, that finally has 42 different multidimensional and mutually exclusive labels ([Jurafsky et al., 1997b]).

The Xu et al. [2002] work on Dialogue Management also classifies DSs considering how task model and dialogue model are used:

- Dialogue Model Implicit. In this models the dialogue flow must be defined for each service jointly with the task model.
  - Task Model Implicit. Example of this type of dialogue management is finite state transitions.
  - Task Model Explicit. This approaches are usually frame-based.
- Dialogue Model Explicit. This model considers domain and application independent discourse management strategies.
  - Task Model Implicit. This approach has no practical development.
  - Task Model Explicit. This approaches has the frame-based advantages plus more naturalness.

Dialogue modeling is the preparation, for each turn, of the system's side of the conversation ([Zue, 1997]).

The dialogue model in the standard language VoiceXML follows the finite state approach. The main advantage that the model used in VoiceXML presents, compared to previous commercial systems that use the finite state model, is that it facilitates the description of the slots representing the various kinds of information the user would be asked to fill. Using this description the system can deal with a certain type of mixed-initiative dialogues: the user can choose the order in which information about the slots asked is given. However, VoiceXML cannot deal with user-initiative dialogues. Besides, the dialogue flow must be defined for each new service.
In this line, many spoken systems use a structural dialogue state model, more flexible than the finite state model, based on frames (or templates) representing the task model. These frames contain slots describing the various kinds of information the user would be asked to fill. The frame-based model is similar to the finite-state model used in VoiceXML systems. In most of those systems there is no explicit dialogue model. The number of frames needed depends on the application tasks complexity. Frame-based models are appropriate when the set of actions the

system can do is reduced (being the number of frames limited), which is not the case in complex applications.
The main problem in structural state dialogues (such as finite and frame-based models) is that they only support very limited user-initiative because the dialogue model is implicit. For each application, a specific dialogue flow has to be defined in which there was an explicit description of the relation between states and actions.
A more flexible communication is provided by including explicit dialogue model as well as task model. Complex DSs that use explicit dialogue model and task model to support rich interactions to different applications in several languages and channels already exist. Most of those systems use plans to manage the communication.

2.2.1 The Dialogue Models used in relevant DSs

2.2.1.1 Examples of Implicit Dialogue Models

- **The GEMINNI Project**

The GEMINI project aims to provide a platform for developing of multimodal (graphics, speech and text) and multilingual DSs ([Hamerich et al., 2004, 2003]. For this purpose, GDialogXML, a XML language for describing dialogues, that allows the automatic generation of VoiceXML and xHTML, was defined.
The GEMINI platform is divided in 3 layers. The first one is the framework layer where the application model is defined. This layer creates XML documents used in next layer. The second layer is the retrieval layer, where GRM, that is modal and language independent, defines the general flow. Finally, the third layer defines the modal and language dependent part of dialogues, that combined with the GRM produces the dialogue model. L.F. dHaro [2004] describes the assistants implemented for provide the platform with the application resources. These assistants are mainly for:

- generating the description of the database of the service
- generating a state-based flow of the application
- designing the interaction of each possible intervention

GDialogXML is a language for describing dialogue application within the GEMINI platform. It is object-oriented and has a tree-structure. It is used to represent the models produced by the layers, and the concepts for modeling data, procedures, dialogue flow and the interactions with the user. The main concept categories are commands, values, procedure declarations, variables and procedure calls. The data used during dialogue interaction is represented as classes with attributes, which can have complex structures.
Dialogues are created linking the GRM data with the modality extensions. The most important elements of the dialogues are:

- *variables definitions.* Either the local variables, or the procedures parameters.
- *output behaviour.* The output is related to the way the system express its interaction. This element is modal-dependent.
- *input behaviour.* This element is related to how to obtain information from the user.
- *control flow.* Control flow is like a procedure body, it expresses what to do with the input.

Although the control flow depends on the modality, the output and input behaviour will be different for the same application and different modality.
The retrieval layer is the main contributor to the control flow element. The assistant of this layer provides dialogue templates to have more complex dialogues, and then the designer can create custom dialogues. Moreover, the assistant helps the user to define database access functions, their

parameters and the matching with the other variables, and how arguments between dialogues are passed. Dialogues can be classified into different types, and the designer can specify whether a dialogue is mixed-initiative of only allows system-initiative interactions.
At dialogue layer there are the modality generators, that using scripts generate the final XML documents that feed the frontends. The modality generators specifies the interactions behaviours, such as the presentation of list of results, or strategies for confirming values. For example, the assistants for this modules allow the designers to specify how to present a list of results depending on the list length: empty, one item, less than maximum and maximum exceeded.

- **The ISU Web Interaction Management**

Hartmann et al. [2007] work describes a *'task modeling language for augmenting web applications with a proactive user interface'* (PUI), i.e. adapting the interface to the user's needs. The prototype implemented displays online support information adapted to the user's needs and preferences, and it predicts the content for input fields based on knowledge inferred from discourse context. The PUI consists of three components: the Process Guidance, the Generation of Justifications, the Generation of Explanations. These components use additional knowledge about the web application stored in structures of task models.
The DM described in Hartmann et al. [2007] is specially focused on web interaction management. A task model is composed of the interaction elements of the application. In Hartmann et al. [2007] the task model is represented as a directed graph where nodes are states and activities. Main components of the task model are *task nodes*, *wrapping nodes* and *relations*. In the *task nodes* the states are represented as *task states*, and activities are represented as *activity*. Both can be linked to a common ontology which is used for generating explanations for the elements represented by *activity*. Wrapping nodes contains *contexts* and *user interface content*, and they are used for information that is not directly related to the work flow process. *Relations* contains *transitions* and *dependencies*, that can use conditions that helps the user through the work flow. The *state nodes* are linked via *transitions* to activities, that represents the elements of the state. Then, the *activity* nodes are linked again to a state after performing the activity.

2.2.1.2 Examples of Explicit Dialogue Models

- **The Information State Update**

'Information State is the information needed to distinguish one dialogue from another.' [Traum et al., 1999])

Information State (IS) is not the same as dialogue state. In structural dialogue state approaches the information is implicit to the state of the dialogue, and each state defines the allowed next states. IS represents not only the general dialogue information, but also the internal state of each participant, such beliefs (knowledge), desires (goals) and intentions (plans). One of the advantages of the ISU theory, compared to other dialogue management theories based on believes, intention and goals, is that IS proposes a clear formalization of the knowledge involved in the context of dialogue information contained in IS may be either static (knowledge bases) and dynamic.

- **The Trindi Project**

Between the large amount of implementations for the ISU approach, TRINDI uses stack and sets. The dialogue moves (DAs) are the actions that limit the type of updates allowed in the IS. The set of dialogue moves are limited by the language interpreter, and also by the ASR performance. These dialogue moves are used for updating IS through rules. Rules are a set of conditions and effects, and how to apply them is an issue of the control strategy.

Specially interesting is the dialogue model described in Traum et al. [1999]. One of the main distinction of the Traum et al. [1999] proposal and the IS implemented within TRINDI is that the information that is supposed to be known by the two participants is separated from the information that only the system knows. Information only known by the system is the private information, that contains the *PLAN*, the *AGENDA* and some *TEMPORAL* information that may be accepted or rejected. The *PLAN*, which is not static since it can be modified during conversation, is the path to get the user's goal. The *AGENDA* contains the short-term actions to be performed by the system, and they are related to the last actions done by the user and part of the *PLAN*. The shared part of the IS, it contains the Commitments (*COM*), the Question Under Discussion (*QUD*), and Latest Utterance (*LU*). *COM* are those propositions expressed by any of the participants, while QUD is more related to the questions or goals that the conversations address. Each of these elements of the IS is formally represented as a stack or set of actions or propositions.

- **The IBISs implementations of the ISU approach**

Larsson, who participated in the TRINDI project ([TRINDI]) described above, developed the IBIS issue-based systems. In Larsson [2002] work the dialogues are represented using plans, and the dialogue model follows the previously discussed TrindiKit system (Traum et al. [1999]), based on the Ginzsburg's Dialogue Gameboard. Both user and system interventions are represented using moves. Each move is composed of actions performed over the concepts and attributes in the IS. The DAs are greet, quit, ask, answer and feedback. The application plans are a sequence of actions. Actions in plans are of type findout, raise and response.
Rules used by the DM are classified into two main groups: update and selection rules. The interpretation of the user's utterance is used for updating the IS. All the rules are executed orderly and iteratively until no more rules can be applied. Among the update rules there are the ones that access to the databases, that update the IS storing the database results. The selection rules are executed also iteratively for selecting the next system actions. Next actions are selected from the Plan and the IS status depending on their content. The selected actions are used for generating the natural language message of the system either asking for more information or for giving the requested information. The system actions are also used for updating the IS again.

- **The Milward and Beveridge [2004] implementation of the ISU approach**

The dialogue model described in Milward and Beveridge [2004] combine the use of ontologies with the ISU theory. The system described aims to perform more natural dialogues by clustering questions dynamically according to the user's answer. This means reordering the system interventions depending on the last user utterance. The DM chooses the precise ordering according to the current state of the dialogue and its own domain knowledge, organized according to ontological relations, such as *is-a*. Thus, the informational structure is provided by a domain ontology. The ontological information can be used for clarification dialogues and to order questions to maximize coherence. The presence of ontological subsumption relation between two related tasks causes the dialogue to infer a rhetorical elaboration relation and overrides the default ordering.
The concept of *Focus of Attention* is described as the attentional state, or the path through a series of focus spaces or dialogue states. The focus space could be seen as a tree, and it is related to the dialogue coherence. The focus tree is a fine-grained structure of the dialogue history, and can be considered as the context of the dialogue. This tree together with the ontological relationships will allow the system to decide on the next moves. The focus of attention is used for generating prompts, interpreting utterances in context and restricting the possible hypotheses of the voice module.

2.2.1.3 Example of Agent-Based Dialogue Models

- **The Advice System**

Advice System ([García-Serrano et al., 2001]) is an interaction agent for a multimedia user interface. García-Serrano et al. [2001] describes the integration of advanced understanding and expression capabilities with the intelligent configuration of answers.
In Advice System the dialogue management task uses semantic representation and pragmatic information, and uses the Searle's set of speech acts ([Searle, 1969]). Interventions are divided into discourse pieces (DAs) that, similar to Larsson's moves, are composed of speech acts. The DM convert speech acts into steps, that corresponds to the feasible states of the dialogue. Each state defines the form of the system answers. States are defined from general point of view: greeting, require clarification, solve task, provide explanation. Specific content is stored in the dialogue model. After processing the information in the speech acts, an intelligent agent decides if more information is needed from the user, or all the information has already retrieved and the system is ready to perform the action requested by the user. When more information is needed from the user, a discourse maker uses a tree structure of the context for generating the question to the user.

2.2.1.4 Example of Stochastic Dialogue Models

- **The DIHANA System**

The DIHANA system ([Griol et al., 2006, 2005]) allows the user accessing to railway timetable and fares information in Spanish. In the context of this work three different DMs have been developed. Two of these DMs are rule-based and one is stochastic. The data for the stochastic-based DM methods was acquired with the Wizard of Oz technique using mixed-initiative communication. The DM uses stochastic models of the dialogue for determining the dialogue strategy. The user's answers from the data acquired is interpreted and represented in frames of DAs. The Speech Understanding module outputs one or more frames of concepts. The processing of such frames consist of updating the Stochastic Dialogue Model and the Dialogue History. The dialogue register contains attributes and confidence scores. The DAs are labeled into three levels: the general speech act, the concept related to the speech act, and the values given.
The structures of the dialogue models are represented by sequences of those DAs, and they has been estimated using only first and second level of labels.
Two models are used for training the system. In the first, the state identifiers contains information about the participant and the label of the first level. The second model contains only information about the labels of the second level. Each model is a bigram and, for each system state, the model establishes the possible transitions to user states, and viceversa.
In each turn, the system takes the interpreted user answer frame, and transits from its current system state to a user state. After updating the Dialogue History with the information in the frame, it transits from the user state to a system state using the information stored in the Dialogue History. The type of the state determines whether to access the databases or ask the user for more information.
Once the next system intervention has been determined, the answer's generation module performs a translation of the semantic representations of the system turns to sentences.

2.3 Generation of the System's Messages

Grice [1975]) established general conditions applying to that human conversations where participants *cooperate* and they are not trying to mislead each other. These conditions were named ***maxims***, and stood for the definition of the *COOPERATIVE PRINCIPLE* (CP), that was the base for the definition of the *CONVERSATIONAL IMPLICATURES*. The CP is defined through four features and their maxims, that are the following:

1. Quantity
   (a) Make your contribution as informative as is required.
   (b) Do not make your contribution more informative than is required.
2. Quality
   (a) Do not say what you believe to be false.
   (b) Do not say that for which you lack adequate evidence.
3. Relation
   (a) Be relevant
4. Manner
   (a) Avoid obscurity of expression
   (b) Avoid ambiguity
   (c) Be brief
   (d) Be orderly

These maxims have been behind the design of the most NLG systems.

Natural Language Generation (NLG) is the process of building a text in Natural Language for communication aims ([Bateman and Zoch, 2005]). We can consider it as the the opposite to Natural Language Processing. DSs are not the only systems that uses NLG. The NLG techniques can also enhance the functionality of VoiceXML by allowing the generation of the system messages automatically. In most VoiceXML applications, the system's messages or prompts are incorporated as canned text in the VoiceXML pages. The NLG techniques could be used to obtain the appropriate system's messages to be included in the VoiceXML documents.

Most of the DSs do not use an advanced NLG module for generating the system's answer. Instead, they use template-based techniques or hand-crafted messages because these strategies are easier to implement and integrate in the DS, and they do not influence a lot the DSs overall performance. Nevertheless, few systems include advanced architectures for the NLG task. Figure 2.2 shows an architecture for an advanced NLG. When applying this architecture to a DS, the input to the NLG is the representation of the answer of the system to an user request. If the DS uses telephony channel, the NLG output would feed a TTS module.

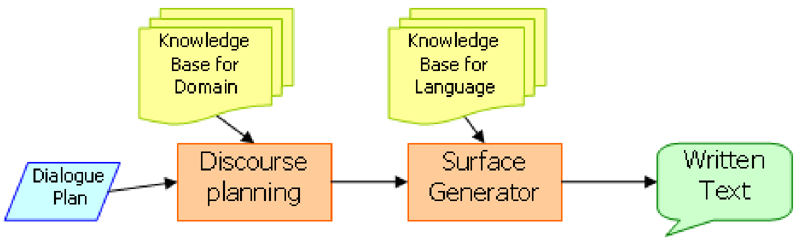


Figure 2.2: Natural Language Generation Architecture

The discourse planner is responsible for deciding ***what to say***. Most relevant works related to these tasks are the schema-based TEXT system ([McKeown, 1985]) and the Rhetorical Structure

Theory (RST, [Thompson, 1987]). The discourse planner module is strongly related to the system that uses it, i.e. the DM.
The surface generator is responsible for deciding ***how to say***. This module is related to linguistics. The inputs to the surface generator module are fragments of the discourse plan, and the outputs are complete sentences related to the DAs, Concepts, Attributes and Values that were planned to communicate. Bateman and Zock [2003] classify surfaces generators into three classes: systemic-functional grammars, tree-adjoining grammars, and meaning-text model.

- **The KPLM System**

One of the most relevant works is that of KPLM system described in Bateman. It is an example of the application of systemic grammars to the NLG task. In systemic grammars, sentences are represented as collections of functions and rules for relating functions and grammatical forms. In KPLM the functional systemic large-scale grammar NIGEL ([Mann and Matthiessen, 1985]) is used. KPLM uses conceptual ontology Upper-Model ([Bateman et al.]).
From the Upper Model, Bateman developed the Generalized Upper Model ([Bateman et al., 1994]), a generalization of the formed to different languages. In our DS, we have adapted the syntactic-semantic taxonomy described in Gatius [2001] for generating the system messages. This taxonomy is inspired on the Generalized Upper Model, that has been demonstrated to be reusable and shareable across different languages and domains, and was already applied to German, English, Italian by the authors. The Generalized Upper Model is an organization of the information motivated for supporting Natural Language Processing requirements.

