

---

DIGUI: A Flexible Dialogue System for  
Guiding the User Interaction to  
Access Web Services

---

*PhD Thesis by*

Meritxell González Bermúdez

*Advisor*

Marta Gatius Vila

Programa de Doctorat en Intel·ligència Artificial  
Departament de Llenguatges i Sistemes Informàtics



**UNIVERSITAT POLITÈCNICA  
DE CATALUNYA  
BARCELONATECH**

*Barcelona, October 2010*

This PhD thesis was publically defended in Barcelona on October 22nd, 2010.

This PhD dissertation obtained the *European mention*.

In the public defense, it was my honor to have the following PhD Jury:

1. Dr. Horacio Rodríguez, Universitat Politècnica de Catalunya, Spain
2. Dra. Núria Castell, Universitat Politècnica de Catalunya, Spain
3. Prof. Dr. Emilio Sanchís, Universidad Politécnica de Valencia, Spain
4. Prof. Leonardo Lesmo, Università di Torino, Italy
5. Dr. Sebastian Vargas, Università degli Studi di Trento, Italy

To whom I am sincerely grateful.

Meritxell Gonzàlez Bermúdez

*Als meus pares,  
que em van ensenyar a caminar.*

*A en Kizito,  
que m'acompanya en el camí.*

*”Caminante, son tus huellas  
el camino nada más;  
caminante, no hay camino,  
se hace camino al andar.”*

Antonio Machado

## Agraïments/Acknowledgments

Aquesta tesi és el resultat de l'esforç de moltes persones sense les quals no hagués estat possible. Algunes persones m'han ensenyat el camí, altres m'hi han abocat o m'han acompanyat, i uns altres m'han donat l'empenta que necessitava en moments en què em sentia defallir.

En primer lloc vull expressar el meu sincer agraïment a la Marta Gatius, que ha dirigit aquesta tesi, per tota la feina que hem fet plegades, les correccions, els comentaris i els savis consells que sempre m'ha donat.

Vull dedicar unes línies d'aquests agraïments a tot l'equip del projecte HOPS, dels qui vaig aprendre tant, i en especial a en Jordi, el Jose Antonio i l'Àngel, amb els que vaig compartir interminables línies de codi, i l'Eli, per fer que el NLG es fes realitat.

També vull donar les gràcies a tots el que són o han sigut en algun moment els meus companys a l'Omega, per compartir i en cert sentit patir aquest llarg viatge, i molt especialment a la Cristina, l'Edgar, l'Emili, l'Ignasi, el Jesús, el JordiP, la Leo, la Muntsa, el Pere, el Roberto i el XaviL, sense els quals els cafès i el berenars no serien el mateix, a la Maria, per les seves sempre enriquidores converses, a la Montse, per compartir amb mi l'organització dels seminaris i la jornada '*PhD::Think In*', i a l'FSM, el Lobo i tots els membres de la *freakexcursió*, pels caps de setmana inoblidables.

I'm also very grateful to Prof. Giuseppe Riccardi, for hosting me at Università di Trento, and to the members of the LSI Lab, for welcoming me into their research team where I met great colleagues from whom I learnt so much. I especially thank Zsuzsa for making my stay immensely enjoyable. I feel lucky to have met good friends like Pierre, Silvia and Stefan, and have surfed the Alps together. And a special mention goes to Arindam and Fernando, for the great moments inside and outside the lab, sharing a football match or a hike in the mountains.

Finalment, vull expressar el més profund agraïment als meus pares, Tomàs i Rosa Mari, per l'amor infinit que doneu als vostres fills, així com als meus germans Óscar i Edu, per l'empenta que sempre he rebut de vosaltres, i a la Judit, la Rut, el Juan i tota la meva família pel vostre suport incondicional. Mai trobaré les paraules que expressin el meu etern agraïment a en Kizito, per la seva silenciosa paciència i inestimable companyia. Gràcies per recórrer el camí amb mi.

A tots, gràcies per creure en mi.

**Funding.**

This research has been funded by the European IST FP6 project HOPS (IST-2002-507967) and the Spanish program “Acción Complementaria de la Comisión Interministerial de Ciencia y Tecnología (CICYT)” TIN2004-0169-E.

The research visit at University of Trento, to obtain the *European mention*, has been funded by the Spanish Ministry of Education program “Ayudas para la Movilidad de Estudiantes para la Obtención de la Mención Europea en el título de Doctor para el curso 2009-2010” and the Marie Curie Action project ADAMACH (contract No. 022593).

## Resum

Els sistemes de diàleg permeten establir una conversa agradable, en la que es poden donar diferents tipus d'interaccions, com ara menús amb els que es presenta a l'usuari un conjunt d'opcions, comandes que expressen ordres, preguntes molt concretes i també preguntes complexes que poden fer referència a entitats prèviament evocades. Precisament perquè els sistemes de diàleg permeten diversos tipus d'interaccions, són adequats per facilitar l'accés de l'usuari a diferent tipus d'aplicacions.

La majoria dels sistemes de diàleg s'adapten a una aplicació concreta, ja que el seu funcionament millora quan es limita la cobertura del llenguatge per a un conjunt de tasques determinades. El problema dels sistemes de diàleg desenvolupats per a una aplicació concreta és que no són fàcilment adaptables a d'altres aplicacions, i requereixen la intervenció d'experts. Per a superar aquesta limitació s'han realitzats molts treballs de recerca centrats en el desenvolupament de sistemes de diàleg fàcilment adaptables a diverses aplicacions i llengües. Els treballs més rellevants en aquesta línia proposen l'ús de models de diàleg i una arquitectura modular amb components reutilitzables per a diferents aplicacions.

El treball d'aquesta tesi es centra en els sistemes de diàleg per a guiar als usuaris quan accedeixen a serveis i continguts disponibles a la web. Hem desenvolupat DIGUI: un sistema de diàleg multilingüe, fàcilment adaptable a noves aplicacions i d'iniciativa mixta que s'adapta a diferents tipus d'interacció. Per a facilitar l'adaptació del sistema a nou serveis i noves llengües, les diferents tasques del sistema són realitzades per quatre mòduls independents, que controlen les quatre tasques principals (processament del llenguatge, control de la interacció, accés a l'aplicació i generació de respostes), i usen dues estructures de dades (context del diàleg i descripció del domini) accessibles des de qualsevol dels mòduls. Amb la finalitat de facilitar la incorporació de noves aplicacions proposem la utilització d'estructures de dades generals per a representar el coneixement conceptual de l'aplicació i el seu domini, així com el coneixement lingüístic.

DIGUI també incorpora un mòdul independent que adapta les estratègies d'iniciativa del diàleg considerant els possibles problemes que apareixen durant la conversa. Aquest component utilitza un model d'iniciativa, basat en diferents dades relacionades amb la comunicació, per a determinar el nivell d'iniciativa que ha d'adoptar el sistema en la següent intervenció.

## Summary

Current dialogue systems can handle friendly and collaborative communication that supports diverse types of interaction, such as menus in which the user is asked to choose an option, form filling in which the user is asked for specific information, commands for expressing users' orders, and complex questions that can even reference previously evoked entities. Consequently, dialogue systems seem to be useful for accessing diverse types of applications.

Practical dialogue systems are mostly adapted to the functionality of the specific application they access. Because the adaptation of those DS to new applications is expensive and has to be done by experts, many studies have focused on the problem of developing dialogue systems that can be used for several applications and languages. Most of these systems use dialogue models and have reusable domain-independent components.

This thesis is concerned with dialogue systems for guiding the user to access web services. We have developed DIGUI: a multilingual mixed-initiative dialogue system that facilitates the adaptation of the system to new services and languages. The dialogue system developed consist of four independent general modules (language understanding, language generator, dialogue manager and task manager) as well as two knowledge structure (dialogue context and domain description) accessible by all the modules. In order to minimize the need for application and language-restricted data we propose the use of separated general bases for representing linguistic, application and domain knowledge.

DIGUI also incorporates an independent component to adapt the dialogue initiative according to how well the communication is progressing. This component uses a conversation model, which considers several features related to the progress of the conversation, to determine the system initiative in the next intervention.