- **RDF/XML + DAML/OIL**

One of the latest works done in NLG is the Wilcock and Jokinen [2003] system, based on **RDF/XML** and **DAML+OIL**. This work is focused on an appropriate presentation of the pieces of information that must be given to the user, toward a human-computer communication with the same characteristics of the human-like one: *'effective, friendly and enjoyable'*. In this work, the generator follows the strategy of shallow generation. Information is domain-specific and the solution is task-specific. In Carolis et al. [2007] the information is structured using RDF/XML. Thus, already developed standards are used, such as personal information vCard. Then, the generator uses XSLT and templates for generating responses. First, the discourse planner creates a text plan tree for the answer, and then the surface generator converts it into a text specification tree, creating a natural message.
Next steps carried on by Wilcock and Jokinen are moving such domain-specific and task-specific generator, to another domain, and in consequence another task. This idea is applied to the task of generating explanations from ontologies, where the new domain were the ontologies itself. Thus, it is demonstrated that the use of ontologies in this way that the use of a task and domain specific generator is applied to any domain and any task.
Finally, in Jokinen et al. it is also described the way that an ontology is used for improving DS responses in the Interact System.

- **The MyMap System**

Another interesting work also related to NLG is the **MyMap** system. MyMap generates personalized tourist description ([Carolis et al., 2007]). It is not exactly a DS, but it also models the user and the context information, and uses it for generating natural language responses.
In MyMap a graphical map made for a human tourist guide is annotated and used as the domain knowledge. This domain knowledge is used to generate the natural language explanations and suggestions.
In MyMap mobile devices are used for providing the user with the tourist information. In such a

system, the contextualized user preferences and interests become a key features. This information is stored in a user profile, named MUP. A content planner decides whether the content and structure of the system's intervention, and a context-sensitive surface generator renders the surface level according to the device in use.
The NLG architecture of MyMap allows the pipeline model of NLG described above. In the planner component the user-related information can be used for constraining the generator's decisions, while in the surface generator it is useful for improving the effectiveness of the generated text, i.e. by using words dependent on the context.
The approach used for the planner module is RST because this method is more suitable for planning text structure and assembling text dynamically. Due to real-time interactions naturalness and the kind of devices used in this system, MyMap approach uses a library of plan-recipes expressed in an XML-based language named DPML. DPLM library is used for specifying the structure of a discourse plan-based on RST method.
The content planner selects the plan that best fits the current situation, and instantiates it with the current context information. At the end of this operation, an XML file easily convertible to any other format using XSLT is obtained. The surface generator uses templates for generating the final descriptions.

2.4 Evaluation of Dialogue Systems

The evaluation of the DSs has been one of the major drawbacks of this area. It is easy to say if a transactional task has been performed correctly or not, but not to say if the access to an informational service has been satisfactory. In this type of services, different degrees of user expectations are achieved, even when talking to humans. Thus, evaluating the DS performance is a matter of different features and degrees of achievements, and depends on the specific task the system is facing. For this reason, most of the DSs developed have been evaluated based on the achievements of the user's expectations, instead of comparing the behaviour with other systems.

Danieli and Gerbino have dealt with improvements in speech communication by studying the error recovery strategies ([Danieli and Gerbino, 1995]). They proposed a set of metrics for evaluating the DSs. In particular, they proposed three different points of view for evaluating DSs:

- the ability for driving the user to his goal,
- the overall quality of the system,
- the robustness, related mainly to error recovery strategies

The first of the items mentioned above can be evaluated in terms of the achievement of the goal (yes/not), but the other two items are subjective ones. In Danieli and Gerbino [1995] it is proposed a set of metrics that try to overcome this subjectivity. These metrics are the following:

- *Implicit Recovery.* The ability of the DM to recover from partial or total failure of recognition or understanding level. It is measured as the percentage between the DM correct conceptual errors and conceptual errors.
- *Contextual appropriateness.* It is a qualitative measure based on the degree of coherence of the system's message. The coherence of an answer depends on the Grace's conversational maxims (Section 2.3).
- *Turn correction ratio.* It is the adding of both the ration of user corrective interventions and the system ones.
- *Transaction success.* It measures the successfulness of the system providing the user with the information requested.

- **The PARADISE framework**

The aim of **PARADISE** was to set a general framework for evaluating spoken DSs. It was developed in the AT&T Labs, and has been used by other systems. PARADISE framework provides a task representation that decouples *WHAT* the system needs for carrying on a task, from *HOW* the system carries on the task. This task representation allows to compare dialogue strategies by specifying the relative contribution of other factors to the overall system performance ([Walker et al., 1997]).
The task representation consists of a table of all the possible attribute-values pairs in a matrix (AVM). This matrix contains all the information the system and the user can exchange. A scenario consists in an instantiation of this matrix, selecting specific values for each attribute.
The evaluation of the success of the scenario uses the Kappa coefficient ([Carletta, 1996]) applied to confusion matrix of dialogues performed. The Kappa coefficient takes into account, in its measurement, the complexity of the task.
The evaluation of the dialogue costs is a function of different cost measurements. Cost measurements are the ones typically used for such systems, as number of utterances per turn, number of turns, number of repair turns and elapsed time. Each measurement can be applied to any complete dialogue or to part of it. Finally, each cost measurement contributes to calculate the overall system performance.
The final measurement of the performance is a function of Kappa coefficient and the different costs. First, each measurement as well as Kappa coefficient is normalized for overcoming values scale problems. Then, each measure is weighted. Weights are based on a degree of contribution to user satisfaction, which is calculated with a typical quantitative agreement of the user to some statements. The weight value is given by a multiple linear regression.

Litman and Pan ([**Litman and Pan, 2002**]) used PARADISE framework for evaluating the **TOOT system**. TOOT is an adaptive DS for retrieving online train schedules. It adapts the dialogue behaviour to the user particular context during conversation. It applies learned rules for modeling if the user is having SR problems. The initial system configuration supports user-initiative dialogues without confirmation, then the system moves to mixed-initiative with implicit confirmation, and finally the system guides the dialogue and uses explicit confirmation.
Their experiments demonstrate that the best learned rule to classify a dialogue as problematic, related to the ASR, only used the predicted percentage of misrecognized utterances.
The first evaluation consists in a two-way ANOVA analysis of these measurements: total number of turns per dialogue, percentage of timeouts, ASR rejections, percentage of misrecognitions, task success, and a user subjective evaluation of a questionnaire.
Then, PARADISE is used for deriving a performance function from the collected data. The user satisfaction is calculated as function of task success and cost measures: misrecognitions, turns, rejections and timeouts. The PARADISE analysis yields different set of performance predictors compared to the analysis of variance evaluations.

In **Hadjdinjak and Mihelic [2006]** some drawbacks regarding the way PARADISE framework builds the function of the user satisfaction prediction are highlight. For example, adding the cost measurements seems an arbitrary decision since some of the score measurements could correlate or could be redundant. They propose a new method for choosing the set of predictors. Moreover, user's satisfaction values need to be normalized for obtaining an acceptable error in the prediction.

- **The model for Tracking Initiative Shift**

An interesting work on evaluating DSs is that of Chu-Carrol and Brown [1998], where the theory of Dempster-Shafer on reasoning under uncertainty is used for evaluating the tracking of initiative on a collaborative DS. In this work the dialogue-initiative and the task-initiative are differentiated. Task-initiative is held by the participant that establishes the development of the plan, i.e. how the task should be accomplished. Dialogue-initiative is held by the participant that establishes the discourse focus, either determining the plan or initiating a subdialogue to current the current plan.

Chu-Carrol and Brown [1998] work argue that dialogue and task-initiative can shift between participants in collaborative interaction. For example, given a goal raised by a participant, three different types of answers are considered: a) the raised question is just answered and there is no dialogue neither task-initiative shift, b) information about the invalidity of the plan is given and the dialogue-initiative is shifted, and c) not only is given information about the invalidity of the plan but also an alternative solution is proposed, and both dialogue and task-initiative are shifted.

Knowing the reasons that motivate initiative shifts between dialogue participants can improve the behaviour of collaborative systems, and, in particular, user-initiative DSs. Chu-Carrol and Brown [1998] work describes the evaluation model developed for this purpose, based on Dempster-Shafer theory, for tracking initiative. One of the reasons for using this theory was that it uses *basic probability assignments* (bpa) to represent the effect of a *piece of evidence* over a set of possible outcomes. For evaluation was used the TRAINS91 corpus. It was manually labeled each turn with two labels (TIH and DIH) indicating the participant holding the task and dialogue-initiative, and the specific cues (of total 8) in the turn. Basic probabilities representing the effect of each cue to initiative shift were assigned. The system performance was evaluated based on its accuracy in predicting the task and dialogue-initiative holders for each dialogue turn.

For applying the model over the corpus, a task and dialogue-initiative indexes measuring the level of involvement in directing the planning process and in determining the discourse focus respectively was associated with each participant. At the end of each dialogue turn, new initiative indexes were computed based on the current indexes and the effects that the observed cues have on changing these values. When a cue is observed in a turn, this observation provides evidence to the value of the task-initiative index in the next turn for current participant. Each cue was associated to two bpas to represent its effect on changing the values of the current task and dialogue-initiative indexes, respectively. A training algorithm determined the bpa functions to represent the effect of each cue.

Chapter 3

Dialogue Management for Guiding the User to Access Web Content

In this Chapter we describe our work. It has been mainly done in the context of the [HOPS] project. One of the main goals of this project has been the improvement of existing commercial DSs by combining practical results from various research areas: Speech, Natural Language, Dialogue Management and Ontologies.
Most recent works on spoken DSs in the commercial area are based on exploiting the VoiceXML language. This standard defines the framework for the dialogue model and the management rules. VoiceXML favours the reusability and the modularization of the components involved in the communication, and thus facilitates the development of DSs. But, it also presents several limitations. It is focused on the use of telephone, thus it only supports voice and touch-tone DTMF input modes. Moreover, VoiceXML presents some dialogue management drawbacks, as the limited support to the user-initiative interactions.
In order to present solutions to the limitations of existing DSs, our work has been focused on the study of how DSs, and in particular VoiceXML platforms, could be improved by using language, dialogue and ontology techniques. For this purpose, in the HOPS project, a DS supporting text and voice in four languages has been developed. This work has been done in three main steps, corresponding to the three prototypes we developed. I participated in the architecture design, and I developed the modules responsible for generating the system's answers and the DM of the last prototype. I also participated in the development and the integration of the module responsible for accessing the databases, and in the integration and adaptation of a previously developed parser.

The system was developed to support two different types of services: *transactional* and *informational*. Current implementation supports access to three services. They are described in Section 3.1 for a better understanding of the rest of the Chapter. Future work of research includes the adaptation of the system to other applications and other type of services. Then, Section 3.2 describes the preliminary prototypes. This description presents the problems we try to solve in the final prototype, that is described in Section 3.3. The DM design and functionalities are described in depth in Section 3.4, and Section 3.5 is dedicated to the Natural Language Generation and the channel-dependent Output Generation.

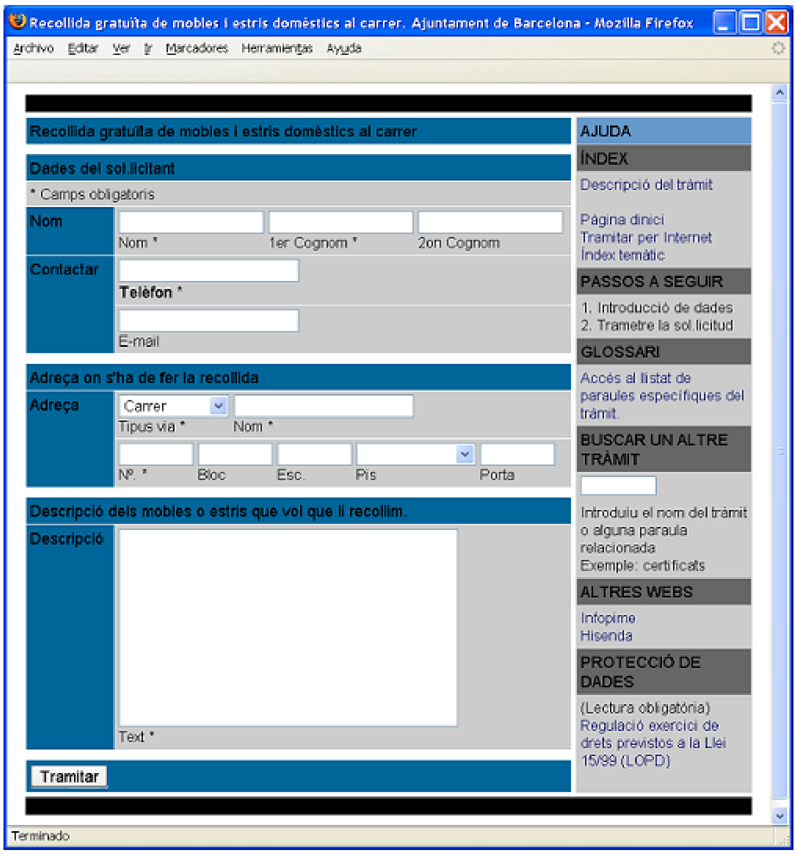
3.1 The Web Services

Although the DM component we have developed could be applied to different types of applications, we have adapted it to two different types of web services: informational (gives a particular piece of information) and transactional (executing a particular transaction).

The first service supported by our DS is the Large Objects Collection **(LOC)**, a transactional service where a citizen asks for a collection of large objects. The system registers the citizen collection request, that is different for *private* and *company* users. The companies must pay for the collection, and the price depens on the volume of the objects.
In case of private citizens, they are asked for the *address* where the objects are, the *objects* typology and the citizen *telephone*. Then, the service calculates the *date* when the city service will collect the objects. The citizen should answer if the date for the collection is *suitable* for him. Finally, the system *registers* the request and gives the registration *status* to the user, as well as a transaction *identification* that the user will need in case he wants to cancel the collection.
In case of companies, they are asked for the *address* where the objects would be collected, an approximate *volume* of them, and the *telephone*. Then the service calculates the *price* of the collection and the *date* the city service will take away the objects, and the user should answer if the date of the collection is *suitable* for him. Finally, the system *registers* the request and gives the registration *status* and the transaction *identification*.
Other task in the LOC service is the *cancellation* of a collection. The citizen can cancel a request providing the system with the transaction *identification*. Then, the system searches for the collection and removes it from the database.
Furthermore, LOC service gives *information* about the *Green Points* in the city. A green point is a place where you can leave pollutant or recyclable objects. This task gives the user the nearest *Green Point* to a given address.
The data needed from a particular user is represented in Figure 3.1 as a form to fill (the web page is obtained from the website of the city of Barcelona and it is in Catalan). This information corresponds to the input parameters which value has to be asked to the user: the user's name, the address, the telephone and the type of the object to be collected. In case of companies the volume of the object is also asked.

A similar service has also been supported in the final prototype: the Abandoned Objects Denounce service **(AOD)**. It is a transactional service very similar to the LOC one, but it only *registers* the type of *objects* and the *address* where the objects are abandoned in the street.

The other service supported by the system is the Cultural Agenda **(CA)**, an informational service giving information about the cultural events that will took place in the city. The user can perform searches by *title*, *location* or *date* of the events (*Query Focus*). Then, the system presents to the user a *list of events* that matches his query. In the particular case that only one event matches the query, the system presents the information related to the specific event. Moreover, the user can ask for a *specific data*, or for a *full description* of an event.



Recollida gratuïta de mobles i estris domèstics al carrer. Ajuntament de Barcelona - Mozilla Firefox

Archivo Editar Ver Ir Marcadores Herramientas Ayuda

Recollida gratuïta de mobles i estris domèstics al carrer

Dades del sol.licitant

* Camps obligatoris

Nom: Nom * | 1er Cognom * | 2on Cognom

Contactar: Telèfon * | E-mail

Adreça on s'ha de fer la recollida

Adreça: Carrer | Tipus via * | Nom *

Nº. * | Bloc | Esc. | Pis | Porta

Descripció dels mobles o estris que vol que li recollim.

Descripció: Text *

Tramitar

AJUDA

ÍNDEX

Descripció del tràmit

Pàgina dinici
Tramitar per Internet
Índex temàtic

PASSOS A SEGUIR

1. Introducció de dades
2. Trametre la sol.licitud

GLOSSARI

Accés al llistat de paraules específiques del tràmit.

BUSCAR UN ALTRE TRÀMIT

Introduïu el nom del tràmit o alguna paraula relacionada
Exemple: certificats

ALTRES WEBS

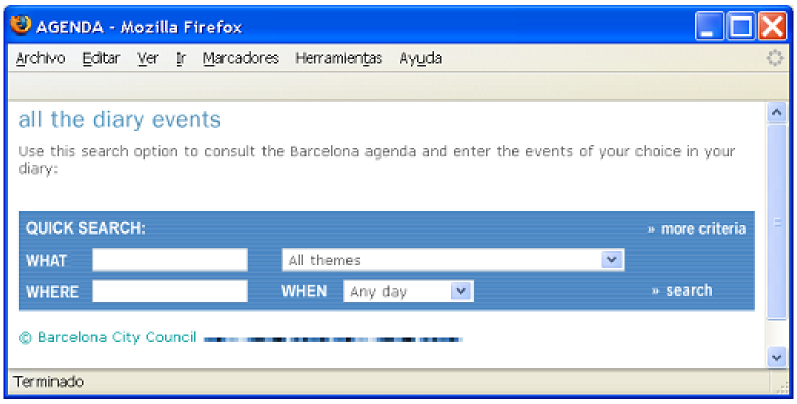
Infopime
Hisenda

PROTECCIÓ DE DADES

(Lectura obligatòria)
Regulació exercici de drets previstos a la Llei 15/99 (LOPD)

Terminado

Figure 3.1: The LOC Web Service



AGENDA - Mozilla Firefox

Archivo Editar Ver Ir Marcadores Herramientas Ayuda

all the diary events

Use this search option to consult the Barcelona agenda and enter the events of your choice in your diary:

QUICK SEARCH: » more criteria

WHAT | All themes

WHERE | WHEN Any day | » search

© Barcelona City Council

Terminado

Figure 3.2: The CA Web Service

3.2 Preliminary prototypes

This Section gives an overview of the two initial prototypes developed in HOPS, and a description of the problems we wanted to solve in the final prototype. The service supported by those prototypes were the LOC and the CA described in Section 3.1.

The DS has been developed for the HOPS project, that deals with the delivery of information and online public services in the local administration. The project is focused on developing a multichannel mass-scale e-government platform. The architecture of the platform has been designed to support interoperability between components distributed over LAN or WAN connections, mass-deployment of services for the citizens, multiple languages and operating systems and synchronous communication. We selected FADA [FADA] as the integration tool for this project because it provides flexibility, scalability and integration between heterogeneous systems and services and does not need administration.

The architecture of most exiting DSs consists of the following modules:

- User Input
- Natural Language Understanding
- Dialogue Management
- Database Access
- Natural Language Generation

One of main differences between the first prototype and the typical implementation of DS (Section 2.1) is that the dialogue flow was represented using ontologies. There was an ontology for each service, and each state in the conversation was represented in the ontology. Each state could contain the following information:

- The attributes the user must provide to the system
- The attributes the system must provide to the user (database access)
- The state precondition

The Figure 3.3 shows the first prototype architecture. In this architecture the DM is a finite state machine. It selects the next states by evaluating the preconditions of possible next states and selects the state with the most restrictive condition matched. The preconditions are expressed in terms of the attribute's values, which are the semantic interpretation of the user input.
Text and speech inputs are processed by different and separated components. The text input is processed by a syntactic-semantic analyzer described in Gatius [2001]. The voice input is processed by the Loquendo's platform VoxNauta. The Automatic Speech Recognizer (ASR) of this platform uses GRXML grammars [W3C, 2004] to process the user's utterance, and returns the semantic interpretation [W3C, 2007]. An example of grammar rule I wrote for the LOC service is shown in Figure 3.4. It is in ABNF format for better legibility.

The system has been designed to access three different databases in Torino, Barcelona and Candem (London). The database access is done through the Action and Query Manager module (AQM), that acts as an interface between the DM and the BackEnd (BE) databases. Moreover, in order to separate the database access from the specific implementation of the BE, a common BE API has been defined.

The BE access is done when the DM needs to execute an update or query in the database. The database updates perform a transactional database access, while the database queries realizes

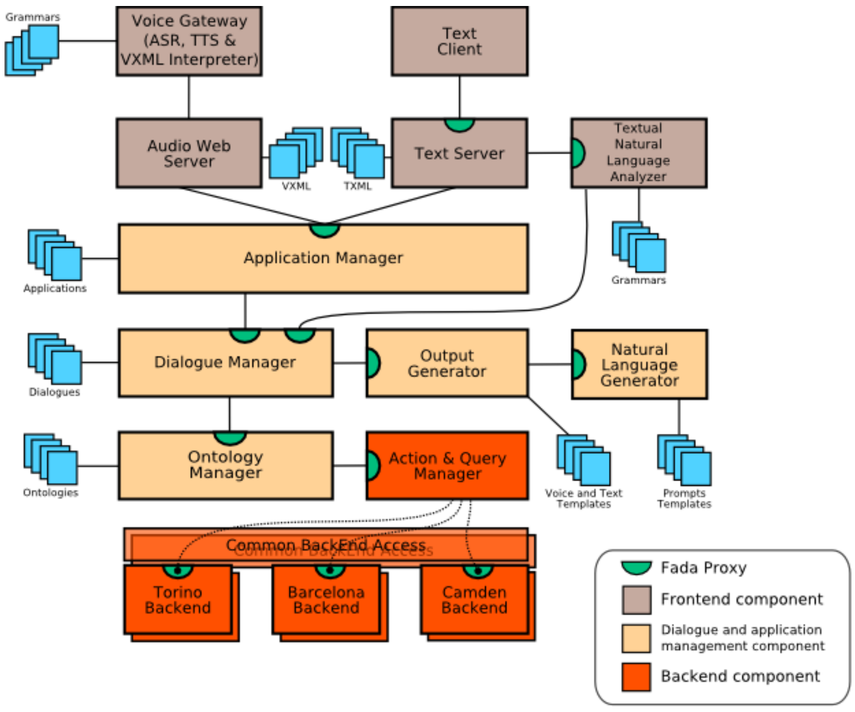


Figure 3.3: First Prototype Architecture

```
#ABNF 1.0;
language en-gb;
root $gramuserType;

public $gramuserType = $GARBAGE
                       ( ( $GramPersonal {:ret} ) | ( $GramCompany {:ret} ) )
                       $GARBAGE {<@userType $ret$>} ;

private $GramPersonal = ( ( ("private individual")) | ( ("individual")) | ( ("private")) | ( ("personal")) |
                        ( ("particular")) ) {return("private")} ;

private $GramCompany = ( ( ("company")) | ( ("firm")) | ( ("business")) | ( ("office")) | ( ("work")) |
                       ( ("organization")) | ( ("complex")) ) {return("company")} ;
```

Figure 3.4: ABNF Grammar for recognizing the user type

a selection of the information contained in the database.
The BE access is done in three steps. First step for accessing the database consists in generating a call to the AQM module. This module builds a generic BE call object that contains the specific data of the execution. This data is:

- The function that must be called
- Input parameters
- Output parameters

Input and output parameters must be linked to parameters in the discourse context of the DM.
In the second step, the AQM module performs the real call to the BE. This call returns an object that must be converted to the output parameters specified in the BE call object.
Finally, the information obtained from the databases is used for updating the discourse context of the DM, and the system shows the user the information that requested.

After the execution of a *transaction*, the system shows the transaction status, which could be:

- OK. The transaction has been done correctly.
- KO. The transaction cannot be done.
- ERROR. There has been an error while trying to perform the transaction.

The *queries* to access an informational service can return:

- No results
- One item
- A list of items
- Too much items
- System error

When there are many results, the BE module calculates how many results would be received if more data restrictions are included in the query. The AQM selects the most discriminative parameter, for which the user will obtain the most constrained data. For example, if the user asks for concerts during the city greater celebration, the system will return many results. Thus, in this supposed case, the BE will calculate how many results will be selected if the user gives a location, a date, or a kind of music. The AQM selects the parameter that returns less items. The information generated by the AQM is sent to the DM, and in the following interventions the system must be asking the user for the parameter selected.

Once the First Prototype was developed we studied its limitations and concluded that new improvements were necessary. System-initiative dialogue may result friendly and efficient for a transactional service, specially for novice users, who do not know the information the system needs. For users that have some experience with the DS, system-initiative dialogues do not result friendly because the users could give all information the service needs without the system's prompts. Furthermore, the system-initiative dialogues can result unnatural and inefficient for informational services.

Figure 3.5 shows the architecture of the second prototype. The aim of the second prototype was twofold: the improvement of the performance of the voice recognizer and the reduction of the cost of adapting the system to a new service. As a first step in this direction, in the second prototype we decided to integrate the components performing syntactic and semantic analysis of text input to the voice component. As a second step, we developed a more general DM, that was planned for being integrated in the final prototype.

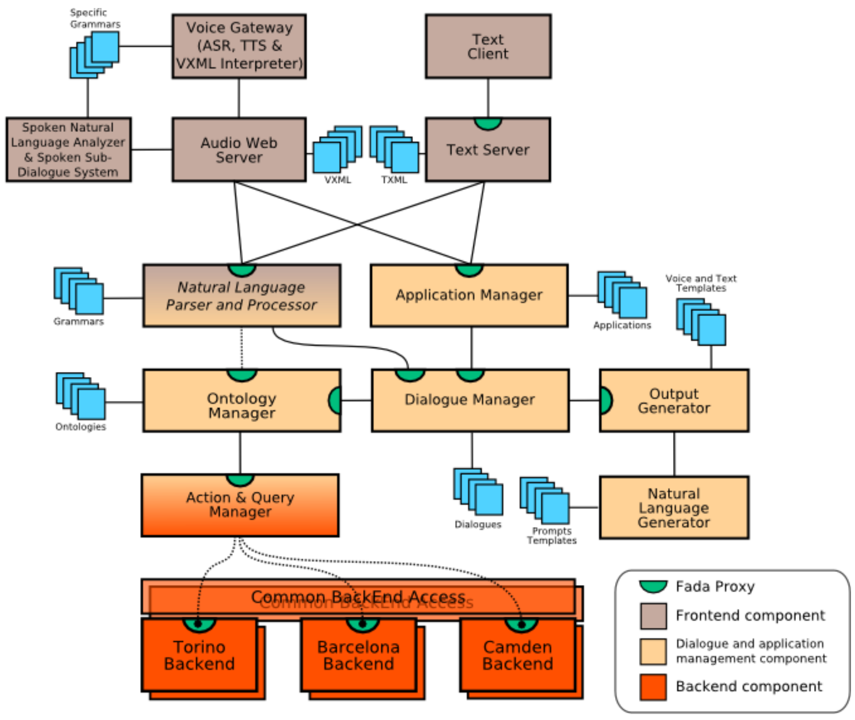


Figure 3.5: Second Prototype Architecture

3.3 Overview of the final prototype

The main improvements done for the final prototype are the following:

- We have developed a DM that supports *mixed-initiative* dialogues and *clarification* subdialogues. This DM uses an explicit *dialogue model* that defines generic dialogue phenomena. The DM uses dialogue plans to identify the user's *intentions* and to choose the next actions to do.
- We have built a Natural Language Generator module (NLG) based on the communication actions and attributes defined in the dialogue model.
- The voice grammars have been extended in order to support more flexible communication. In previous prototypes, the grammars modeled the possible user's answers to the last system question. In the final version, the grammars also model other possible user's interactions.
- An ontology represents the services supported by the system. This ontology is accessed by the Natural Language Parser and Processor (NLPP) and by the service and language discovery engine used in the first interaction with the user.

The architecture of the final system is shown in Figure 3.6. In this system, both voice and text input are processed by the NLPP module. The text input is introduced through the web-based Text Client (TC), and the written sentences is passed to the NLPP. The voice input is introduced through the telephone, and the transcription of the user's utterance is passed to the NLPP.
In the first interaction with the user, when neither the language nor the service is know, the OM performs the service and language discovery task (SS). The OM is also used for generating the the NLPP knowledge base.
Once the Application Manager module (AM) has got the semantic interpretation of the user's intervention, it is sent to the DM. The DM uses libraries containing the communication plans necessary for each service task. Using the plan the system guides the user to provide the value

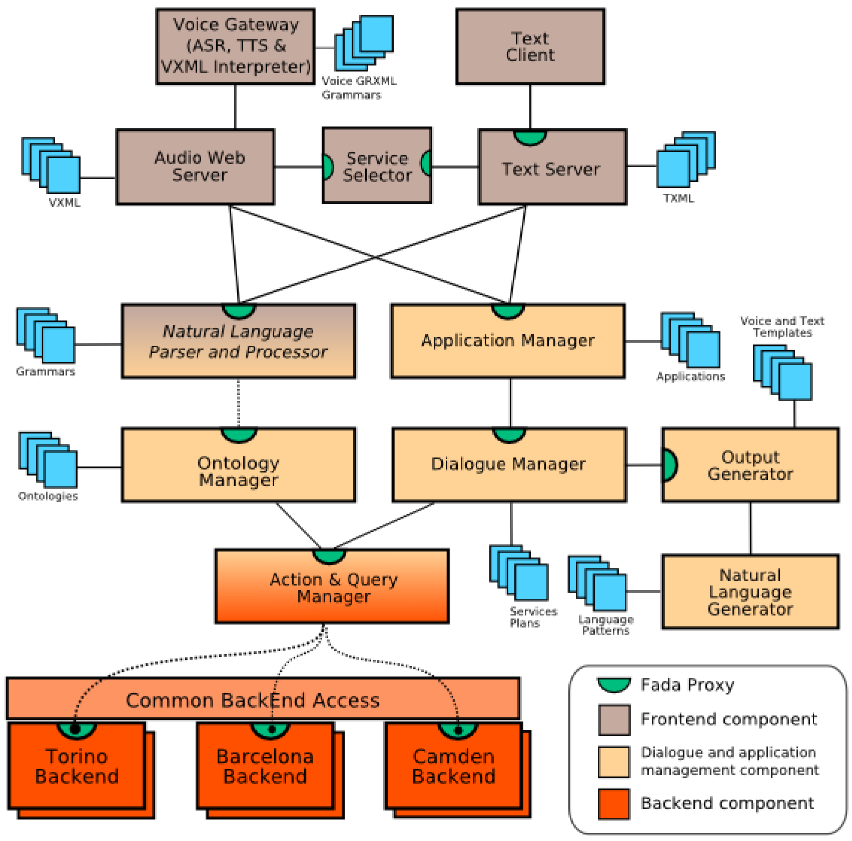


Figure 3.6: Third Prototype

of all the input parameters that the service needs. Once all the parameters has been provided, the DM accesses the database and gives the user the results of the query. The result can be a transaction status or a list of selected objects. The database access is performed through the AQM as in preliminary prototypes.

3.4 Information State Dialogue Management

In order to improve the communication process and the engineering features of our initial DS, we design a new DM component. Our main goal was to develop a flexible and efficient DM independent on the specific service and language, and oriented to guide the citizens when accessing the web by phone and text.
The DM we design for the final prototype incorporates an explicit dialogue model, based on the Information State Update theory (ISU). In the ISU approach ([Traum et al., 1999]), the dialogue model supports user-initiative dialogues and other complex phenomena, such as feedback strategies, which are necessary to achieve a friendly communication. Moreover, in the DM we design, the task model is completely separated of the dialogue model. The ISU theory combines both structural-state and plan-based dialogue approaches.

The ISU approach provides:

- the tools to formalize the information considered by the dialogue participants beliefs, intentions and plans,
- and how this information is updated during the communication.

ISU consists of the following components:

- The description of the information state (IS)
- A set of dialogue moves
- A set of rules updating the IS
- A control strategy to select next rule

I implemented the DM following the Larsson's computational approach to ISU described in Larsson [2002], which is focused on the notion of *Questions Under Discussion* **(QUD)**. The DM uses the plans to detect the questions raised by the user. These plans are described in depth in Section 3.4.5. They are generated statically and described and stored in libraries, reducing the dialogue management complexity.