## Acronyms

A	Adaptive initiative
ActualTD	Actual Task Data
BDI	Beliefs, Desires and Intentions
CA	Cultural Agenda
CK	Conceptual Knowledge
DA	Dialogue Act
DM	Dialogue Manager
DS	Dialogue System
GP	Green Point
IS	Information State
ISU	Information State Approach
LOC	Large Objects Collection
NLG	Natural Language Generation
NLU	Natural Language Understanding
POMDP	Partially Observable Markov Decision Process
RST	Rhetorical Rtructure Theory
TM	Task Manager
TSR	Task Success Rate

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Objective . . . . .	14
1.2	Overview . . . . .	15
<b>2</b>	<b>Brief introduction to the state of the art</b>	<b>17</b>
2.1	Functionalities of dialogue systems . . . . .	17
2.2	Dialogue management and dialogue models . . . . .	20
2.3	Dialogue and task initiative . . . . .	24
2.4	Task management for dialogue systems . . . . .	25
2.5	Generation of the system's messages . . . . .	27
2.6	Evaluation of dialogue systems . . . . .	29
2.7	Overview of related studies . . . . .	30
<b>3</b>	<b>Overview of DIGUI</b>	<b>35</b>
3.1	Dialogue system components . . . . .	36
3.2	Interaction flow of the dialogue system . . . . .	39
3.3	The web services typology . . . . .	42
<b>4</b>	<b>Dialogue management</b>	<b>45</b>
4.1	Dialogue manager components . . . . .	45
4.2	The dialogue model . . . . .	48
4.2.1	The set of dialogue moves . . . . .	48
4.2.2	The information state . . . . .	51
4.2.3	Rules for updating the information state . . . . .	56

4.2.4	The control algorithm . . . . .	57
4.3	The communication plans . . . . .	58
4.4	The adaptive module . . . . .	60
4.4.1	Obtaining cues . . . . .	61
4.4.2	Adapting the degree of initiative . . . . .	63
4.5	Summary . . . . .	65
<b>5</b>	<b>Task management</b>	<b>67</b>
5.1	Generalization of the service structure . . . . .	67
5.1.1	Service plans . . . . .	68
5.1.2	Transactional service: general schema . . . . .	68
5.1.3	Informative services: general schema . . . . .	71
5.2	The task manager . . . . .	76
5.2.1	Task representation . . . . .	76
5.2.2	The processes involved in task management . . . . .	78
5.2.3	The algorithms that process the tasks . . . . .	83
5.3	Summary . . . . .	88
<b>6</b>	<b>Natural language modules</b>	<b>89</b>
6.1	Obtaining linguistic resources . . . . .	90
6.2	Natural language understanding . . . . .	91
6.2.1	The analyzer . . . . .	94
6.2.2	The interpreter . . . . .	96
6.3	Natural language generation . . . . .	98
6.3.1	The patterns . . . . .	99
6.4	Summary . . . . .	100
<b>7</b>	<b>Implementation and evaluation</b>	<b>105</b>
7.1	Implementation . . . . .	105
7.2	The evaluation . . . . .	108
7.2.1	Methodology . . . . .	109
7.2.2	A comparison of the linguistic resources: first vs. second experiment . .	112
7.2.3	A comparison of the initiative strategy . . . . .	113

<i>CONTENTS</i>	11
7.2.4 A comparison between transactional and informative services . . . . .	114
7.2.5 Additional discussion . . . . .	116
<b>8 Conclusions</b>	<b>119</b>
8.1 Further studies . . . . .	122
<b>A Examples</b>	<b>125</b>
A.1 Large Objects Collection Example . . . . .	125
A.2 Cultural Agenda Example 1 . . . . .	130
A.3 Cultural Agenda Example 2 . . . . .	132
<b>B Analysis of the structure of the tasks</b>	<b>137</b>
B.1 The transactional service LOC . . . . .	137
B.1.1 The task “Collection” . . . . .	141
B.1.2 The task “Obtain_GP” . . . . .	146
B.1.3 The task “Cancel_Collection” . . . . .	147
B.2 The informative service (CA) . . . . .	148
B.2.1 The task “Obtain_Location” . . . . .	152
B.2.2 The task “Obtain_Event” . . . . .	154
B.2.3 The task “FindList_CA” . . . . .	157
<b>C Resources</b>	<b>163</b>
C.1 The task schemas . . . . .	164
C.1.1 Schemas for the LOC service . . . . .	166
C.1.2 Schemas for the CA service . . . . .	168
C.2 The domain knowledge . . . . .	170
C.3 The language patterns . . . . .	174
C.4 Acquisition of the resources . . . . .	175
C.4.1 Creating and adapting the resources to a new service . . . . .	176
<b>D Related publications</b>	<b>179</b>
<b>Bibliography</b>	<b>193</b>



# Introduction

---

Current dialogue systems can handle friendly and collaborative communication that supports diverse types of interactions, such as menus in which the user is asked to choose an option, form filling in which the user is asked for specific information, commands for expressing users' orders and complex questions that can even reference previously evoked entities. Consequently, they can be used to access different types of applications. In particular, spoken dialogue systems are appropriate for devices that do not allow web browsing, such as telephones, or the use of hands, such as vehicle GPS.

Practical spoken systems are mostly adapted to the functionality of the specific application they access, since voice technology still presents recognition problems in open domains. The cost of developing application-restricted systems is high, as experts are needed. Moreover, these systems are difficult to adapt to new applications.

Consequently, many studies have focused on the problem of developing dialogue systems that can be used for different applications and languages. Some of them 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, especially in mixed-initiative systems, in which the dialogue initiative can be taken by the user of the system.

The present thesis is particularly concerned with dialogue systems for guiding the user to access web services. Dialogue systems can improve the usability and accessibility of web contents, e.g. virtual assistants. Web interfaces need to be more interactive and adaptable

to different types of users, services and channels since the need to enhance web usability is increasing. Several studies have already been undertaken in this line, on topics such as web voice interfaces and collaborative agents that help the user to interact with the web service. VoiceXML is a well-known standard formalism that is used in many spoken dialogue systems. The VoiceXML standard is appropriate for defining the flow of system-initiative dialogues, in which the system asks the users to provide the specific information the service needs. However, VoiceXML systems support only very limited user-initiative (the user can only choose the order in which to give the information requested by the system) and do not support complex dialogue phenomena, such as clarification. Furthermore, the dialogue flow in VoiceXML systems has to be defined for each service.

## 1.1 Objective

This thesis is concerned with the development of multi-lingual and multi-domain dialogue systems. In particular, we focus on the design of built-in flexible architecture to improve communication and reduce the cost of adapting the system to other services. We propose a modular dialogue system that uses general components as well as domain and language-dependent resources. The components of the dialogue system have been developed using well-known innovative technologies. Hence, our main contribution is the integration of these components to build a flexible and adaptive multilingual dialogue system.

To facilitate the adaptation of the dialogue system to new services and languages and to achieve friendly communication, we propose the use of a rich dialogue model and the incorporation of an independent component to adapt the dialogue strategies according to how well the communication is progressing. DIGUI, the dialogue system developed, consists of four independent general modules: language processing, dialogue management, task management and language generation.

Dialogue management is concerned mainly with conversational control and guidance, i.e. turn-taking control, keeping track of the dialogue state and deciding on the system's next actions. Most commercial dialogue systems, whose main goal is robustness, only support system-initiative dialogues that ask the user for the information needed to accomplish a task.

In such approaches, the dialogue manager is a simple state automaton that represents all possible interactions. A specific dialogue flow is used for each service, in which there is an explicit description of the relation between the states and the participant's actions. The separation of the types of knowledge involved in communication improves both the communication and the portability of the dialogue system. More complex dialogue managers improve the communication and the engineering process of adapting the system to new services by using explicit dialogue models separated from the specific task models that represent the application knowledge. In recent research on dialogue systems, dialogue managers use complex models and plans to address the user's goals. The main drawback of such models and plans is their complexity. To reduce this complexity, we study the structure of different types of tasks that appear in the web services considered in this thesis, and the communication actions and phenomena that occur in the dialogues collected when users accessed these services. We propose the use of general schemas for different web services to facilitate dialogue management, task management and the integration of new services.