I have adapted the Larsson's proposal considering the DS has to provide access to the two types of web services described in Section 3.1: transactional and informational. Following this goal I have implemented the ISU components described in Section 3.4.2.

3.4.1 The Set of Dialogue Moves

In our system, the resulting interpretation of the user's spoken utterances or written sentences consists of one or more moves, plus the semantic content. The semantic contents consists of the attribute and one or more values. These moves are sent to the DM, that processes them updating the discourse context, and then the DM selects the next system moves. System's utterances express the moves chosen.

These are the moves typified in our DM:

- Moves performed by both the user and the system
  - ***greet***. It is used at the beginning of the communication. It is associated with the load of the initial plan.
  - ***asking***. It is related to the raise of an issue. The plans describe the possible questions the user can raise and the actions the system has to follow to solve them. Each asking move of the user corresponds to an issue the DM must solve. These issues correspond to the web services tasks. The DM uses the plan associated with the task to guide the user to give all information the service requires. An asking move of the system is an issue to be raised by the DM. These moves are represented in the plans, and corresponds to the input parameters of the service tasks that the user must provide.
  - ***answering***. It concerns addressing an issue. When the answer is performed by the user, its content is relevant to the questions in the communication plan. These moves correspond to the input parameters of the service. When the answer is performed by the system its content is the result of the question the user requested. Basically, these moves correspond to the output parameters of the service.
  - ***feedback***. It is used to generate phrases indicating an understanding referred to the previous sentence. There are basically three types of feedback: positive (conf), negative and interrogative (afc). The positive one is an implicit confirmation of the understanding, e.g. *conf(serviceId, ca) - Understood. You want to consult the Cultural Agenda*. An interrogative feedback is an explicit confirmation of the understanding, e.g. *afc(serviceId,ca) - You want to consult the Cultural Agenda. Is that correct?*. The system also needs to use the negative feedback in several cases, such as when *no input*, *no match* or *error* occurs: *- I'm sorry, I did not hear anything from you, - Pardon? I don't quite understand you, - I'm sorry, there was an error consulting the database.*
  - ***confirmation***. It is an affirmative or negative answer to a Yes-No Question.
  - ***pagination***. It is the move used to present a list of result to the user. The pagination move contains a list of objects. This list could be too much long for browsing, specially in voice channel, where the user should remember all the information the system is providing. The channel is responsible for specifying how many items are presented to the user in each interaction (results window). After presenting these items, the system allows the user to select one of them, to get next window of items or to skip the enumeration.
  - ***quiting***. It is used at the end to indicate that the communication is over.
- Moves only performed by the system
  - ***executing***. It is a **transactional** access to the database of a transactional service. This type of move updates the database. The result of executing this move is another move with an attribute that indicates the status of the transaction.
  - ***exequery***. It is a **select query** to the database. The result of executing this move varies from one query to another. The possible resulting moves are:
    - answering, the database finds only one item that matches the query. In this case the database stores an answer move for each attribute of the item with the attribute's values.
    - pagination (list of results), the database finds a list of items that matches the query. The list is presented to the user in paginated mode. A *window* of N results is defined, and the system presents N items to the user. The user may select one item from the N presented or ask for the next N items, the previous N items, or exit the paginated mode.

    * asking (missing query parameters), the database finds a lot of results. The system calculates which attribute better distributes the query results and returns an asking move for the attribute.
    * feedback (no results, bad input value or access error), after any other behaviour different from the above.
  - ***clarify***. It is an interaction move in which the system presents some options to the user and the user has to choose one of them. For example, in case the user provides ambiguous information to a previous asking move, this information could be related to different plans. Thus, the DM needs to ask the user to clarify which service or task he wants to use. Different plans can contain actions related to the same information. For example, both Cancellation and Collection tasks of the LOC service needs to know the address of the user. Most of the administration proceedings requires to know the personal data of the user. Moreover, different services could use the same information. For example the price attribute is related to the collection for companies, and it is an event attribute.
  - ***reset***. It is a control move used to return to the beginning of the plan specified.

The sentences of the user and the system can be represented by one of these dialogue moves, or a combination of them. For example, an implicit confirmation (feedback move) followed by a question (asking move).

```
Understood. The place is Filmoteca De Catalunya.
I apologize there are no available events with those characteristics.
Do you want to do another search?
```

3.4.2 The Information State

The Information State (IS) is the information related to the state of the communication between the user and the system. The IS contains the information *shared* between the system and the user, and the *private* information only known by the system. The information described below is represented in separated stacks and queues.

- The **SHARED PART** of the IS description consists of the following information:

Commitments (COM). The information that appears in communication and that is accepted (not rejected). The commitments consists of the Questions Under Discussion (QUD) and the results. The information that any participant has expressed (the believes) becomes a commitment if it is not rejected. For example, the next sentence contains the query focus for the CA:

```
I want to go to the cinema today
```

The system recognizes two attributes to be commitments: the date (value is *TODAY*), and the type of event (value is *CINEMA*).

Question Under Discussion (QUD). This information consists of the local unsolved QUDs. The attributes in this stack are the ones available for ellipsis: the ones related to the available PLANS, as well as the ones the system has to confirm. In the next example, the attribute *serviceId* is stored in the QUD:

```
I want to go to the cinema today
   asking ( serviceId, ca )
   answering ( eventType, cinema )
   answering ( date, today )
```

ISSUES. This information consists of the open issues. That is, that attributes that have been accommodated or optimistically grounded, and have to be confirmed yet. In case of voice interventions there is also a confidence level [0..1] associated to them. If the confidence level is less than 0.6, the system asks for an explicit confirmation. In the example, both the event and the date attributes would be stored in ISSUES because the next user intervention would be related to the attributes. Once confirmed, they are stored in QUD.

```
U: I want to go to the cinema today. (0.6)
QUD:
   ( serviceId, ca )
ISSUES:
   ( eventType, cinema )
   ( date, today )
S: Do you want to go to the cinema? The date you are interested is today?
   afc( eventType, cinema )
   afc( date, today )
U: yes.
ISSUES:
QUD:
   ( serviceId, ca )
   ( eventType, cinema )
   ( date, today )
```

Past Movements (PM). Both, the user and the system interventions are stored in order to solve simple references. In the previous example, the system knows that the confirmation move is related to the event and date attributes because there is a past move containing a confirmation question (afc).

```
U: I want to go to the cinema today. (0.6)
S: Do you want to go to the cinema? The date you are interested is today?
   afc( eventType, cinema )
   afc( date, today )
U: yes.
   answer( yes )
PM:
   afc( eventType, cinema )
   afc( date, today )
```

The affirmative confirmation is related to the previous explicit confirmation questions.

Last Utterance (LU). This information consists of the last utterance processed, which moves are in NIM. The information related to the intervention consists of the confidence level, the participant (user or system) and the parser status (nomatch, noinput, error).

ACTIONS. That is the results of the database queries performed. There are 4 different types of results: no results, one result, a list of results and too much results. The proposition containing information with the *'fact that the query has been done'* is stored in COM, and the action containing the type of the results of the query is stored in the stack ACTIONS. The specific results are stored in other stacks. For example, attributes for the single result are stored in the stack containing the believes (BEL), while the fact that there is one result is stored in ACTIONS.

```
I want to go to the cinema today.
ACTIONS:
 exequery ( listofevents, single )
COM:
 exequery ( listofevents, done )
BEL:
   ( title, Pirati dei Caribi )
   ( date, today )
   ( eventType, cinema )
   ( price, 4.50 )
   ( location, Corso Rimini 54 )
   ( participant, ( Johnny Depp, Orlando Bloom, Keira Knightley ) )
```

The data from the query stored in BEL is moved to the stack COM once it is presented to the user and the user accepts it.

- The **PRIVATE PART** consists of the following information:

AGENDA. This information consists of the moves to do in the near future. It determines the next system moves. Depending on the user's moves, a plan is loaded. The actions in the stack PLAN are related to the system actions to achieve the goal of the plan. In each interaction some actions from the stack PLAN are selected for the next system intervention, and they are placed into the AGENDA.

```
U: When is played the Pirati dei Caribi?
S: You are interested in the Pirati dei Caribi. The date is today. Do you want to hear a full description?
AGENDA:
 conf ( title, Pirati dei Caribi )
 answering ( date, today )
 asking ( fulldescription )
```

In this example, the system implicitly confirm the information given by the user, answers the user his previous question and asks him what to do next.

PLAN and SUBPLAN. The PLAN consists of the long-term actions. It can be seen as the receipt for achieving the user (and system) goal. PLAN is updated in each interaction, and it is loaded when the user raises an asking move. PLAN actions can be simple or complex. The latter are related to subplans, which are composed of a set of simple actions governed by preconditions. The reference to a complex action in the plan is named ***macroaction***. When a macroaction in the plans is found, the subplan related to it is executed. A plan could be differently executed depending on intermediate user's answers. These alternatives are specified in subplans containing actions with preconditions.
This is an excerpt of the plan for the CA service:

```
I want to go to the cinema today
PLAN:
   asking ( queryFocus )
   asking ( requestedData )
   macroaction ( valuefocus )
   exequery ( listofevents )
   macroaction ( listofevents )
   ...
```

In this example the macroaction *valuefocus* is related to a subplan for asking the values of the query attributes, the macroaction *listofevents* is related to a subplan for answering the values of the requested data.

Each time the system ends the process of updating the IS, the plan loaded in the stack PLAN is executed in order to put into the AGENDA the next actions. When a *macroaction* is selected from the stack PLAN, the subplan related to the *macroaction* is loaded into the stack SUBPLAN. Then, all the preconditions of each action in the subplan are evaluated, and each time a precondition is matched, the action related to it is pushed in the stack PLAN. The rest of actions are removed from stack SUBPLAN. This process allows the DM to know the actions related to a subplan. The, given a reject move in the next user's intervention, all the actions related to the rejected subplan must be removed from the stack PLAN.

```
PLAN:
   asking ( queryFocus )
   asking ( requestedData ) default: title, condition: not queryFocus title
   macroaction ( valuefocus )
   exequery ( listofevents )
   macroaction ( listofevents )

U: I'm looking for events in my neighborhood
   asking ( serviceId, ca )
   answering ( queryFocus, location)
SUBPLAN:
   asking (location), condition: (queryFocus location)
PLAN:
   asking ( location )
   macroaction ( valuefocus )
   exequery ( listofevents )
   macroaction ( listofevents )

S: Ok. You are looking for titles. Which place are you interested in?
U: Well, I prefer events for today anywhere.
   answering ( queryFocus, date )
   answering ( date, TODAY )
SUBPLAN:
PLAN:
   exequery ( listofevents )
   macroaction ( listofevents )
```

In this example, the subplan of query attributes is loaded, and the action of asking the location is the only action which precondition is matched and loaded in the stack PLAN. Then, the user change the query attribute. The DM rejects the previous believe (queryFocus, location) and uses the stack SUBPLAN for removing the actions from stack PLAN. Then the plan is executed again, and the next action will be to execute a query.

Believes (BEL). Believes are propositions obtained from the user or results of a database query. Propositions from the user are believes until they are accepted and become commitments. After the user interaction the system performs some actions. Some of these actions could be answers, that are also stored in the stack BEL. If the user does not reject the previous believes, they are also accepted and they become commitments.

```
U: I want information about the Pirati dei Caribi
BEL:
   ( title, Pirati dei Caribi )
S: Ok, you are interested in the Pirati dei Caribi. Which information do you want?
U: The full description.
COM:
   ( title, Pirati dei Caribi)
BEL:
   ( date, today )
   ( eventType, cinema )
   ( price, 4.50 )
   ( location, Corso Rimini 54 )
   ( participant, ( Johnny Depp, Orlando Bloom, Keira Knightley ) )
```

In this example the title has become a commitment because the user has not rejected the system's answer. The rest of information in Believes is not Commitment because it has not appeared in the communication.

Temporal Information of the User (USRTEMP). It consists of relevant parts of the IS as they were right before integrating the latest system's utterance.

Temporal Information of the System (SYSTEMP). It consists of relevant parts of the IS as they were right before integrating the latest user's utterance.

The information about the latest utterances that has not been confirmed is represented in an independent data structure (LU, USRTEMP, SYSTEMP) that facilitates performing corrective and recovery processes in case the user gives a negative feedback or confirmation.

Not Integrated Moves (NIM). It contains the moves of the last utterance not yet processed. Both the user and the system moves are copied in the NIM queue. Then, the system processes the moves in the queue one by one. Each time one move is integrated in the IS, it is removed from the NIM. The update process finishes when the NIM is empty or none of the NIM moves can be integrated.

Pagination Moves (PAG). It is related to the query results. A pagination move is the result of getting a list of results from the database. A window of N results is defined for each channel. Thus, only the N results are given to the user in one single interaction. Information about all the results and the pagination status is stored in the stack PAG.

```
U: I want to go to the cinema today.
S: The event type is cinema. The date is today. I have found a total of 20 events,
   please choose one saying the corresponding number.
   1: Pirati dei Caribi,
   2: Spiderman 3 and
   3: Oceans'13.
   Say MORE to search for more events or EXIT to change the search criteria.
PAG:
   total: 20
   window: 3
   current: 1
U: more
S: These are the events from 4 to 6 of total 20.
   4: Lassie,
   5: Premonition and
   6: Zodiac.
   Say MORE to search for more events or EXIT to change the search criteria.
PAG:
   total: 20
   window: 3
   current: 4
U: exit
S: Do you want to do another search?
PAG:
```

In this example, the user first asks for the next items, and then skips the pagination.

3.4.3 The Rules updating the Information State

The interpretation of the user's utterances are moves plus the semantic representation of the related parameters. Then, the DM selects the next system's moves. These moves generated by the DM are selected from the ***AGENDA*** in the IS and stored in the stack ***NextMoves***.
Both the user and the system moves are processed by the DM. The DM applies a set of rules that, first, ***update*** the IS with the user actions, then ***select*** the next actions to be done by the system, and finally ***update*** the IS with the system actions. I have adopted and implemented the rules defined in Larsson [2002].

The rules that govern the way the IS is updated and the next system actions are selected are grouped in two separated main classes: ***update*** and ***select***. The complete list of rules is shown in table 3.1.
The ***update*** rules modify the data structures that represents the discourse context considering the IS content and the dialogue moves performed in the last utterance. They are separated in the subclasses shown below.

- *Downdating*. These rules simplify stacks by combining the information they contain.
- *Integrate*. These rules process one by one all the moves in the ***NIM***. There are specific rules for each possible move.
- *Accommodate*. These rules deal with user's interventions giving information not previously raised.

- *Accessing the plans.* These rules access the stored communication plans and load them in the private part of the IS representation.

- *Accessing the web services.* These rules control the access to the database services.

The ***select*** rules decide the next dialogue moves and their content. They are separated in the following subclasses:

- *Actions.* These rules select the actions that perform some IS changes, e.g. load a plan, select ***PLAN*** actions and actualize the ***AGENDA***.

- *Feedback.* These rules select feedback moves from the ***AGENDA*** and actualize the stack ***NextMoves***.

- *The rest of moves.* These rules select the rest of moves from the ***AGENDA*** and actualize the stack ***NextMoves***.

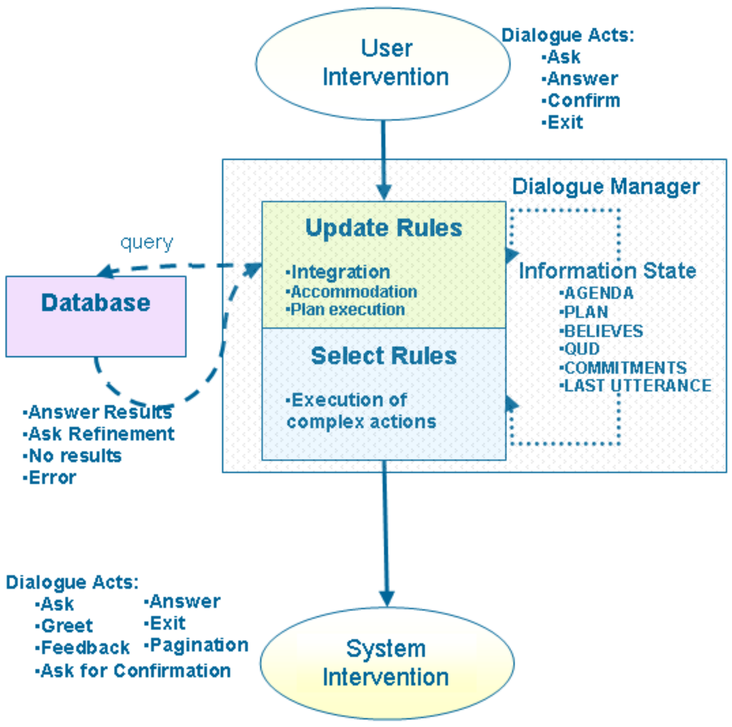


Figure 3.7: Dialogue Manager Flow

3.4.4 The Dialogue Management Algorithm

As was shown in Figure 3.7, the algorithm executed by the DM consists of repeating three separated processes. The system begins the conversation executing a *greet* move. Then, the system waits for the user's utterance. When the user utterance is received, the algorithm is executed until either the system or the user launch a quit move.

A pseudo code of the main algorithm is shown in Figure 3.8. First steps consist of the initialization of the data structures and the system variables. Then, the activation of the rules begins when the semantic interpretation of the user's intervention is sent to the DM. This semantic interpretation consists of the set of moves performed by the user together with the content of the

UPDATE	**function's name**	***SELECT***	**function's name**
	grounding		backupSharedSys
	backupSharedUsr		downdateMoves
Downdating	downdateNIM		
	downdateQUD		
	downdateActions		
	downdateISSUES		
Plan Access	load_plan		
	remove_Findout		
Service Access	execute_consultDB		
	execute_queryDB		
	rule's name		**rule's name**
Actions	selectIcmUndIntAsk	Actions	clarifyIssue
	selectIcmUndIntAnswer		clarifyIssueAction
	rejectIssue		selectConfirmAction
	rejectProp		selectConfirmActionError
	selectIni		selectConfirmActionRefine
Integration	retract		clarifyDependentIssue
	integrateUsrAsk		selectRespond
	integrateSysAsk		reraiseIssue
	integrateConfirm		selectFromPlan
	integrateNegIcmAnswer		selectReset
	integratePosIcmAnswer		selectAnsweringFromPlan
	integrateUsrAnswer		selectDefaultValue
	integrateUsrFullAnswer	Feedback	selectIcmConNeg
	integrateUsrPagination		selectIcmPerNeg
	integrateSysAnswer		selectIcmSemNeg
	integrateSysPagination		selectIcmUndNeg
	integrateUndPosICM		selectPagNo
	integrateUndIntICM		selectIcmOther
	integrateUsrPerNegICM	Moves	selectAnswer
	integrateUsrAccNegICM		selectBooleanAnswer
	integrateOtherICM		selectPagination
	integrateGreet		selectAsk
	integrateReset		selectConfirm
	integrateSysQuit		selectGreet
	integrateUsrQuit		selectQuit
Accommodation	accommodateIssues2QUD		
	accommodateQUD2Issues		
	accommodatePlan2Issues		
	accommodateCom2Issues		
	accommodateCom2IssuesDependent		
	accommodateDependentIssue		
	accommodateAction		

Table 3.1: Dialogue Manager Rules

intervention, and they are stored in the object *utterance*. The `update` algorithm (3.9) processes the object *utterance* and updates the ***IS***. When the `update` algorithm finishes, the next actions to do are in the stack ***AGENDA***. Then, the `select` algorithm (3.10) decides which actions to store in the stack ***NextMoves***. The stack ***NextMoves*** generates the object ***utterance*** of the system intervention. The stack ***NextMoves*** is also used to call the module LG that generates the language-dependent sentences of the system. Those sentences are sent to module OG, that builds the XML page. This XML page is finally sent to the frontends, and the system waits for the next user intervention.
The algorithm finishes when the user realizes a quit move, or the system finds a quit move in the stack ***NextMoves***.

```
Algorithm DialogueManager (plans_libraries, resources, Channel, Lang)

NextMoves ← {}                                           // Initialize the stack NextMoves
participant ← system                                     // First participant is the system
IS ← initialize_IS                                       // Creates the private and shared IS structures
state ← running                                          // Sets DM status running

read ( plans_libraries )                                 // Reads plans from libraries
read ( resources )                                       // Reads information about the plan's attributes
IS.private.PLAN ← plans.main_plan                        // Loads the initial plan
push NextMoves ← greet                                   // First move system does is greet
answer ← sys_process( )                                  // Calls the LG, the OG, and process the system utterance
show answer

repeat
  utterance ← wait_user_utterance                        // The user utterance is the NLPP interpretation of the
                                                         // sentence, either written or transcribed from voice
  NextMoves ← usr_process( utterance )
  answer ← sys_process( NextMoves )                      // stack NextMoves generates system answer and utterance
  show answer                                            // Returns to the AM the XML page
until ( state ← stop )                                   // Only the system can change the state when a quit
                                                         // move is processed

function sys_process()                                   // Processes the system utterance
  sentences ← callLanguageGenerator( Lang, NextMoves, IS ) // Generates the messages
  answer ← callOutputGenerator ( Channel, sentences, IS )  // Generates the XML page
  sys_ut ← NextMoves                                     // Creates the utterance from NextMoves
  update_algorithm( sys_ut )                             // Calls the update IS algorithm
  return answer                                          // Returns the XML page

function usr_process(usr_ut)                             // Processes the user utterance
  update_algorithm( usr_ut );                            // Calls the update IS algorithm
  NextMoves ← select_algorithm                           // Calls the select moves algorithm
  return NextMoves                                       // Returns the next moves to perform
```

Figure 3.8: Pseudo code for the Dialogue Manager algorithm

The `update algorithm` begins grounding the utterance, either from user or system. Grounding the utterance consists of preparing the IS for being updated and copying the utterance moves to the ***NIM*** queue. Then, the system performs a backup of the user's stacks, and if the utterance is from the user, the ***NIM*** queue is downdated (it is checked the correctness of the moves in ***NIM***, and reordered).
Second, there comes the loop for updating the IS, where the `do_update` function is called until no more updates are performed.
Finally, when there is a user's utterance and succeeds the integration process, the algorithm removes any resolved issues from stacks ***ISSUES*** and ***PLAN***, and, if necessary, loads a new plan.

The function `do_update` of the `update algorithm` performs a loop for cycling through all non-integrated moves and tries to integrate them calling the function `do_integrate`. Actions and accommodation rules are applied when any move has been integrated from the ***NIM*** queue. After an action or move accommodation, the system tries to integrate more moves in the ***NIM***.

If integration fails, the ***NIM*** queue is shifted (first element of the queue is set at the end) and integration is tried again. This process continues until all the elements in the queue has been shifted, and all moves have had a shot at being integrated.

The function `execute_plan` is responsible for executing the first action in the stack PLAN. When there is an asking move, the DM must check that the question has not been already answered. If the first action is a macroaction, the subplan related to the macroaction must be loaded. Finally, there are specific rules for executing a database transaction or a database query.
In a transaction, the DM considers three possible results: done, rollback (not possible to do) and error. The DM updates the move containing the requested execution, adds the information about the execution result, and puts the move in the stack **BELIEVES**. Then, other rules will place the move for communicating the result to the user in the stack ***NextMoves***.
In queries, the DM considers four types of results: no results, one result, a list of results or too much results. If there are no results, a new move containing the exequery move with this information is stored in the stack BEL. If there is only one result, then the same move will be created and stored in the stack ACTION containing the result from the database. Since an item can contain more than one attributes, the DM selects which information is shown to the user, depending on the plan and the user's answers. If the database returns a list of results, they are stored in a pagination move. This move is stored in the stack ACTIONS. The `update algorithm` prioritizes the rules related to pagination while there are moves in the stack PAG. When the user selects one item from the list, the information related to this item is stored in the stack BEL, as in the case of only one result.

The `select algorithm` involves two parts: first selects an item in the ***AGENDA*** and then selects a move which realizes this action and put this move into the stack ***NextMoves***.

3.4.5 The Services Plans

The DM uses communication plans to recognize the user's intentions and to determine the next action the system should perform to satisfy the user's needs.
Traditionally, in plan-based systems, plans are generated dynamically during the communication using plan recognition techniques from the artificial intelligence area to recognize the user's intention, and templates to perform actions. Once the DM has recognized the user's goal, the system can optimize the plan to achieve it, considering the dialogue history. Although this technique is powerful, it is not the most efficient for all types of dialogues. For simple dialogues on web services, such as those supported by our system, plans can be defined and stored in a library when a new service is incorporated, they do not need to be generated dynamically, neither to be optimized. Since the current web services supported by our system are not very complex, the DM we have developed does not need general reasoning capabilities for planning and plan recognition.

In our system, there is a plan for each service task, that is considered as the user's goal. There are more plans than services, because a service can perform more than one task. Particularly, in the services currently supported by our system (Section 3.1) there is one plan for the *CA service*, three plans for the *LOC service* and one plan for the *AOD service*, plus one plan for the initial interaction.

The plan defines the information the system needs from the user (asking and answering moves), and the information the system needs from the database (exequery and executing moves). This information is mainly related to the parameters of the service task. The plan also determines the relations between all the moves, by means of preconditions or definition of subplans.

A communication plan can be decomposed into simple and complex actions (related to subplans). In current implementation of the DM, the simple actions can be:

```
Algorithm update_algorithm ( utterance )

Grounding                                          // Prepares IS and copies utterance moves to IS
backupSharedUsr                                    // User stacks backup

if ( utterance is from user ) downdateNIM          // Checks the order the moves must be processed
repeat
  done ← do_update()
until ( not done )                                 // Repeats this until no integration and no accommodation is possible
if ( utterance is from user )
  repeat
    done ← execute_plan()                          // Repeats exec plan until it fails
  until ( not done )
            // Repeats until no more execution is possible at the current stage of the dialogue
downdateQUD                                        // Checks information in the stacks

function do_update() : boolean
  mark first position of NIM                       // Non-Integrated Moves queue
  repeat                                           // Tries to integrate as many moves as possible by cycling through the NIM
    done ← do_integrate
    if ( done )
      reset NIM to initial position
    else
      shift NIM
  until ( not fully shifted )                      // Repeats this until nothing can be integrated
  try Actions rules
  try Accommodation rules
  return ( True if any rule applied )              //If accommodation or actions has been applied, does the same thing again

function do_integrate() : boolean
  try Integration rules                            // Tries to apply all the integration rules
  downdateISSUES
  downdateActions
  remove_Findout                                   // Removes already solved ISSUES
  load_plan
  return ( True if any Integration rule applied )

function execute_plan() : boolean
  fact ← first IS.private.PLAN
  if ( fact is asking move )
    return remove_Findout
  else if ( fact is executing move )
    return execute_consultDB                       // Executes transactional database access
  else if ( fact is exequery move )
    return execute_queryDB                         // Executes informational database access
  else if ( fact is macroaction move )
    return evaluate and load subplan
          // If the subplan can be executed, evaluates its actions and put the matched ones in the PLAN
  else
    return False
```

Figure 3.9: Pseudo code for the Update algorithm

```
Algorithm select_algorithm() : stackofmoves

next_moves ← {}
backupSharedSys                              // System stacks backup
try Action rules
repeat
  try Feedback rules ( next_moves )
  try Moves rules ( next_moves )
until ( no rule applied )                    // Tries to select as many moves as possible
downdateMoves                                // Checks a friendly order of moves
return next_moves
```

Figure 3.10: Pseudo code for the Select algorithm

- ***asking***, the system asks the user the information needed to perform the task.
- ***answering***, the system gives information to the user.
- ***confirm***, the systems asks the user to confirm some data.
- ***backend access***, the system accesses the service to perform a transaction or execute a query.

Subplans are related to complex actions and consist of a set of simple actions. A precondition governs each action. These preconditions determines in which discourse context the action must be executed.

Each action contains additional information about the parameter is related to. This informations is basically related to the services parameters, and it is useful in some tasks of the DM and the DS. An action consists of:

- The arity of the parameter: if it allows just one value or more than one. This information is mandatory.
- The default value. There are parameters with a default value. In most of these cases, the value of those parameters are not asked to the user, but it is considered if the user gives this information.
- The precondition of an action. It is optional and mainly related to the values of other parameters of the plan.
- The type of the value. It is optional. This information is useful for recognition process. For example, specific grammars are used for recognizing the value for boolean or integer parameters.

Figures 3.11, 3.12 and 3.13 are the plans in our system. There is a separate plan for each service task. Plans are represented in ordered bottom-up stacks. The elements are simple actions (*action* in the Figures) and complex actions (*subplan* in the Figures).

```
#ini plan
plan serviceIdentification init
action asking serviceId uniary
endactions
endplan
```

Figure 3.11: Initial Plan for the Information State-Based DM

```
#LOC PLAN
plan loc serviceId
action asking serviceTask uniary
endactions
#information
plan information serviceTask
action macroaction endloc
action asking returnmenu boolary
action answering gpaddress uniary
action executing gpaddress uniary
action asking userAddress uniary
endactions
endplan
#collection
plan collection serviceTask
action macroaction endloc
action asking returnmenu boolary
action macroaction dotransaction
action asking confirmdate boolary
action executing collectiondate uniary
action asking userPhone uniary
action asking userAddress uniary
action macroaction depcollection
action asking userType uniary
endactions
endplan
#cancellation
plan cancellation serviceTask
action macroaction endloc
action asking returnmenu boolary
action answering cancelled uniary
action executing cancelled uniary
action asking transactionId uniary
endactions
endplan
subplan depcollection
action asking throwingObject multiary commitment userType private
action asking totalVolume uniary commitment userType company
endactions
endsubplan
subplan dotransaction
action answering confirmdate boolary commitment confirmdate no
action answering transactionId uniary commitment confirmdate yes
action executing transactionId uniary commitment confirmdate yes
endactions
endsubplan
subplan endloc
action quiting
action reset serviceIdentification uniary commitment returnmenu yes
endactions
endsubplan
endplan
#FI LOC PLAN
```

Figure 3.12: 'LOC Service' Plan for the Information State-Based DM

```
#CA PLAN
plan ca serviceId
action macroaction endca
action macroaction initca
action asking anothersearch boolary
action macroaction descr
action macroaction listofevents
action exequery listofevents multiary
action macroaction valuefocus
action asking requestedData multiary default title commitment queryFocus title noext
action asking queryFocus multiary
endactions
subplan valuefocus
action asking time uniary commitment queryFocus time
action asking price uniary commitment queryFocus price
action asking location uniary commitment queryFocus location
action asking date uniary commitment queryFocus date
action asking subeventType uniary commitment queryFocus subeventType
action asking eventType uniary commitment queryFocus eventType
action asking title uniary commitment queryFocus title
endactions
endsubplan
subplan endca
action quiting
action reset serviceIdentification uniary commitment returnmenu yes
endactions
endsubplan
subplan initca
action asking returnmenu boolary
action reset ca uniary commitment anothersearch yes
endactions
endsubplan
subplan listofevents
action macroaction eventinfo
action pagination listofevents multiary commitment list listofevents listindex requestedData
endactions
endsubplan
subplan eventinfo
action asking fulldescription boolary believes description *
action answering time uniary commitment requestedData time
action answering price uniary commitment requestedData price
action answering title uniary commitment requestedData title
action answering participant uniary commitment requestedData participant
action answering subeventType uniary commitment requestedData subeventType
action answering eventType uniary commitment requestedData eventType
action answering music uniary commitment requestedData music
action answering date uniary commitment requestedData date
action answering location uniary commitment requestedData location
endactions
endsubplan
subplan descr
action answering description uniary commitment fulldescription yes
action answering time uniary commitment fulldescription yes
action answering price uniary commitment fulldescription yes
action answering title uniary commitment fulldescription yes
action answering participant uniary commitment fulldescription yes
action answering subeventType uniary commitment fulldescription yes
action answering eventType uniary commitment fulldescription yes
action answering music uniary commitment fulldescription yes
action answering date uniary commitment fulldescription yes
action answering location uniary commitment fulldescription yes
endactions
endsubplan
endplan
#FI CA PLAN
```

Figure 3.13: 'CA Service' Plan for the Information State-Based DM

In order to facilitate the generation of plans for a new service, we have defined general templates for the two types of web services the system has to support: informational and transactional services. In transactional services the system must ask the user for all the information corresponding to the mandatory input parameters that have not been previously given. In informational services the system must ask the user information to constrain the search (*queryFocus*) as well as the information requested (*requestedData*).