We also propose the use of general linguistic bases to generate the dialogue system's responses. The system has to provide natural and efficient communication messages. Complex strategies and mechanisms improve communication. However, they are futile when the system cannot express them. Thus, the language generator module may be able to produce messages that express a wide variety of information, such as feedback, acknowledgment or clarification. In our system, language-independent knowledge and components are separated from language-dependent ones. This separation makes multilingualism possible and facilitates the adoption of additional languages.

## 1.2 Overview

This document is organized as follows: Chapter 2 gives a summary review of the state of the art in dialogue systems, dialogue management, task management, natural language generation and evaluation. Chapter 3 gives an overview of DIGUI, the dialogue system we have developed. Chapter 4 presents our approach to dialogue management and the adaptive initiative. Chapter 5 addresses the representation and processes related to the service tasks. It describes the generalized schemas for guiding the user to access web services. This approach is based on

the analysis of the dialogue structures described in Appendix B. Chapter 6 describes the two modules related to natural language processing. Chapters 7 and 8 describe the implementation, the experiments and the general conclusions. Appendix A contains several examples of the main system actions and the data involved when the system guides the user to access the services. Appendix C describes the design and implementation of the resources used to develop DIGUI. Finally, Appendix D lists the publications that this thesis has produced so far.

# Brief introduction to the state of the art in dialogue systems

---

This chapter introduces dialogue systems. It focuses mainly on the topics addressed in this thesis. First, we describe the main functionalities and modules of the dialogue systems. Then, we discuss well-known dialogue systems and their main characteristics.

## 2.1 Functionalities of dialogue systems

Dialogue systems (henceforth, DS) have improved in both the functionality and the engineering features of their development process. The current state is due to several lines of research. The most significant of these are the separation of the application and communication knowledge and the adaptation of the communication process to applications and users.

DSs are a specific type of conversational systems (Figure 2.1 shows a generic diagram for them). Other type of conversational systems are personalized recommendation systems (e.g. [Thompson et al., 2004]), collaborative agents (e.g. COLLAGEN, [Rich et al., 2001]), or argumentation/persuasion (an emerging topic on human-computer interaction), e.g. [Andrews, 2008]. In all conversational systems the component that controls the interaction depends on the content and representation of the discourse context and the conceptual representation of the information that appears in the conversation. Similarly, the process that generates the system's messages depends on the representation of the discourse context and the interaction manager. The general system behaviour depends on the degree of collaboration among the modules.

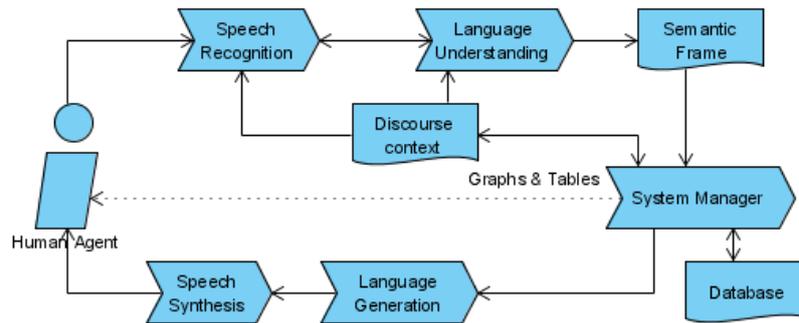


Figure 2.1: Generic diagram for conversational systems

The functionalities of dialogue systems are well defined in [Smith et al., 1995]: an earlier study that focused on improving the required functions for an efficient DS. Specifically, this system includes the following functionalities:

1. The system has to achieve the user's goal. That is, the system has to provide the information that the user requested.
2. The system has to manage the tasks that arise during the conversation. The dialogue participants may initiate subtasks that help to solve the main goal. These subtasks can be seen as sub-dialogues.
3. The information that appears during the communication has to be well-structured. The data that appear during the conversation are the common knowledge that the participants share, and the process that manages the interaction depends mainly on the representation of the conversation data.
4. The system may support varying degrees of initiative for different users. While novel users may need the system to guide the dialogue (system-initiative), experienced ones may prefer to take the initiative and guide the dialogue to the goal (user-initiative), and even to use shortcuts. Most research systems allow mixed-initiative interaction, although most commercial systems only use system-initiative strategies. Few systems include mechanisms for adapting the degree of initiative to the user.
5. The system has to predict the user's answer. Speech recognition is still a hard task. Advanced systems that use structurally stored information can take advantage of the dialogue focus and can assist the recognition processes. The DSs that are adapted to the communication needs of the application improve their performance because mistakes and ambiguities are reduced. However, the more directed the recognition of the user's

intervention, the fewer degrees of freedom the user has in his/her turn. In addition, application-restricted DSs are expensive to develop and difficult to reuse.

The more specific problem of developing multilingual DS for different applications has also been the focus of many research studies. Most of these propose a flexible and modular architecture and separate the representation of the different types of knowledge involved in the communication: application knowledge, dialogue knowledge and modality and language-specific knowledge. By isolating the representation of the conceptual knowledge involved in a specific domain or application, the DS constituents can be reused. When knowledge is formal and explicitly organized, its clarity, consistency and accessibility are improved. We describe our approach for enhancing usability and accessibility in [Gatius and González, 2007b].

The voice browser<sup>1</sup> and multi-modal interaction<sup>2</sup> activity groups of W3C define standard languages and architectures that facilitate the development of DSs to access web contents. A well-known language is VoiceXML<sup>3</sup>, the XML-based open standard that is explicitly dedicated to the development of Internet-powered telephone applications. Many commercial systems are implemented using VoiceXML-based platforms. In addition, many research studies are based on Voice-XML, e.g. the aTTempS system [Villarejo et al., 2003b] developed at TALP and the HOPS project<sup>4</sup> [Gatius and González, 2004b], which promoted this thesis. One of the limitations of VoiceXML is that it does not support the definition of complex interactions for sophisticated applications. Large-scale DSs that support rich interactions with different applications overcome the limitations of Voice-XML by using specific modules for dialogue and task management and general dialogue and task models (even when they use VoiceXML for speech recognition). DSs that use several languages and channels still constitute a challenge.

The following sections provide an overview of the DS constituents addressed in this thesis: dialogue and task management, the generation of linguistic resources and the evaluation of dialogue systems.

---

<sup>1</sup><http://www.w3.org/Voice/>

<sup>2</sup><http://www.w3.org/2002/mmi/>

<sup>3</sup><http://www.w3.org/TR/voicexml21/>

<sup>4</sup><http://www.bcn.es/hops/>

## 2.2 Dialogue management and dialogue models

A dialogue is a particular kind of discourse. The distinguishing characteristics of dialogues are: turns are taken, participants have common ground knowledge and some phenomena occur that are not present in other types of discourse, such as pauses, errors, confirmations and false beginnings.

Dialogue management techniques deal with the use of dialogue to meet the user's objectives. The dialogue management process controls the conversation and guides the user. It also determines the allowed type of interaction, which is related to the underlying task, the domain and the user.

The classification of dialogue managers has been the focus of many studies. For instance, in the system TRIPS described in [Allen et al., 2001a], the dialogue managers are classified depending on the technique that is used and the task complexity. The simplest approaches are those that use finite state scripts (e.g. ATTEMPS, [Villarejo et al., 2003a]). In terms of task complexity, these are followed by frame-based systems, in which users ask questions and the system performs simple clarifications (e.g. ADAPTIVE PLACE ADVISOR, [Thompson et al., 2004]). Finally, plan- and agent-based approaches are used for the most complex tasks (e.g. ADVICE SYSTEM, [García-Serrano et al., 2001]).

We can also differentiate between two types of dialogues, depending on how the dialogue state is represented: structural dialogue state and plan-based. In structural approaches, each dialogue state represents the results of an action performed in a previous state. These approaches, which include finite state models, are still used in many spoken systems. For example, VoiceXML, the standard language widely adopted to provide telephone access to web content, follows a finite state model. The main advantage of VoiceXML over previous commercial systems that use the finite state model is that it facilitates the description of the slots representing the data that the user is asked to fill. Using this description, the system can deal with a certain type of mixed-initiative dialogues: he/she can choose the order in which information about the requested slots is given. However, VoiceXML cannot deal with user-initiative dialogues, and the dialogue flow must be defined for each new service.

Many other spoken systems use a structural dialogue state model based on frames (or templates). This approach is more flexible than the finite state model. The frames contain slots that describe the various kinds of information the user will be asked to provide. The number of frames required depends on the complexity of the application tasks. Frame-based models are appropriate when the system can only carry out a small set of actions (as the number of frames is limited), which is not the case in complex applications.

In contrast, plan-based dialogue management theories support collaborative communication, but are difficult to implement in practical DSs. Plan-based dialogue models consider utterances as communication actions that are part of a plan. In these approaches, general planners and plan recognizers are used to produce cooperative and sophisticated behaviour. In most dialogue models, each communication action is considered as one or more dialogue acts (DA). Many studies propose the set of DAs that are needed to describe the parts of an interaction. A well-known set was described in [Jurafsky et al., 1997, 1998]. In this study, 42 different clusters were defined for DAs. The most significant were backchannels, statements, questions, exits, answers and confirmations.

New theories have been defined to combine the main advantages of structural and plan-based approaches, namely efficiency and flexibility. For instance, the information state update theory (ISU, [Traum et al., 1999]) formalizes the type of information that the dialogue participants consider and how this information is updated during the communication.

The literature includes hand-crafted dialogue strategies, e.g. [Larsson, 2002], that are based on a set of rules for managing the dialogue state as well as mathematical models for controlling the dialogue, as in [Louloudis et al., 2005] and [Young et al., 2010]. In the latter, the ISU approach is integrated into a POMDP<sup>5</sup>-based framework for dialogue management.

The type of dialogue management may also depend on how task and dialogue models are used. The following is a classification proposed by [Xu et al., 2002]:

- Implicit dialogue model. The dialogue flow is defined for each service together with the task model.
  - Implicit task model. Examples are finite-state based approaches.

---

<sup>5</sup>Partially Observable Markov Decision Processes

- Explicit task model. This is usually a frame-based approach.
- Explicit dialogue model. This model considers domain- and application-independent management strategies.
  - Implicit task model. This approach has no practical development.
  - Explicit task model. This approach has the same advantages as frame-based systems and is more natural. Examples include plan-based approaches.

### **Implicit dialogue models**

The main problem in structural state dialogues (such as finite state and frame-based models) is that they only support very limited user-initiative. A specific dialogue flow has to be defined for each application. Specifically, an explicit description of the relation between states and actions must be given. The information is implicit in the state of the dialogue and each state defines the next states that are allowed. For instance, [Hartmann et al., 2007] describes a prototype that displays support information online and is adapted to the user's needs and preferences. The DM is focused on web interaction management in particular. The task model is composed of the application's interaction elements and is represented as a directed graph in which nodes are states and activities.

### **Explicit dialogue models**

More flexible communication is provided by including explicit dialogue and task models. Several complex DSs use explicit dialogue and task models to support rich interactions with different applications in several languages and channels. Most of these systems use plans to manage the communication.

In the aforementioned ISU approach, the information state (IS) represents not only the general dialogue information, but also the internal state of each participant, such as their beliefs (knowledge), desires (goals) and intentions (plans). One of the advantages of the ISU theory, compared to other dialogue management theories based on beliefs, goals and intentions, is that the IS proposes a clear formalization of the knowledge involved in the dialogue context.

TRINDI<sup>6</sup> is one of the most distinguished projects among the large number of implemen-

---

<sup>6</sup><http://www.ling.gu.se/projekt/trindi/>

tations of the ISU approach (e.g. [Quesada, 2002], [Milward and Beveridge, 2004]). One of the main features of the TRINDI proposal in [Traum et al., 1999] and the IS implemented within TRINDI is the dialogue model. In its design, the information that is supposed to be known by the two participants is separated from the information that only the system knows. The information that is only known by the system is the private information, whereas the shared part of the IS corresponds to the information known by both agents.

Staffan Larsson, who participated in the TRINDI project, developed the IBIS issue-based systems. In [Larsson, 2002] the dialogues are represented using plans and the dialogue model follows the TrindiKit system, which is based on Ginzsburg’s dialogue gameboard.

In the European project TALK<sup>7</sup>, the IS was extended to tackle multi-modal input and output. It was also expanded to address targets of the specific applications integrated in the DS. The authors described several applications for which they had developed extensions for the IS: multi-modal discourse context, extended discourse history for addressing multi-modal input interpretation and output realization, and extended task representation for planning processes. A remarkable outcome of the TALK project, in relation to the system presented in this thesis, is the discussion about the task and the dialogue model in [Blaylock and Allen, 2005]. The dialogue is based on a collaborative problem-solving model that represents communicative intentions at the utterance level. In contrast, our model for task management represents communicative intentions at the task level.

### **Probabilistic dialogue models**

The research community is gaining interest in probabilistic dialogue models, due to the development of reinforcement learning-based DSs. In particular, DMs have been developed that are based on POMDP (e.g. [Williams and Young, 2007], [Young et al., 2010], [Varges et al., 2009a]). User simulators that can generate synthetic data to train these systems are also popular in the research community (e.g. [Schatzmann et al., 2006], [Quarteroni et al., 2010]). However, this thesis does not consider the use of stochastic models.

Previous (and current) probabilistic systems need to implement rule-based DSs to obtain human-computer dialogues for a specific application domain and to (manually) annotate the corpus to train the stochastic modules (e.g. [Griol et al., 2006]). For instance, in the DIHANA

---

<sup>7</sup><http://www.talk-project.org>

project, three DMs were developed in the domain of railway timetables and fares, as described in [Griol et al., 2005]. The data for the stochastic-based DM were acquired using the Wizard of Oz technique. Data were labelled and processed using frames of dialogue acts, concepts and values. Finally, the stochastic-based DM was trained using the information from different annotation levels and two models.

## 2.3 Dialogue and task initiative

As mentioned in the functionalities of the dialogue systems, the system should support varying degrees of initiative for different users. Novel users generally need the system to guide the interaction (system-initiative). In contrast, expert users prefer to take the initiative and to reach the goals in shorter ways (user-initiative).

The dialogue participant that takes the (task) initiative dictates which decomposition of the goal will be used by both participants during the problem-solving task. This definition was taken from [Guinn, 1998], in which the collaborative dialogues for problem-solving tasks are studied. Nevertheless, the definition is also valid for the types of dialogues we study in this thesis, in which the participant who takes the initiative establishes how the plan will develop, i.e. how the goal should be accomplished.

In Guinn's study, the initiative changes back and forth between the two participants depending on which goals they are working on. In DIGUI, the initiative changes between the system and the user depending on the progress of the communication, i.e. the number or recognition errors, misunderstandings and/or task parameters. This switching of initiative between participants is called mixed-initiative.

Guinn developed a model for changing the initiative from one participant to the other. Such a model is based on task completion efficiency and uses a probabilistic model to track an agent's ability to solve the task.

The model for tracking the initiative of the dialogue in DIGUI is based on [Chu-Carroll and Brown, 1998]. In this thesis, we have implemented this model in the dialogue system. The potential cues that trigger initiative shifts are investigated and the model that we developed tracks the distribution of dialogue- and task-initiatives between participants, based on the combined effect of a set of observed cues.

In [Chu-Carroll and Brown, 1998] the dialogue- and task-initiative can shift between the participants who interact collaboratively. For example, given a goal raised by a participant, three different types of answers are considered: the question is just answered and there is no dialogue- or task-initiative shift; information about the invalidity of the plan is given and the dialogue-initiative is shifted; and not only is information given about the invalidity of the plan, but also an alternative solution is proposed, and both dialogue- and task-initiative are shifted.

The evaluation model developed for the tracking initiative is based on Dempster-Shafer theory. This theory uses basic probabilities to represent the effect of a piece of evidence on a set of possible outcomes. Basic probabilities were assigned that represent the effect of each cue on the initiative shift and the system performance was evaluated based on its accuracy in predicting the dialogue- and task-initiative holders for each dialogue turn. When a cue was observed in a turn, this observation provided evidence of the value of the dialogue- and task-initiative in the next turn. Each cue was associated with two basic probabilities: the evidence that dialogue- and task-initiatives provide. Then, a training algorithm determined the basic probability functions, to represent the evidence of each cue.