- **Transactional Services**

When accessing transactional services the user has to give very specific information to perform a particular task. Then, the service presents the transaction results. The general communication plan in dialogues to access transactional services is shown in Figure 3.14. As can be seen in the figure, the DM is responsible for the following actions:

- Collection of the information needed from the user. The system asks the user to give the values of the input parameters of the service.
- Access the service to perform the transaction.
- Presentation of the results or information about the process, i.e. *'The transaction has been correctly done'*.

A transactional service can support more than one task, i.e. in our LOC Service, it can be able to make a collection, cancel a previous collection and give information about the service itself. In these cases a plan for each service task is generated, plus a plan to determine the specific task the user is asking for.

We have adapted the general plan of transactional tasks to the collection and the cancellation tasks. The generation of a plan for the cancellation task is very simple. The only parameter which value has to be asked to the user is the transaction identification that was given when the collection was arranged. The result of this transaction is a message that confirms the cancellation or informs about a particular problem.
The information needed to fix a date for collection depends on the type of user. If the user is a particular, the collection is free and if it is a company, the price depends on the volume of the objects to be collected. For this reason, the first action in the plan for the cancellation task is to ask the user if he is a particular or a company. Then, the next action in the plan consists of a complex action composed of specific actions for each type of user. The next actions in the plan consist of asking the user the value of the rest of the task parameters needed. The lasts actions in the plan are related to the presentation of the results of the transaction. In this example, the DM answers the collection date and, in case of companies, the price and, finally asks the user to confirm the transaction.

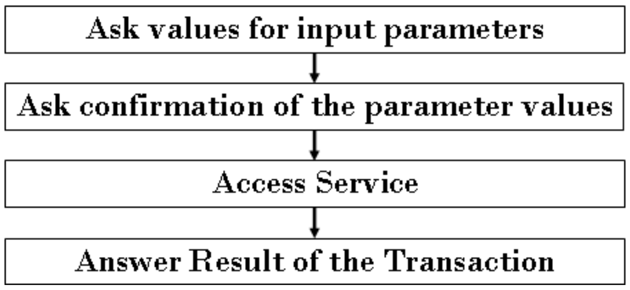

Ask values for input parameters
Ask confirmation of the parameter values
Access Service
Answer Result of the Transaction

Figure 3.14: General Plan for Transactional Services

- **Informational Services**

The user's intention when accessing a web informational service is to obtain specific information from the service. In most cases the DM has to guide the user to give the particular information about what he/she is searching. This information is used to restrict the search. Then, the results obtained are presented to the user. The general description of the actions the DM performs to guide the user to access informational services is shown in Figure 3.15.
In order to provide access to informational web services the DM usually performs the following actions:

- Collection of the information needed to restrict the search. This action consists in asking the user specific information about the elements searched. Unlike in the case of transactional services, the user may choose to give the value of only a subset of the possible parameters.
- Collection of the information requested from the search. In this action the user must specify which information is looking for. Although for each service there is a default description of the results (i.e. The title of the event), the user can also explicitly ask for different data (i.e. Where is played Pirati dei Caribi?).
- Access the service. In our DS, the DM accesses the web services through a specific module, which evaluates the number of resulting elements that satisfy the user's query. If two many elements are obtained then the module that access the service determines the new parameters to restrict the search, which are then asked to the user.
- Presentation of the results. Only a partial description of the elements obtained is presented to the user. Finally, once the partial description of the results is given, the DM asks the user if he wants a complete description of any of the elements presented.

The CA service gives information about the cultural events in the city. When accessing the service the user can search an event giving different data: the particular event title, the event type, the location, the date or any combination of these. The description of the events found includes information about all this data. In order to generate a communication plan for the CA service we have adapted the general plan for informational services to the service parameters. We have considered any of the service parameters (title, type, location, data) could be the focus parameters. At the same time, any of these parameters could be the requested data, although the title is the default requested data.

3.5 Natural Language Generation

This Section describes the generation of the system's answers. In particular, it describes the the OG module, which is responsible for generating the XML pages that communicate the DM with the frontends, and the NLG module, that generates the system's messages.

3.5.1 Output Generation

The DM Component communicates to the frontend layer through XML pages. It uses VoiceXML [W3C, 2006] pages for the voice channel, and TextXML [Hernández et al., 2006] pages for the textual channel. The OG is responsible for providing these XML pages to the DM.
The OG uses templates for building XML pages. These templates are compositional and use GAPs. In our context, a GAP is a mark in the XML document where other information must be inserted: the language, the messages, the attribute's names, the linguistic resources to be used, the control variables or other templates.

The OG considers four different types of interactions between the user and the system:

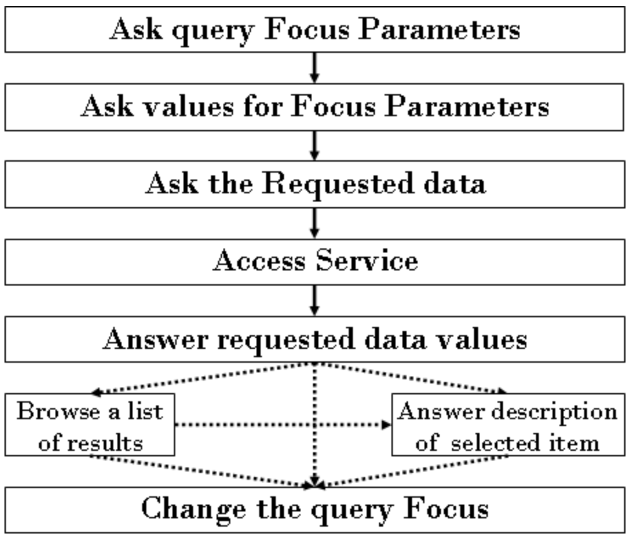

Ask query Focus Parameters
Ask values for Focus Parameters
Ask the Requested data
Access Service
Answer requested data values
Browse a list of results
Answer description of selected item
Change the query Focus

Figure 3.15: General Plan for Informational Services

- To obtain the language and the service identification
- To obtain the value for one or more attributes
- To present the results of the BE call
- To say goodbye to the user

In the first interaction between the user and the system, the language and the service that the user wants to use are established. Then, the DM needs to ask the user some attributes in order to complete a query to the database. These attributes could be asked together with an open question. However, if not all attributes are obtained, the DM goes over until they are obtained. When all the attributes are obtained, the DM accesses the BE and shows the results to the user. Finally, the communication is politely ended.

First interaction with the user is always the same. In the textual channel the system asks the user `May I help you?` in all the available languages in the DS. Then, the user's answer is sent to the SS module, that performs the service and language discovery. In voice channel, technical limitations do not allow to perform the interaction in the same way. The user first selects the language, and then the systems asks `May I help you?` in the language selected.

In next interactions, the DM uses the stack ***NextMoves*** for specifying the system interventions. This stack is also used by the OG in order to select and include in the XML files the information related to system's communications that aids the frontends recognizing the next user's sentences. In principle, the user will answer the system's questions, thus the language resources related to the asked parameters are prioritized over the rest of parameters. This information facilitates the disambiguation of the user's answer. The XML page is built from a general template. OG fills the template with the messages, the linguistic resources references, the language identification, and some other system information. Figure 3.16 shows part of the general template for generating VoiceXML files.

In the voice channel, in order to built the VoiceXML page containing the paginated mode interaction, the OG uses another template similar to the general one, but with some GAPs for an inline grammar [W3C, 2006]. This grammar models the specific pagination moves.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<vxml version="2.0" @APPROOT@ xml:lang="@VLANG@">

<link dtmf="0" event="help">
    <grammar mode="voice" src="@VXMLPATH@@GRAMMARPATH@gramsession.gram#rule_help" type="application/srgs"/>
</link>

<form>
            <var name="params" expr="'@UNFILLEDATTRIBUTES@'"/>

            <field name="@ATTR_NAME@">

                @GRAMMAR@

                @DTMFGRAMMAR@

                <prompt> @ATTR_PROMPT@ </prompt>

                <help> @ATTR_HELP_EVENT_1@ </help>

                <noinput> @ATTR_NOINPUT_EVENT_1@ </noinput>

                <nomatch> @ATTR_NOMATCH_EVENT_1@ </nomatch>

                @DTMFPROMPTS@

                <filled>
                        <assign name="NLPURL" expr="NLPURL+'&amp;utterance='+utterance+'&amp;langId='+langId"/>
                        <data name="nlp" srcexpr="NLPURL"/>
                        <assign name="extractedParamsArray" expr="getParamsArray(nlp)"/>
                </filled>

            </field>

            <block>
                <submit expr="AMURL" namelist="extractedParamsArray"/>
            </block>
    </form>
</vxml>
```

Figure 3.16: VoiceXML General Template

When generating the files containing next interaction, the appropiate sentences are selected from a list of sentences generated when adapting the system. Those sentences contains some GAPs to be filled at run-time. These GAPs correspond to the attribute values recognized in previous interactions. Examples of the use of GAPs are the confirmation interaction and the presentation of results, where the GAPs must be filled with the information obtained from the user and the information retrieved from the database respectively. The OG searches the sentence for these GAPs. When one of them is found, the OG extracts the attribute name. Then, asks the DM for the values of the attribute. The DM finds the value in the IS structure. There are few cases where the value needs a formatting before showing it to the user:

- Telephone. In voice system, it must be indicated that the value is a telephone, thus the speech synthesizer could read the number correctly.

- Date. The internal system date encoding must be translated into a user comprehensive expression.

- Translations. Some parameter's values need translation before inserting in the messages. These translations are provided by the NLG module.

3.5.2 Multilingual Language Generation

Our system produces messages in four languages: Catalan, English, Italian and Spanish. The messages are built as a concatenation of the phrases related to each move in the stack ***NextMoves***. We have created semiautomatically a set of phrases for each possible type of move for the two web services supported.

We have adapted the syntactic-semantic taxonomy described in Gatius [2001] (inspired in the Generalized Upper Model of Bateman et al. [1994]) to generate those phrases in the four languages we are working with. The approach we have followed consists of an appropriate representation of the relations between the different types of knowledge involved in communication: dialogue knowledge, conceptual knowledge and linguistic knowledge. When adapting the taxonomy we have focused on dialogues that guide the user to access the web services. Thus, the appropriate representation of the linguistic structures involved in the communication in these web services may limit the language that has to be generated at run-time.

The phrases generated (and stored in libraries) are related to the dialogue acts the system can perform: ask the user for a specific data, explicitly or implicitly ask for confirmation of a value, and present results. Each phrase is related to a particular move, and a particular task parameter. Several task parameters appear in more than one service, such as telephone, address and dates. Sentences for these parameters can be reused across services. Several other parameters are specific to one particular service. Sentences for these parameters are generated using the patterns associated with syntactic-semantic categories.

We have defined patterns to represent several forms in which the dialogue acts can be expressed. For each language, each class in the syntactic-semantic taxonomy has been associated with the linguistic structures involved in the dialogue acts: ask, answer, implicit confirmation and explicit confirmation. When the system is adapted to a new service, the task parameters have to be classified according to the syntactic-semantic taxonomy. Additionally, they have to be linked to the corresponding lexical entries (in each language). Then, the system messages can be automatically generated in each language by adapting the general patterns associated with each taxonomy class to the particular lexical entries.

The generated patterns are represented in XML files. The patterns have been organized to facilitate their access at run-time. In Section 3.5.2.3 we explain the structure of the files containing the patterns and how they are stored in libraries.

3.5.2.1 Patterns

The phrases used by our DM are built using Rules. The Rules are built from Patterns *(< pattern_name >)*, Concepts *([$concept_name])*, Attributes *([&attribute_name])*, attribute's Values *([@attribute_name])*, and Lexical entries *([lexical_entry_name])*. Concepts and Attributes classes are related to a language realization, as well as Lexical entries. Values of attributes are inserted in the phrases at run-time, since depends on discourse context and databases information. The patterns must take into account this issue, building as many value-genre-independent phrases as possible.

The Patterns contain semantic and syntactic information, but also morphological information. We have also classified those Patterns in three categories, that are applied only to Rules. The categories used are:

- **Open**. When the pattern generates an open question.
- **Mid**. When the pattern generates an implicit question.
- **Directive**. When the pattern generates a directive question.

Figure 3.17 and 3.18 shows an example of an ask Rule and the phrases generated. All the Patterns appearing have the same semantics but different syntactics. Using different patterns to express the same meaning we can generate different phrases with the same meaning, what makes the system more flexible and friendly. For instance, when the system needs to repeat the question, we don't need to use the same sentence. Examples of the final built phrases are shown in Figure 3.18.

```
ask<event_type> -->
   Open      ([q-word] [verb_copulative_sg] [art] <event_type> (that [verb_wish_s])?)
   Open      (What<event_type> [verb_wish_q]?)
   Open      (What<event_type> [verb_wish_prep_q]?)
   Mid       (What <event_type> [verb_wish_q](: [music], [theatre], [cinema], [art]
             or [appointment])?)
   Mid       (What <event_type> [verb_wish_prep_q](: [music], [theatre], [cinema],
             [art] or [appointment])?)
   Directive ([verb_request] [art] <event_type> [verb_wish_s])
   Directive ([verb_request] what <event_type> [verb_wish_s])
   Directive ([verb_choose] [art] <event_type>: [music], [theatre], [cinema], [art] or [appointment])
```

Figure 3.17: The Rule $ask < event\_type >$

```
ask-ca.event_type.0=what is the type of event that you are looking for?
ask-ca.event_type.1=What type of event do you want : music, theatre, cinema, art or appointment?
ask-ca.event_type.2=choose the type of event : music, theatre, cinema, art or appointment.
```

Figure 3.18: Some examples of phrases for $ask < event\_type >$ Rule of total 96 phrases

During communication the frontend modules use first the Open phrases. If the user does not answer or answers incorrectly, the system uses the phrases belonging to category Mid. In case the user fails again, the system uses the Directive phrases. Finally, after three attempts, the system uses a menu options, if possible. In several cases there are no menu options, for example for infinite values. In those cases the systems invites the user to call an operator. Phrases for menu options are possible to built when the possible values for the parameter are few or numbers, thus we can enumerate them and the user can use keywords or the touch-pad phone.

3.5.2.2 General Messages for all the Services

There are some speech acts not related to any parameter. Most of them are neither related to any particular service, but used by them all. We have treated those speech acts separately and we have linked them directly to their linguistic representation instead of using patterns to generate them. This representation consists of one or more utterances. In order to build a friendly service, we used paraphrasing and synonyms to get a more friendly and flexible dialogue. Examples of these Rules are shown in Figure 3.19.
There are also other speech acts related to a specific service among the ones supported. The most are related to results of the database access. These speech acts are not very productive and are used only in one service and in specific situations. We have linked them directly to its linguistic representation, however some of them could be represented by a pattern.