## 2.4 Task management for dialogue systems

Plan-based DSs can be viewed as BDI (beliefs, desires and intentions) models in which the user plan consists of obtaining some information from the system or ordering actions to be performed by the system. The system uses rules to guess which plan the user wants to use. DSs often apply implicit and simple models to represent the tasks that are accessed. However, in plan-based DSs, the system needs well-designed models to represent the tasks. In many DSs, each user's goal is related to a specific task in the service, so that the models that represent the tasks have to include a schema for each potential goal.

Separation of the dialogue knowledge (in the dialogue manager) from the task knowledge (in the task manager) was first described in [Allen et al., 2001b]. However, most studies on DSs are focussed on developing and enhancing the dialogue models (and plans) for a specific application. Consequently, plans that represent the tasks are commonly hand-crafted for each specific system, and rarely flexible or generalizable.

Nevertheless, some studies examine the most generalizable representation of the application information. This approach helps in the planning of the system actions and improves the

system's response, which results in better communication with the user (e.g. [Steedman and Petrick, 2007], [Rieser and Lemon, 2009], [Varges et al., 2009b]). For instance, in the PKS SYSTEM, described in [Steedman and Petrick, 2007], the task and dialogue models are represented using a first-order logic language. In addition, the system applies general AI reasoning and planning methods to plan the dialogue actions. Scaling to a realistic domain is a main drawback of this methodology, as [Steedman and Petrick, 2007] pointed out in the conclusions.

However, there are many studies on automatic extraction and semantic learning of task structures from web services. Most of these build wrappers for decoupling the complexity of accessing heterogeneous web applications and for extracting information and relevant data from semi-structured sources (e.g. [Muslea et al., 2001]). We consider that the integration of this type of system in a DS could be very advantageous. Specifically, it could be useful for accessing several different web applications in the same domain at the same time, e.g. to access the cultural events listed in different sources or the train timetables of different companies. In addition, it can be useful for integrating the domain knowledge into the different knowledge resources of the system. The present study could be extended in the future to take advantage of this powerful enhancement. A related line of research is the automatic generation of dialogues from web services, which has been a popular research topic for years. The following are relevant studies on learning the task model from the application: SALT, [Wang, 2002]; SMARTWEB, [Sonntag et al., 2007]; and WEBTALK, [Feng et al., 2005]. PLOW, described in [Allen et al., 2007], is a different approach that learns the task model from the users of the system (who act as teachers). The integration of this kind of mechanism was outside the scope of this thesis.

Another interesting line of research related to our study consists of the use of ontologies to represent the different types of knowledge we use. During the preliminary stages of developing the system, we studied the use of ontologies to facilitate the representation of the knowledge involved in the DS (e.g. [González and Gatus, 2005], [Gatus and González, 2006b]). This approach has proven efficient in previous studies, such as the dialogue system described in [Gatus and Rodríguez, 2002]. In this study the concepts and the communication acts are

1: Quantity
<i>a) Make your contribution as informative as required.</i> <i>b) Do not make your contribution more informative than required.</i>
2: Quality
<i>c) Do not say what you believe to be false.</i> <i>d) Do not say anything for which you lack adequate evidence.</i>
3: Relation
<i>e) Be relevant.</i>
4: Manner
<i>f) Avoid obscurity of expression.</i> <i>g) Avoid ambiguity.</i> <i>h) Be brief.</i> <i>i) Be orderly.</i>

Table 2.1: Grice's conversational implicatures

represented in the conceptual ontology and the domain-restricted grammars and lexicons are obtained automatically. In DIGUI we also studied how to automatically generate the dialogue plans and the linguistic resources. These studies demonstrated that the approach was useful for dealing with multilingualism, for reusing resources and for improving the performance of the natural language modules ([Gatius and González, 2006a], [Gatius and González, 2005d]).

## 2.5 Generation of the system's messages

Natural language generation (NLG) is defined by [Bateman and Zoch, 2005] as the process of building a text in natural language for communication aims. We can consider it as the opposite of natural language processing.

Grice defined the *cooperative principle maxims* in [Grice, 1975]. This study established the general conditions that apply to human conversations when participants cooperate and they are not trying to mislead each other. The *cooperative principle* is the basis of Grice's definition of *conversational implicatures*, which are classified into the four groups shown in table 2.1.

Most DSs do not use an advanced NLG module to generate the system's answer. Instead, they use template-based techniques or hand-crafted messages. The latter strategies are easier to implement and integrate into the DS and they do not have a strong influence on the overall performance of the DS. Nevertheless, few systems include advanced architectures for the NLG

task.

Figure 2.2 shows the architecture for an advanced NLG. In DSs, the input to the NLG is the representation of the system’s answer to a user request. If the DS uses voice input (i.e. the telephony channel), the NLG output feeds a TTS module.

The discourse planner is closely related to the system that uses it (such as the DM) and decides *what to say* in the system’s messages. Some of the most relevant studies on this task are the schema-based text system (e.g. [McKeown, 1985]) and rhetorical structure theory (RST, [Thompson, 1987]).

The surface generator, which is more closely related to linguistics, decides *how to say* the system’s messages. The input to the surface generator consists of fragments of the discourse plan. The output consists of complete sentences related to the actions, concepts, attributes and values to be communicated.

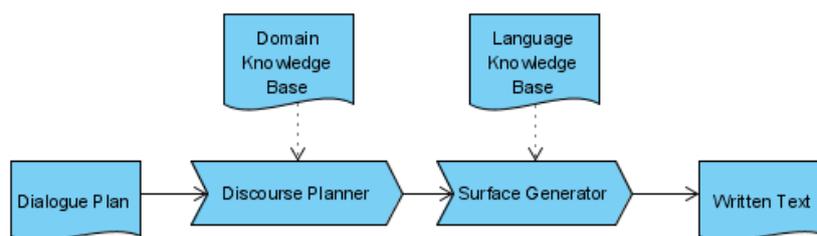


Figure 2.2: Natural language generation architecture

In this thesis, we are also interested in the problem of what to include in the system’s responses, which is the task of the discourse planner. We have addressed this problem in two stages: as part of the DM (selection of the agenda) and as a task management problem (classification of the results).

This approach to the discourse planner has been considered by several other authors. For instance, [Rieser and Lemon, 2009] describe an interactive question answering system that accesses databases instead of raw free text. In this system, the goal is to improve the performance of the information-seeking system. Therefore, they consider that the discourse planner is a main submodule of the DM. A typical dialogue strategy that this submodule uses consists of determining, at each turn, whether to report the results to the user or ask the user for more constraints.

## 2.6 Evaluation of dialogue systems

The evaluation and comparison of DSs is not a solved problem. Varying degrees of user expectations are met when different types of services are accessed, even when talking to humans. An evaluation of the DS performance must consider different features and always depends on the specific task that the system is dealing with. Thus, most existing DSs have been evaluated on the basis of the achievement of user expectations, instead of comparing the behaviour with other systems.

A widely used framework that supports this type of evaluation for spoken DS is PARADISE, which is described in [Walker et al., 1997]. The PARADISE framework, developed in the AT&T Labs, provides task representation that decouples *what* the system needs to carry out a task from *how* the system undertakes it. This task representation allows a comparison of dialogue strategies as it specifies the relative contribution of other factors to the overall system performance.

The evaluation of dialogue performance is a function of different measurements. Examples of measurements are: the number of utterances per turn, the number of turns, the number of repair turns and the elapsed time. Each measurement contributes to the calculation of the system performance as a function of the Kappa coefficient and a weight. The weight depends on the contribution degree to the user satisfaction, which is calculated from the user's quantitative agreement with some statements.

However, several disadvantages have been found. For instance, [Hadjinjak and Mihelic, 2006] highlighted some drawbacks in the way the PARADISE framework builds the function of user satisfaction prediction.

Recently, the Dialog Research Center (DialRC) at Carnegie Mellon has led the Spoken Dialogue Challenge<sup>8</sup>, a competition to compare different spoken dialogue systems' performance on the same task. The main goal of the challenge is to commit the specialized research community to a common evaluation framework. In addition, it aims to understand the variables that are involved and the mechanisms required to compare different systems.

---

<sup>8</sup><http://www.dialrc.org/sdc/>

## 2.7 Overview of related studies

This section gives a general summary of several well-known dialogue systems that are related to the DS presented in this thesis.

### The Galaxy-II architecture

The Galaxy-II architecture (Seneff et al. [1998, 1999]) supports DSs with a variety of characteristics: multi-modality, multi-channel, different domains and multilingualism. This architecture consists of a hub of servers. Each server is based on the sequential application of rules. In addition, a separate evaluation server has been developed. This enables scripts to be written for the hub, to run multiple versions of the same server and to compare them.

This architecture is one of the most frequently used systems for implementing practical dialogue applications, particularly for research purposes. For instance, the SLS group<sup>9</sup> has implemented several applications for different domains using Galaxy-II architecture. The most relevant among them are: VOYAGER (1989), PEGASUS (1995), MERCURY (1999), JUPITER (1997) and ORION (2000).

The main drawback of the system is that specialized skills are needed to develop new servers and run them into the hub. DIGUI also has a modular architecture, but it is probably less flexible than Galaxy-II architecture. Nevertheless, DIGUI already includes the implementation of the manager modules, so that only the specific resources (i.e. plans, grammars and lexicons) have to be developed to create a new application.

### The Rochester architecture

The conversational interaction and spoken dialogue research group at Rochester<sup>10</sup> has developed four DSs since 1982: ARGOT, TRAINS, TRIPS and CHESTER. TRAINS and TRIPS constitute two large research projects on spoken dialogue systems.

The first TRAINS version was developed in 1991 and newer versions have emerged as improvements have been made. TRAINS made it possible to compile corpus on train routing and the train scheduling domain, which is one of the most widely used in DSs research. Consequently, probabilistic DSs have been developed on the basis of this domain.

---

<sup>9</sup><http://groups.csail.mit.edu/sls/>

<sup>10</sup><http://www.cs.rochester.edu/research/cisd/>

TRIPS is of particular interest in relation to this thesis. It is a collaborative and multi-modal system for a rescue scenario emergency based on a generic system that uses plan-based and agent-based models for abstract problem-solving tasks. TRIPS architecture consists of three agents: interpretation, generation and behaviour. Each area includes a control manager. The control manager coordinates the actions and shares information with other managers. The information shared among the three agents consists of the discourse context, the reference management and the task management. This architecture is similar to the one we use in DIGUI. In our system, we have the NL understanding and generation, the dialogue manager and the task manager. The information shared among these modules is the dialogue context, the domain knowledge base and the task specifications. Thus, the main difference is the task model design. DIGUI has been adapted to access web services, while TRIPS uses models for abstract problem-solving tasks. In addition, since TRIPS is a collaborative system, neither the system nor the user has control of the interaction. In contrast, the DIGUI system includes an initiative model, so that one of the two agents holds the initiative.

### **The LIMSI RailTel<sup>11</sup> and Ritel<sup>12</sup> systems**

Another relevant system in the train timetable domain is RAILTEL, [Lamel et al., 1997]. Different prototypes were implemented at LIMSI (France), CSELT (Italy) and CCIR (United Kingdom). The RAILTEL architecture consists of six modules: the ASR, the NLU, the DM, information retrieval, the NLG and the speech synthesizer. This system has two interesting features in relation to our study: the NLG is based on grammars, in which the non-terminals are conditioned on the discourse context, and the system never gives a negative response to the user, but instead relaxes the constraints so that it can propose a solution. In the case of DIGUI, the NLG is based on patterns in which the data on the dialogue context are used to select the patterns and complete them. Hence, the NLG is context-driven, namely on the basis of the DM agenda. Furthermore, we propose user constraint relaxation mechanisms to obtain more suitable results from databases, and as a strategy to achieve more friendly communication with the user.

The RITEL system, described in [Rosset et al., 2006], is an open-domain interactive information retrieval system. Its architecture is different from RAILTEL, and so the task

---

<sup>11</sup><http://www.limsi.fr/tlp/railtel.html>

<sup>12</sup><http://ritel.limsi.fr>

paradigm that it follows is also different. A relevant aspect of this system is also the NLG component. This component is more important in interactive systems than in pure information retrieval systems, because the former need to demonstrate cooperativeness to the user. The RITEL system includes different NLG techniques to enhance communication. The system design and interaction among several components determine the type of interaction and identify which generation technique is appropriate for which purpose (e.g. flexibility, reuse of document formulations). The generation techniques are classified according to their functionality: to enhance the interaction or to enhance the search. Examples of generation techniques are: self-assessing formulations using system confidence, co-building the user query (e.g. confirmation), the resolution of potential ambiguities by formulating discriminating questions (i.e. clarification), phatic functions (e.g. feedback to the user), estimation of the granularity of the questions, complete and suggestive answers.

The RITEL system does not have a DM, but the modules are interconnected. In contrast, in DIGUI most of these properties have been implemented in the DM (e.g. confirmation, clarification, feedback), as a rule of the ISU.

### **The Adaptive Place Advisor system**

The ADAPTIVE PLACE ADVISOR, described in [Thompson et al., 2004], is a personalized system that provides recommendations of restaurants that meet the user's preferences. The ADAPTIVE PLACE ADVISOR architecture is similar to that used for typical DSs, although its objective is not to achieve a goal, but to give the user recommendation. We want to highlight this particular system because the conversational interaction that it carries out is similar to the process that our task manager follows. The system considers two phases in the conversation: constraint satisfaction and item presentation. During constraint satisfaction, the dialogue context determines the system's message and the set of user's answers that it expects. The solutions that the system provides are based on a constraint-satisfaction problem that follows a case-based reasoning system. The users can accept or reject the proposed solutions. Finally, user answers are used to eliminate alternatives, i.e. to constrain the query.

In DIGUI we follow a similar approach to access web services ([Gatius and González, 2010]). The task manager considers the following phases: task evaluation and execution, and presentation of the results. The former consists of evaluating whether the task can be

executed and then accessing the service (i.e. executing the task). The latter phase consists of determining the sort of results (e.g. list, item, missing data/constraints, error) and deciding what to give (the answer) to the user. In DIGUI, the user answer can constrain, relax or update the query (i.e. update the parameters and evaluate the task).

### **The SmartKom and SmartWeb systems**

The SMARTKOM [Wahlster, 2003] and SMARTWEB [Wahlster, 2007] projects are two well-known DSs that use ontologies to represent the domain and the application knowledge. In our system, we also take advantage of the use of ontologies for the interpretation process. Nevertheless, we are particularly interested in these two systems, as they are also web application-oriented. SMARTKOM is a multi-modal and collaborative system that has been integrated with 11 applications. In contrast, the SMARTWEB project tackles information-seeking dialogues and has been proven to work properly with task-oriented dialogues to perform transactions through a web service. The main feature of the newer SMARTWEB system compared to SMARTKOM is that it supports open-domain question-answering using the entire web as its knowledge base.

### **The MyMap system**

The MYMAP system, described in [Carolis et al., 2007], is an interesting study related to NLG. It generates personalized tourist descriptions for mobile devices. Although it is not exactly a DS, the MYMAP system also models the user and the context information, and then uses it to generate natural language responses. A graphical map made for a human tourist guide is annotated and used as the domain knowledge. Then, this domain knowledge is used to generate the natural language explanations and suggestions. First, the discourse planner creates a plan-tree for the answer. It uses a library of plan-recipes that specifies the structure of a discourse plan based on the RST method. The discourse planner selects the plan that best fits the current context and instantiates it. At the end of this operation, the surface generator obtains an XML file containing the plan-tree, which can easily be converted to any other format using XSLT. Finally, it converts the plan-tree into a natural message and generates the final tourist descriptions.

In DIGUI, the discourse planner of the NLG is basically driven by the agenda of the

DM. Then, the NLG uses the dialogue context to select the set generation rules that better realize the Agenda. The surface generator, in collaboration with the Adaptive module, selects the sentences that linguistically express the selected actions that the Agenda undertakes.