```
Apologize -->         [apo]
Please -->            Please
Opening -->           Hello |
                      Good [@hour_strip]
Greeting -->          Welcome to the automatic platform of [@city]. May I help you?
Not_understand -->    [apo], I don't understand
Not_Input -->         I didn't hear you
Not_Match -->         [apo], I didn't understand you
Repeat -->            Can you repeat, please?
Call_again -->        Please, call again
Contact_operator -->  Please, contact an operator
Ok -->                The query is successful
Error -->             [apo], an error has occurred
Acknowledge -->       Ok |
                      Understood |
                      I see
Thanks -->            Thank you |
                      Thanks
Thanks_bye -->        Thanks for using this service
Ask-More-Info -->     Would you like more information? |
                      Do you want any other information?
Wait -->              Wait a moment, please |
                      Let's see
Goodbye -->           Goodbye |
                      Bye
```

Figure 3.19: General messages associated to speech acts

3.5.2.3 The Files Representing the Language Patterns

We have organized the information related to the patterns by language. For each language, there is a XML file containing the information related to each specific service, and the information common to all services. The XML file is organized in sections named *language_descriptors*. Each *language_descriptor* contains the information related to a specific service, or the common information. Each *language_descriptor* has two parameters indicating which language and which service is related to. Each *language_descriptor* contains: Rules, Patterns, Concepts, Attributes, Lexical entries and Moves without patterns. Rules, patterns and moves without patterns are related to marks and phrases. Concepts, attributes and lexical entries are associated with phrases.
Figure 3.20 shows a representation of an example of the content of a *language_descriptor*. The operators used to describe the content are the following:

- **answw, ask**: the moves.
- **val_place, search**: related to the parameter.
- **Directive, Mid, Open**: indicates the category of the pattern.
- [**name**]: Lexical entry.
- $< name >$: Class pattern entry.
- [**$name**]: Concept entry.
- [**#name**]: Attribute entry.
- [**@name**]: Value entry.
- **()**: Optional constituents.
- |: More than one form is possible.

```
answ<val_place> -->
   (the [\$event]) [@event] [verb_related_place] in [@place]

ask<search> -->
   Directive  [verb_wish_q] [verb_search_by] <of_name>, <of_place>, [#date_attribute] (of the [\$event]),
              <of_people>, <of_time>, [#type_attribute] of the [\$event] or <of_price>?
   Mid        Please, describe the [\$event] giving me: the <of_name>, the <of_place>, the <of_date>,
              the <of_people>, the <of_time>, the <of_price> or the [#type_attribute] (of the [\$event])
   Open       Can you [verb_request] some data about the [\$event] or [\$event_pl] [verb_wish_s]?

<event_type> --> [#type_attribute] (of [\$event]) | [\$event] [#type_attribute]

[\$event] --> event

[#date_attribute] --> date | day

[verb_wish_q] --> do you want | would you like
```

Figure 3.20: A fragment of a *language_descriptor*

There can be different *language_descriptors* in one document related to different languages and different domains. The components of the *language_descriptor* are described next:

- The attributes of a ***language_descriptor*** are the language identification and the name of the domain.
- The attributes of a **Rule** are the name of the domain, the order, the name of the parameter that is related to, and the name of the move.
- The attribute of a **Pattern** is the name of the parameter that is related to.
- The attribute of a **Concept** is name of the concept that is related to.
- The attribute of an **Attribute** is the name of the attribute that is related to.
- The attribute of a **Lexical entry** is the name of the attribute that is related to.
- The **Marks** have a [!CDATA] value containing the identifier of the item the mark is related to. Their attribute is the type identification whether it is related to a pattern, concept, attribute, value or lexical.
- The **Phrases** have a [!CDATA] value containing a phrase.

Chapter 4

Conclusions

4.1 Our Contributions

In order to achieve friendly communication for different types of services and users, we have developed a DM component that uses a dialogue model that defines general dialogue behaviour. The DM supports different types of dialogues: system-initiative and user-initiative. The basic functions of the DM become more complex when dealing with user-initiative dialogues. Our DM uses communication plans to detect the user's interventions and to decide the next system's actions. In our system these communication plans are obtained when incorporating a new service, and they are represented explicitly. Our DM also deals with feedback related to grounding. Feedback strategies improve the human-machine communication because novice users are not aware of the system's capabilities, and because in user-initiative dialogues there are misunderstandings. Our DM informs the user about what has been understood in the last intervention, which default values have been used, and, when necessary, which type of error has been occurred.

We have also developed a NLG module for our DM. Most DSs uses simple paraphrasing and hand-crafted sentences for generating the system's messages. Our system uses a pattern-based mechanism where the patterns for expressing the possible system's dialogue acts are generated semiautomatically when a new service is incorporated. A syntactic-semantic taxonomy has been used for facilitating this task. This strategy has been used for representing the information related to the services supported by our DS and in four languages.

4.2 Evaluation

As described in Section 2.4, comparison of DMs (and DSs) is difficult because they depend on the degree of integration of the system components, such as the ASR and NLU components. In fact, the performance of the functionalities of the DM (and the DS) is specially dependent on the SR component. The main goal of most metrics for evaluating DM components is to maximize the users' satisfaction, and the user's satisfaction seems directly related to the mean recognition score.

The DS developed in the HOPS project has been recently evaluated by citizens, and early results demonstrate that the system behaviour is acceptable when it is used through the text channel, but that speech technologies present still some limitations for being used with conversational systems. English and Italian tests have obtained better results than Spanish and Catalan ones. This fact states the dependency of the overall system performance to the recognition modules. Both ASR and NLU modules of the system were developed in Italy, and resources related to Italian and English languages received more attention. Figures 4.1 shows the user's satisfaction results obtained in Candem, Torino and Barcelona.

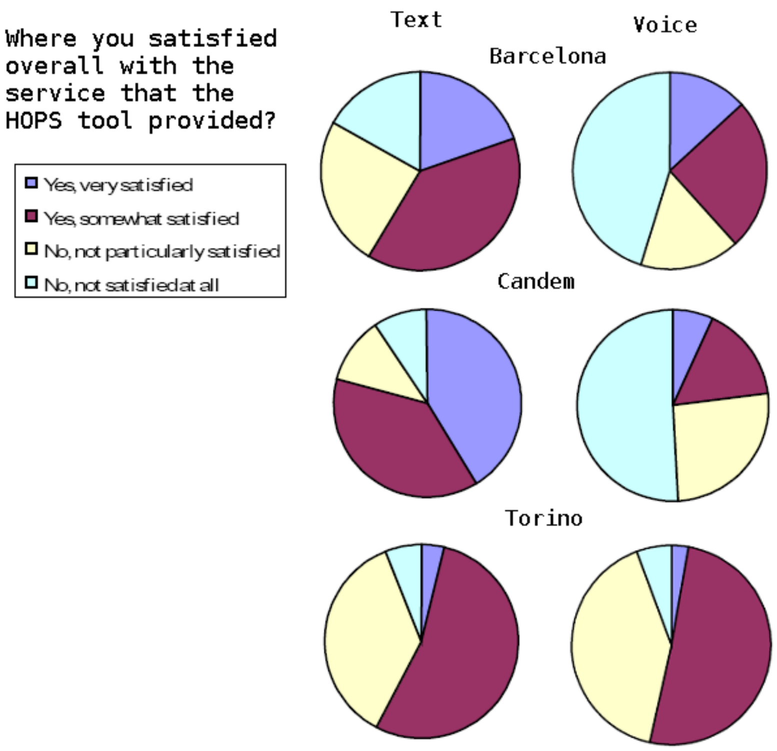


Figure 4.1: The user's satisfaction metrics

The improvements of the DM we developed motivated an early evaluation of the component while it was not completely integrated in the final prototype. In order to evaluate the DM independently of the speech we used the text mode to perform all the experiments. Furthermore, the access to the application's databases was simulated. We performed both qualitative and quantitative examination: we used a questionnaire to examine the overall behaviour of the DM, and we also carried on empirical experiments to evaluate the performance.

For the experiments ten volunteers (age average of 28) were recruited from the Software Department of our University (who had never worked on dialogue before). We gave each subject a list of eight scenarios they had to play (Figure 4.2). For the first six plays the users were considered novice and the system was configured for using prompts guiding the interaction. For the last two plays the users were considered experts and the system's messages used were more open, allowing more flexible communication.

The dialogues collected for each scenario were analyzed according to quantitative metrics that measure task success, dialogue efficiency costs (dialogue time, number of turns) and dialogue quality (correction turns, parser errors and appropriateness of the system's messages). Figure 4.1 shows the results obtained.

The average of dialog duration for CA and LOC scenarios is not correlated to the number of turns. The system asks the user for more information in the LOC scenarios (more turns) but the system's questions are easier to answer and understand (less time). When the user is considered an expert and the system's messages are more open, the system needs more time for the LOC (113). The first play has a significative higher correction ratio (50%) than the other plays. The 73,6% of the correction turns are errors due to the NLU. Most of the errors when processing the user intervention are caused by the low coverage of the grammars.