### **The CHAT system**

CHAT is a conversational helper for automotive tasks that has been specifically designed to deal with different aspects of imperfect communication. Specifically, the study described in [Vargas et al., 2009b] addresses the methods and strategies used to deal with information-seeking dialogues and interactive item selection. It particularly stresses the presentation of the information, which is also related to the study carried out in this thesis. [Vargas et al., 2009b] show that improving the system responses and, consequently, the dialogue communication, depends on the proper representation and management of the data involved when the application is accessed.

The constraints expressed by the user are generalized and specialized (the task model contains the information needed to determine how to perform them). Then, the knowledge manager and the content optimization select the data from the application databases and modify the queries. The methodology also includes the computation of statistics and meta-knowledge about the results. These statistics are used to generate more effective responses.

The DIGUI's objectives include obtaining friendly communication with the user. We have studied a corpus of dialogues to determine the structure of the tasks. Throughout this study, we have classified the kind of contents that the system can produce, and determined how to better provide the contents interactively. In addition, we also have used the corpus of dialogues to set up a user model that aims to automatically adapt the initiative of the dialogue interaction. In the future, we are interested in extending the models to tackle additional user features, such as those in the CHAT system.

# Overview of DIGUI

---

The DIGUI dialogue system (DS) focuses mainly on two aspects: i) achieving friendly communication and ii) portability, so as facilitating adaptability to different applications (particularly web services), languages and modes, including voice-based platforms. To achieve this, we aimed to develop a modular architecture whose design facilitates the integration of new applications and languages into the system.

A step towards the achievement of friendly communication has been taken in more complex systems by utilizing information about the user (beliefs, desires and intentions), general dialogue mechanisms (such as clarifications and corrections) and communication plans (such as plans defining the steps to follow to solve a domain problem). DIGUI incorporates a module that shifts the initiative between participants according to several metrics related to communication progress. This mechanism makes the user feel understood. The cost of developing mixed-initiative DSs is high, particularly if they must be adaptable to new applications. DSs that support mixed-initiative communication have to consider several aspects of the interaction to cope with the user's intentions. Consequently, we studied the most appropriate representation of the types of knowledge involved in communication that takes place when the user is guided to access the web: general dialogue mechanisms, domain-restricted knowledge (linguistic and conceptual), and information about the user.

In our system, application and language restricted knowledge is completely separated from the general processes. The language-dependent modules use general plus application-restricted linguistic knowledge that can be easily translated into several languages. This methodology, described in [Gatius and González, 2005d], has been proven for Catalan, Spanish, English and Italian. Hence, the dialogue manager (DM) is application and language independent.

In order to facilitate the integration of other services into the system we studied the application's task structures and we developed a generalization of the task models that makes it possible to represent a wide range of web service applications, which are the type of applications we considered in particular in the design of the DS.

This chapter gives an overview of the architecture of the DS. It describes the architecture of the DS and gives the functionalities of each module and the general workflow of the system. Finally, there is a description of the type of applications we consider within DIGUI and how they are structured and represented.

### 3.1 Dialogue system components

The general architecture followed in most well-known DSs is well described in [Jurafsky and Martin, 2008]. It consists of separate and interrelated modules, each of which is responsible for controlling a specific subtask involved in the human-computer interaction. A brief explanation of the modules is given below:

- NLU: natural language understanding generates a representation of the user's turn.
- NLG: natural language generation produces the system's response.
- DM: the dialogue manager can be seen as the central module of the DS that controls communication between the participants.
- TM: the task manager or application manager holds specific knowledge about the applications that are being accessed through the DS and controls access to them.
- ASR: the functionality of the automatic speech recognition is to convert an acoustic signal into a textual representation.
- TTS: the text-to-speech module converts the textual sentence generated by the system into an acoustic signal.

Figure 3.1 shows a diagram of the general architecture of DIGUI. It fits the general architecture briefly described above. The figure shows the components that are involved in each module and the direct dependencies among them. Solid arrows indicate the logical communication flow in the architecture, while dotted arrows indicate the dependencies among the components.



The DM and TM are the central modules in the DS. The DM is responsible for incorporating into the IS the knowledge obtained from the user's turns, as well as for planning and generating the system's turn. The design of the DM follows the information state update (ISU) approach (described in Section 2.2). This approach endows the system with more natural communication, which is one of the main focuses of the DS design. The ISU approach uses an explicit dialogue model that supports complex communication phenomena. Along with the DM, we developed the adaptive module, which was described in [Gatius and González, 2009]. This module is responsible for the variation of the initiative in the system given several metrics related to communication. Specifically, it gives different weights to various cues that are related to the dialogue progress. These cues determine the most appropriate agent (user or system) to hold the initiative during the most immediate dialogue turns. Then, the system is responsible for dynamically varying the degree of initiative during the conversation, and for adapting the system's response to the communication.

The TM drives the completion and execution of the tasks. It accesses the services and provides the results of the task execution. We have defined several schemes with which we can represent a wide range of web services. These schemes facilitate the integration of new services and different domain data into the platform.

Chapter 4 and 5 give detailed descriptions of both modules, the DM and the TM, the dialogue and task models, the processes they follow and the strategies implemented. Both modules have been fully developed by the author for this thesis.

The NLG module is called once the DM has decided the data to include in the system respond. It is composed of two components that can be found in most NLG architectures: the response generator and the surface generator. The former uses the results of the DM to decide what to respond, whereas the latter gives the specific sentences that will be displayed to the user. The NLG module was developed by the author of this thesis in collaboration with a linguist, who developed the linguistic resources. The result of this development was described in [Gatius et al., 2007a]. More details are given in Chapter 6.

The NLU module generates the interpretation of the user's turn and has two components: the syntactic-semantic analyzer and the interpretation component. The analyzer is a general purpose left-corner parser that uses context-free grammars. The grammars used in the system include general and restricted rules for specific applications, which are developed

for an application's domain. This component has been adapted from a previously existing parser described in [Gatius, 2001]. The interpretation component processes the semantic representation given by the analyzer and generates a more accurate and domain-oriented interpretation using the knowledge IS and the CK structures. Our prototype only supports text-based communication. However, as its design facilitates the incorporation of voice mode, the grammars have been specially designed to be completely compatible with the SRGS grammars<sup>1</sup> used in the VoiceXML specification<sup>2</sup>. This component and their resources, described in detail in Chapter 6, have been specifically developed for this DS in collaboration with the thesis advisor, who developed the semantic interpreter for the parser.

## 3.2 Interaction flow of the dialogue system

Figure 3.2 shows the general schema of the interaction process within the DS. A detailed description of this process for the specific example in Dialogue 3.1 is given below. In this example, the user is looking for a cinema showing a specific film.

### Dialogue 3.1

---

S1: Welcome to the Automatic Platform. How may I help you?

U1: I'd like to see Jolie's film.

S2: I am sorry. I don't understand your request. What information are you looking for?

U2: I want to see the film performed by Angelina Jolie.

S3: OK. Several cinemas are showing that film. Which location in the city would you prefer?

U3: Near the main station.

S4: OK. Two cinemas near the main station are showing the film you are interested in: Yelmo Cineplex and Cinesa. Do you want detailed information on either of them?