```
Scenarios for novice uses
 1. Ask for the movies in the filmoteca cinema.
 2. You are interested on information about the movie Volver.
 3. Ask about what is there today.
 4. You want to know the date of the play Mar i cel.
 5. You want your table to be collected.
 6. You want information about the collection service.
Scenarios for expert users
 1. Ask for information about a specific event.
 2. Ask for a collection.
```

Figure 4.2: The scenarios used

Scenario	Success	Time (sec)	Turn pairs	Corrections
Novices				
CA				
1	81,82%	144	10	50,00%
2	88,89%	119	9	50,00%
3	80,00%	63	6	50,00%
4	91,00%	137	9	50,00%
Results	**85,18%**	**116**	**9**	**27,78%**
LOC				
5	100,00%	93	13	23,08%
6	90,00%	107	13	30,77%
Results	**95,00%**	**100**	**13**	**26,93%**
Experts				
CA	66,67%	94	10	20,00%
LOC	90,00%	113	15	26,67%
Results	**78,33%**	**104**	**13**	**23,33%**
Total	**86,17%**	**107**	**12**	**26,01%**

Table 4.1: Quantitative Metrics

We have also analyzed the appropriateness of the DM responses. Figure 4.2 shows the percentage of the system's messages that are appropriate, inappropriate and ambiguous. The 93,20% of the system's responses were appropriate and only the system's messages for experts using the CA were below the average (89,6%).

Scenario	**Appropriate**	**Inappropriate**	**Ambiguous**
Novices			
CA	93,40%	2,80%	3,20%
LOC	93,90%	0,50%	4,30%
Experts			
CA	89,60%	4,00%	6,40%
LOC	96,00%	0,60%	3,50%
Results	**93,20%**	**1,90%**	**4,10%**

Table 4.2: The system's messages appropriateness

In order to evaluate the satisfaction of the users we ask the volunteers to complete questionnaire of ten questions. Each subject answered the questionnaire after finishing the first six scenarios (as novices) and once again after playing the last two scenarios (as experts). Figure 4.3 shows the questions and their results (rating scales are from 0 to 5).

0-strongly disagree, 5-strongly agree	**Novices**	**Experts**
Overall impression is good	2	3
The system provided the desired information	3	4
You feel understood by the system	2	3
You knew what the system expected from you	3	3
You perceived the dialogue as pleasant	3	3
You would have expected more help from the system	3	3
You were able to understand the system	3	4
You prefer to use another source of information	3	3
In the future, you would use the system again	2	3
You would prefer using the telephone	3	3

Table 4.3: Qualitative Evaluation

The results are good in general terms because task success is high and the system's messages, which are generated automatically, are appropriate in most cases. The users were frustrated when their interventions were not understood. It could be one of the reasons of the low ratings in the questionnaires. We think the coverage of the NLU have to be improved to achieve the user's satisfaction. Additionally, text mode have to be considered only as complementary mode, being the speech mode the preferred one by most of the users.

Qualitative results encourage and motivate the future work planned for this project thesis. Improvements can be done at different aspects of the DM we have developed. They are described in Section 5.

Chapter 5

Thesis Project

In this work we have presented an approach for information state-based dialogue management, as well as for natural language generation. We have also described how we have implemented these approaches within a DS.

The research lines we will follow will be:

1. In the DS we have developed, study how ontologies could be used for contributing in different aspects of dialogue, as well as for generating the system's messages and more appropriate linguistic resources for each service. Ontologies have been already used for the Natural Language Processing, and the use of ontologies representing the domain knowledge could be also used for improving the dialogue management.
2. We want to improve the process of adapting the system to new services and domains. Particularly, we want to study how the process of generating the communication plans for each new service can be improved, and how the system could be adapted to other types of applications (i.e. involving knowledge from several web sites).
3. Making the dialogue more flexible and adaptable to the user and the channel: creating new moves and strategies, extending dialogue model, and using a user model.
4. Evaluate the system. After the evaluation of the HOPS project, we have a large log of dialogues. We will use them to study main drawbacks in the system, and propose directions for the next improvements. Moreover, we want to establish a framework for evaluating those improvements at different modules and different levels in the overall DS locally developed.

This thesis work is expected to conclude by the middle of 2009. The Gantt chart in Figure 5.1 shows the devised project scheduling. The last 6 months of our work will be spent in the writing of the doctoral thesis dissertation.

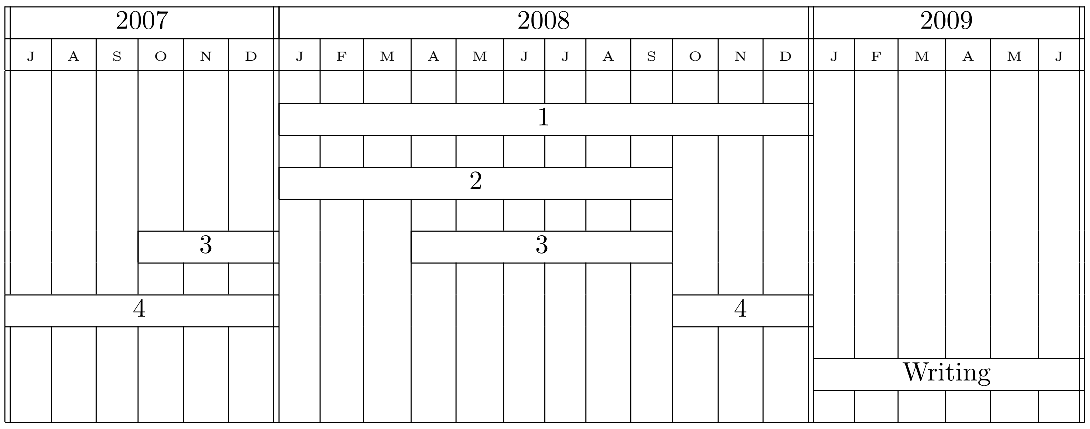


Figure 5.1: Thesis Project Scheduling

Chapter 6

Publications

Here are listed the publications that the research carried so far has produced, classified by which of the three main areas that the author's work has embraced they cover: architecture, dialogue management and use of ontologies.

6.1 Architecture

- [Hernández et al., 2006]
  Pablo Hernández, Jordi Sánchez, Ángel López, Sheyla Militello, Marta Gatius, Meritxell González, Eli Comelles, Leonardo Lesmo, Xavier Noria, Robert Salla, Carlos de la Morena, Jose Antonio Fernández and Alberto Deiro.
  HOPS Architecture Specifications
  European Project HOPS (IST-2002-507967) Deliverable D4.1, 2006

- [Gatius and González, 2005b]
  Marta Gatius and Meritxell González
  Un sistema de diálogo multilingüe dirigido por la semántica
  Revista de la SEPLN, Vol.34. , 2005.

6.2 Dialogue Management

- [Gatius et al., 2007b]
  Marta Gatius, Meritxell González, Eli Comelles and Leonardo Lesmo
  Natural Language Processing and Dialogue Management Development
  European Project HOPS (IST-2002-507967) Deliverable D4.3, April 2007

- [Gatius and González, 2007a]
  Marta Gatius and Meritxell González
  Discourse Management in Voice Systems for Accessing Web Services
  Workshop on the Semantics and Pragmatics of Dialogue (DECALOG), 2007

- [Gatius et al., 2007a]
  Marta Gatius, Meritxell González and Elisabet Comelles
  An Information State-Based Dialogue Manager for Making Voice Web Smarter
  16th International Worl Wide Web Conference (WWW2007), 2007

- [Gatius and González, 2007b]
  Marta Gatius and Meritxell González
  A Multilingual Dialogue System for Accessing the Web
  3rd International Conference on Web Information Systems and Technologies (WEBIST), 2007

- [González and Gatius, 2005]
  Meritxell González and Marta Gatius
  Un sistema de diálogo multicanal para acceder a la información y servicios de las administraciones públicas
  XXI Congreso de la SEPLN, I Congreso Español de Informática (CEDI), 2005

6.3 Dialogue and Ontologies

- [Gatius and González, 2006b]
  Marta Gatius and Meritxell González
  Integrating Semantic Web and Language Technologies to Improve the Online Public Administrations Services
  15th International Worl Wide Web Conference (WWW2006), 2006

- [Gatius and González, 2006a]
  Marta Gatius and Meritxell González
  Using Application-Specific Ontologies to Improve Performance in a Bottom-up Parser
  Workshop Knowledge and Reasoning for Language Processing (KRAQ), 2006

- [Gatius and González, 2005c]
  Marta Gatius and Meritxell González
  Obtaining Linguistic Resources for Dialogue Systems from Application Specifications and Domain Ontologies
  10th International Conference on Speech and Computer (SPECOM), 2005

- [Gatius et al., 2005]
  Marta Gatius, Meritxell González, Leonardo Lesmo and Pietro Torasso
  Natural Language Processing Technologies
  European Project HOPS (IST-2002-507967) Deliverable D3.2, 2005

- [Gatius and González, 2005a]
  Marta Gatius and Meritxell González
  Using Ontologies for Improving the Communication Process in a Dialogue System
  In the Proceedings of the Sixth International Workshop on Computational Semantics (IWCS-6), 2005

- [Gatius and González, 2004a]
  Marta Gatius and Meritxell González
  Utilización de ontologías en el desarrollo de sistemas de diálogo
  III Jornadas en Tecnología del Habla (3JTH), 2004

- [Gatius and González, 2004b]
  Marta Gatius and Meritxell González
  Ontology-driven VoiceXML Dialogues Generation
  Workshop at the Berliner XML-Tage (XMLTage), 2004

Acknowledgments

This research has been funded by the EU IST FP6 project HOPS (IST-2002-507967).

Bibliography

J. Allen and M. Core. Draft of damsl: Dioalog act markup in several layers. ""`http://http://www.cs.rochester.edu/research/speech/damsl/RevisedManual/RevisedManual.html`"", 1997.

J. Allen, G. Ferguson, N. Blaylock, D. Byron, N. Chambers, M. Dzikovska, L. Galescu, and M. Swift. Chester: Towards a personal medication advisor. *Journal of Biomedical Informatics*, 39(5):500–513, 2006.

J.F. Allen, D.K. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. Toward conversational human-computer interaction. *AI Magazine*, 22(4):27–38, 2001.

J. Bateman. The theoretical status of ontologies in natural language processing. In *Workshop on Text Representation and Domain Modelling - Ideas from Linguistics and AI*, 1991.

J. Bateman and M. Zoch. *Natural Language Generation.* The Oxford Handbook of Computational Linguistics, Ruslan Mitkov, Oxford University Press, 2005.

J. Bateman and M. Zock. *Natural Language Generation.* R. Mitkov, The Oxford Handbook of Computational Linguistics. Oxford University Press, 2003.

J. A. Bateman.

J. A. Bateman, R. T. Kasper, J. D. Moore, and R. A. Whitney.

J.A. Bateman, B. Magnini, and F. Rinaldi. The generalized Italian, German, English upper model. In *ECAI-94 Workshop on Implemented Ontologies*, 1994.

J.C. Carletta. Assessing the reliability of subjective codings. *Computational Linguistics*, 22(2): 249–254, 1996.

B. De Carolis, G. Cozzolongo, S. Pizzutilo, and V. Silvestri. Mymap: Generating personalized tourist descriptions. *Applied Inteligence*, (26):111–124, 2007.

J. Chu-Carrol and M. Brown. An evidential model for tracking initiative in collaborative dialogue interactions. *Applied Inteligence - User Modeling and User-Adapted Interaction*, (8):218–253, 1998.

M. Danieli and E. Gerbino. Metrics for evaluating dialogue strategies in a spoken language system. In *AAAI String Symposium on Empirical Methods in Doscourse Interpretation and Generation*, pages 34–39, 1995.

FADA. "`http://fada.sourceforge.net/`". Federated Autonomous Directory Architecture.

A. García-Serrano, J. Calle, and J. Hernández. Dialogue management for an advice-giving virtual assistant. In *IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2001.

M. Gatius. *Using an ontology for guiding NL interaction with knowledge based systems.* PhD thesis, Universitat Politècnica de Catalunya, 2001.

M. Gatius and M. González. Utilización de ontologías en el desarrollo de sistemas de diálogo. In *Proceedings of the III Jornadas en Tecnología del Habla*, pages –, 2004a.

M. Gatius and M. González. Ontology-driven voicexml dialogues generation. In *Proceedings of the Workshop at the Berliner XML-Tage*, pages –, 2004b.

M. Gatius and M. González. Using ontologies for improving the communication process in a dialogue system. In *Proceedings of the Sixth International Workshop on Computational Semantics*, pages –, 2005a.

M. Gatius and M. González. Un sistema de diálogo multilingüe dirigido por la semántica. *SEPLN*, (34):–, 2005b.

M. Gatius and M. González. Obtaining linguistic resources for dialogue systems from application specifications and domain ontologies. In *Proceedings of the 10th International Conference on Speech and Computer*, pages –, 2005c.

M. Gatius and M. González. Using application-specific ontologies to improve performance in a bottom-up parser. In *Proceedings of the Workshop Knowledge and Reasoning for Language Processing*, pages –, 2006a.

M. Gatius and M. González. Integrating semantic web and language technologies to improve the online public administrations services. In *Proceedings of the 15th International Worl Wide Web Conference*, pages –, 2006b.

M. Gatius and M. González. Discourse management in voice systems for accessing web services. In *Proceedings of the Workshop on the Semantics and Pragmatics of Dialogue*, pages –, 2007a.

M. Gatius and M. González. A multilingual dialogue system for accessing the web. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies*, pages –, 2007b.

M. Gatius and H. Rodríguez. Natural language guided dialogues for accessing the web. In *Text, Speech and DialoguePublished*, volume 2448, 2002.

M. Gatius and H. Rodríguez. A domain-restricted task-guided natural language interface generator. In *Second Edition Workshop on Flexible Query Answering Systems*, 1996.

M. Gatius, M. González, L. Lesmo, and P. Torasso. Natural language processing technologies. European Project HOPS (IST-2002-507967) Deliverable D3.2, 2005.

M. Gatius, M. González, and E. Comelles. An information state-based dialogue manager for making voice web smarter. In *Proceedings of the 16th International Worl Wide Web Conference*, pages –, 2007a.

M. Gatius, M. González, Eli Comelles, and Leonardo Lesmo. Natural language processing and dialogue management development. European Project HOPS (IST-2002-507967) Deliverable D4.3, 2007b.

M. González and M. Gatius. Un sistema de diálogo multicanal para acceder a la información y servicios de las administraciones públicas. In *XXI Congreso de la SEPLN, I Congreso Español de Informática*, pages –, 2005.

H.P. Grice. *Logic and Conversation*, volume 3. P. Cole and J. L. Morgan, Syntax and Semantics: Speech Acts. New York: Academic Press, 1975.

D. Griol, F. Torres, L.F. Hurtado, E. Sanchis, and E. Segarra. Different approaches to the dialogue management in the dihana project. In *SPECOM*, 2005.

D. Griol, F. Torres, L. Hurtado, S. Grau, F. García, E. Sanchís, and E. Segarra. A dialog system for the dihana project. In *Proceedings of the SPECOM*, 2006.

B.J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86 (2):269–357, 1996.

B.J. Grosz and C.L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

M. Hadjdinjak and F. Mihelic. The paradise evaluation framework: Issues and findings. *Computational Linguistics*, 2(32):265–272, 2006.

S.W. Hamerich, Y.F. Wang, V. Schubert, V. Schless, and S. Igel. Xml-based dialogue descriptions in the gemini project. In *Berliner XML-Tage*, 2003.

S.W. Hamerich, V. Schubert, V. Schless, R. de Córdoba, J.M. Pardo, L.F. dHaro, B. Kladis, O. Kocsis, and S. Igel. Semi-automatic generation of dialogue applications in the gemini project. In *SIGdial Workshop on Discourse and Dialogue*, pages 31–34, 2004.

M. Hartmann, D. Schreiber, and M. Kaiser. Task moldels for proactive web applications. In *WebIST*, 2007.

P. Hernández, J. Sánchez, A. López, S. Militello, M. Gatius, M. González, E. Comelles, L. Lesmo, X. Noria, R. Salla, C. de la Morena, J.A. Fernández, and A. Deiro. Hops architecture specifications. European Project HOPS (IST-2002-507967) Deliverable D4.1, 2006.

HOPS. "`http://www.hops-fp6.org`". FP6 Project IST-2002-507967.

Kristiina Jokinen, Antti Kerminen, Mauri Kaipainen, Tommi Jauhiainen, Graham Wilcock, Markku Turunen, Jaako Hakulinen, Jukka Kuusisto, and Krista Lagus. Adaptive dialogue systems - interaction with interact. In *SIGdial Workshop on Discourse and Dialogue*, pages 64–73.

D. Jurafsky, R. Bates, N. Coccaro, R. Martin, M. Meteer, K. Ries, E. Shriberg, A. Stolcke, P. Taylor, and C. Van Ess-Dykema. Automatic detection of discourse structure for speech recognition and understanding. In *IEEE Workshop on Speech Recognition and Understanding*, pages 88–95, 1997a.

D. Jurafsky, E. Shriberg, and D. Biasca. Switchboard-damsl labeling project coder's manual. Technical Report 97-02, University of Colorado, Institute of Cognitive Science, 1997b. URL `http://www.colorado.edu/ling/jurafsky/manual.august1.html`.

D. Jurafsky, R. Bates, N. Coccaro, R. Martin, M. Meteer, K. Ries, E. Shriberg, A. Stolcke, P. Taylor, and C. Van Ess-Dykema. Switchboard discourse language modeling. Project Report Research Note 30, Center for Speech and Language Processing, Johns Hopkins University, Baltimore, MD, 1998a.

D. Jurafsky, E. Shriberg, B. Fox, and T. Curl. Lexical, prosodic, and syntactic cues for dialog acts. In Manfred Stede, Leo Wanner, and Eduard Hovy, editors, *Discourse Relations and Discourse Markers: Proceedings of the Conference*, pages 114–120. Association for Computational Linguistics, 1998b.

L.F. Lamel, S.K. Bennacef, S. Rosset, L. Devillers, S. Foukia, J.J. Gangolf, and J.L. Gauvain. The limsi railtel system: Field trial of a telephone service for rail travel information. *SPEECH Communication*, (23):67–82, 1997.

S. Larsson. *Issue-based Dialogue Management*. PhD thesis, Göteborg University, Sweden, 2002.

R. San-Segundo J.M. Montero J. Macías-Guarasa J.M. Pardo L.F. dHaro, R. de Córdoba. Strategies to reduce design time in multimodal/multilingual dialog applications. In *Proceedings of the Interspeech*, 2004.

LIMSI. "`http://www.limsi.fr/tlp/railtel.html`". Spoken Language Processing Group.

D. Litman and S. Pan. Designing and evaluating an adaptive spoken dialogue system. *Applied Inteligence - User Modeling and User-Adapted Interaction*, (12):111–137, 2002.

D. Louloudis, N. Fakotakis, and G. Kokkinakis. Mathematical modeling of dialogue control strategies for interactive dialopgue systems. In *SPECOM*, pages 741–744, 2005.

W. Mann and C. Matthiessen. *Nigel: A Systemic Grammar for Text Generation*. R. Benson and J. Greaves, Systemic Perspectives on Discourse: Selected Papers from the 9th International Systemics Workshop. London: Ablex Publishers Company, 1985.

K. McKeown. Discourse strategies for generating natural-language text. *Artificial Intelligence*, (27):1–41, 1985.

D. Milward and M. Beveridge. Ontology-based dialogue systems. In *IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2003.

D. Milward and M. Beveridge. Ontologies and the structure of dialogue. In *CATALOG, 8th Workshop on the Semantics and Pragmatics of Dialogue*, 2004.

R. Porzel, N. Pfleger, S. Merten, M. Löckelt, I. Gurevich, R. Engel, and J. Alezandersson. More on less: Further applications of ontologies in multi-modal dialogue systems. In *IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2003.

C. Rich, C.L. Sidner, and N. Lesh. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15–25, 2001.

J.R. Searle. *Speech Acts: an essay of the philosophy of language*. Cambridge University Press, 1969.

S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. Galaxy-ii: A reference architecture for conversational system development. In *Proceedings of the ICSLP*, volume 3, pages 931–934, 1998.

S. Seneff, R. Lau, and J. Polifroni. Organization, communication and control in the galaxy-ii conversational system. In *Proceedings of the Eurospeech*, pages 1271–1274, 1999.

SLS. "`http://groups.csail.mit.edu/sls/sls-blue-noflash.shtml`". SLS Lab Web at MIT.

R.W. Smith, D.R. Hipp, and A.W. Biermann. An architecture for voice ds based on prolog-style theorem proving. *Computational Linguistics*, 21(3):281–320, 1995.

A. Stolcke, K. Ries, N. Coccaro, E. Shriberg, R. Bates, D. Jurafsky, P. Taylor, R. Martin, M. Meteer, and C. Van Ess-Dykema. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, 26(3):339–371, 2000.

C.A. Thompson, M.H. Göker, and P. Langley. A personalized system for conversational recommendations. *AI Research*, (21):393–428, 2004.

S.A. Thompson. *Rhetorical Structure Theory: description and construction of text structures*. G. Kempen, Natural Language Generation: Recent Advances in Artificial Intelligence, Phychology, and Linguistics. Dordrecht: Kluwer Academic Publishers, 1987.

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

TRINDI. "`http://www.ling.gu.se/projekt/trindi/`". Task-Oriented Instructional Dialogue.

URCS. "`http://www.cs.rochester.edu/research/cisd/`". Conversational Interaction and Spoken Dialogue Research Group, Department of Computer Science at University of Rochester.

L. Villarejo, N. Castell, and J. Hernando. Dialogue management in an automatic metereological information system. In *International Conference on Industrial and Engineering Applications on Artificial Intelligence and Expert Systems*, volume 2718, pages 495–504, 2003a.

L. Villarejo, J. Hernando, and N. Castell. Voicexml in a real automatic metereological information system. In *Proceedings of the Berlin XML Days*, 2003b.

L. Villarejo, J. Hernando, N. Castell, J. Padrell, and A. Abad. Architecture and dialogue design for a voice operated information system. *Applied Intelligence*, (24):253–261, 2006.

Multimodal Interaction Activity W3C, a. URL `http://www.w3.org/2002/mmi/`.

SISR W3C. Semantic interpretation for speech recognition (sisr) version 1.0, 2007. URL `http://www.w3.org/TR/semantic-interpretation/`.

SRGS W3C. Speech recognition grammar specification version 1.0, 2004. URL `http://www.w3.org/TR/speech-grammar/`.

Voice Browser Activity W3C, b. URL `http://www.w3.org/Voice/`.

VoiceXML W3C. Voice extensible markup language (voicexml) 2.1, 2006. URL `http://www.w3.org/TR/voicexml21/`.

W. Wahlster. Smartkom: Symmetric multimodality in an adaptive and reusable dialogue shell. In *Proceedings of the Human Computer Interaction Status Conference*, 2003.

M.A. Walker, D.J. Litman, C.A. Kamm, and A. Abella. Paradise: A framework for evaluating spoken dialogue agents. In *ACL*, pages 271–280, 1997.

K. Walton. *Commitment in Dialogue: Basic concepts of Interpersonal Reasoning.* Suny Series in Logic and Language, State University of New York Press, 1995.

G. Wilcock and K. Jokinen. Generating responses and explanations from rdf/xml and daml+oil. In *IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2003.

W. Winiwarter. *Recent Developments in Human Language Technology*, volume 164. R. R. Wagner (ed). A Min Tjoa @ Work, books@ocg.at, 2002.

W. Xu, B. Xu, T. Huang, and H. Xia. Bridging the gap between dialogue management and dialogue models. In *SIGdial Workshop on Discourse and Dialogue*, volume 2, pages 201–210. Association for Computational Linguistics, 2002.

Victor Zue. Conversational interfaces: Advances and challenges. In *Proc. Eurospeech '97*, pages 9–18, Rhodes, Greece, 1997.