---

The interaction flow begins when the system initiates the communication (turn S1) and the user expresses the requirements using natural language (turns U1). Then, the system analyzes the user's sentence. Since no task or service has been initiated yet, the system must identify which action the user wants the system to perform (this is the user's goal). This

---

<sup>1</sup><http://www.w3.org/TR/speech-grammar/>

<sup>2</sup><http://www.w3.org/TR/voicexml21/>

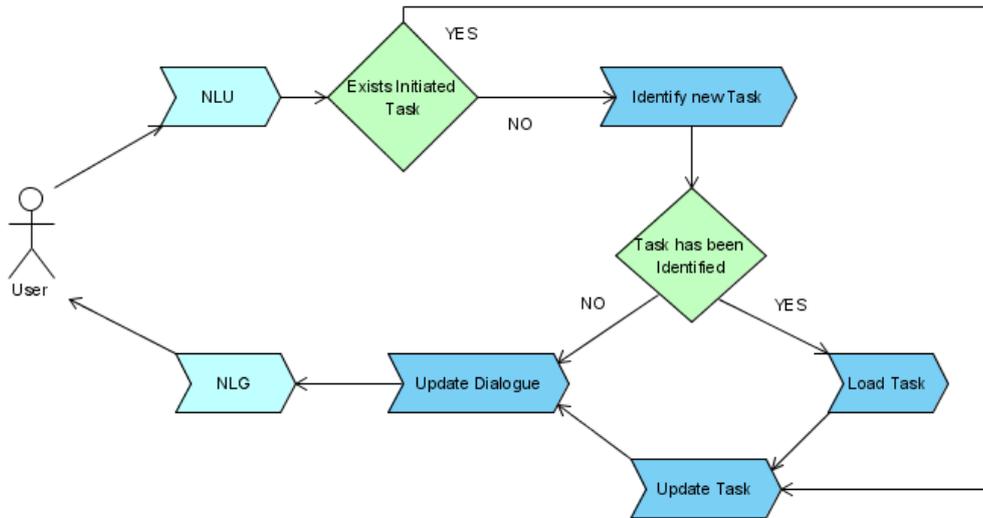


Figure 3.2: The interaction flow within DIGUI

process entails several steps. First, the system must identify which service the user intends to use. Then, the system must also determine the task within the service. And finally, the system loads the determined task that follows up the user goal.

If the system cannot identify the service, it initiates a sub-dialogue (S2) to achieve this goal. Once the system has identified the service and the task (U2), the appropriate task is loaded and processed. Then, the task is executed (S3), if possible, and the system updates the dialogue context and gives the answer. If more information is still needed from the user, the system will ask the user. If the user interaction refers to a previously identified task or service that has not been executed already, the system updates the instance representing that task (U3).

A close examination of Dialogue 3.1 reveals the following about TM and DM functionalities. In Dialogue 3.1, the DM used the domain knowledge to integrate the information about the latest Jolie film into the IS. It means that the system knows *which film* is performed by *Angelina Jolie*. Then, the TM accessed the service and received the requested information. It can do so since the system knows *where* the Angelina Jolie film is being shown. The system has noted that there are *too many* cinemas playing the latest Angelina Jolie film, and that the user needs to give more concise constraints. And finally, the system consulted the CK to establish which parameter would best *narrow the constraints* and so the system suggests the user to select a city area.

The NLU and NLG modules perform opposite but similar processes. The former produces a structure that contains the data extracted from the user's turn, while the latter produces the linguistic realization of a structure containing the data related to the system's turn. The user's sentence is usually related to the information expressed by the system. Thus, the context of the communication helps to process the user's turns.

In Dialogue 3.1, the sentence U2 produces the following interpretation:

$$(3.1) \text{ user\_turn} = [\textit{serviceid} = \textit{ca}, \textit{participant} = \textit{“Angelina Jolie”}]$$

After receiving this structure, the DM loads the task related to the service identified as “*serviceid=ca*” and processes the schema related to this specific task.

The DM produces a similar structure, which contains the system's turn (3.2). The NLG produces the sentence using the information about the user's turn, the data from the context and the domain knowledge. It has two phases: first it organizes and completes if necessary the information related to the system's turn, in order to produce a natural sentence. Then, the NLG selects specific phrases to compose the final sentence (the linguistic realization). The selection of the phrases depends on the strategies selected by the DM for the specific turn. Different phrases will be used for the two modes of the initiative: directed and open. The sentence S3 in Dialogue 3.1 is the result of the following dialogue moves, expressed in an open mode (without restricting the user's next turn):

$$(3.2) [\textit{answer}[\textit{load\_task} = \textit{ok}], \textit{answer}[\textit{location} = \textit{L}], \textit{ask}[\textit{location.district}]]$$

In this representation, L is a complex structure that is generated by the application. The TM includes a mechanism to detect when the number of results obtained from the application is too high to display them all in a single turn. In the example, the system generates a turn with the sentence “Several cinemas are playing that film”.

More formally, the interaction flow is:

- 1.NLU → analysis of the sentence
- 2.NLU → interpretation using the context
- 3.DM → integrate the user turn into the IS
- 4.TM → evaluate and update the representation of the task data
- 5.TM → execute the task
- 6.TM → build the results and update the IS
- 7.DM → update the IS
- 8.DM → select next actions
- 9.NLG → generation of the system response
- 10.NLG → linguistic realization of the system response

### 3.3 The web services typology

The system design was inspired by our work in the HOPS project<sup>3</sup>, a European project focused on the development of a multi-channel e-government platform to facilitate access to local administration services ([Gatius and González, 2005a]). Consequently, the HOPS-DS was initially designed to provide access to two public web services: LOC and CA. The LOC (large objects collection) service allows citizens to arrange for their old furniture to be collected. It also allows companies to get rid of large amounts of rubbish. The CA (cultural agenda) service enables citizens to consult the cultural events that take place in the city. The service also provides information about the events, such as where they will take place.

By analyzing the tasks involved in these services, we have established a set of different types of tasks that appear in the web services. We consider a web service as an application that offers several tasks that are accessible through the web. The DS offers access to these tasks. In our system, each task can be split into smaller ones, and these subtasks can be shared among other tasks.

We consider two types of web services: transactional and information. Figure 3.3 represents the general typology we consider for tasks (bottom boxes) and the relation with the specific service tasks integrated into our prototype (upper boxes). We represent the communication

<sup>3</sup>The Deliverable D4.3. of the HOPS project describes the natural language and dialogue management technologies we developed, [Gatius et al., 2007b].

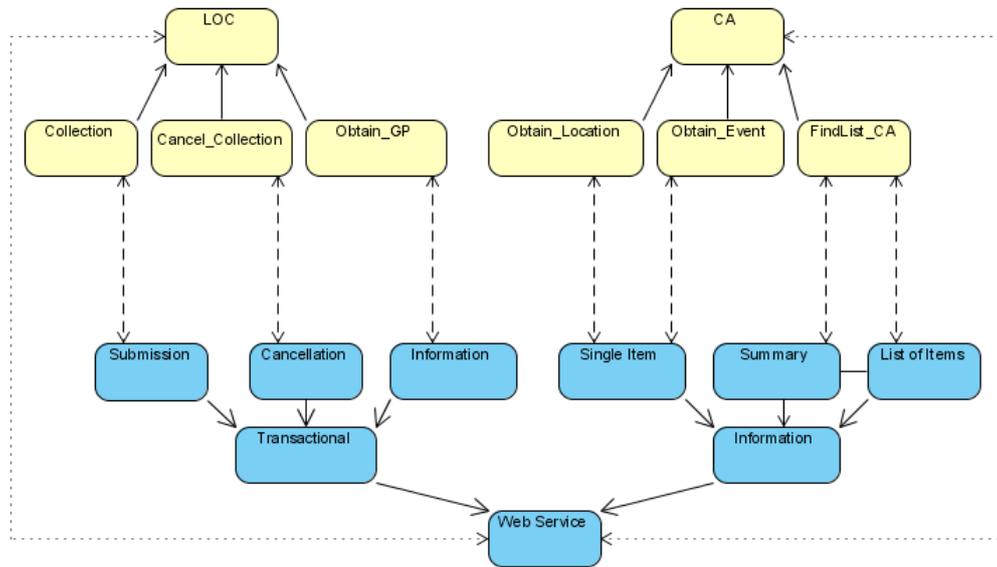


Figure 3.3: Classification of the tasks in DIGUI

tasks involved in these types of service as top-level tasks with abstract operators that can be decomposed into a group of communication acts. We classify the tasks according to the different communication acts that they involve. As a result, the tasks can be semi-automatically instantiated for each web service.

Transactional services are those that perform a transaction. We divide the tasks involved in these services (possible type of user's goal) into three top-level tasks: submission, cancellation and information. The LOC service represented in the figure is a transactional service with three top-level tasks: arrange a collection, cancel a collection and search for and obtain information about recycling points<sup>4</sup>. The process of arranging a collection is composed of two tasks: state the collection date and submit the transaction. Cancelling a collection is also composed of two tasks: identify the collection and submit the cancellation.

Informative services are those in charge of seeking and presenting information. We consider three top-level tasks for these types of services: find a list of items, get all the known information about a specific item, obtain a summary of the results retrieved. However, not all the services will be able to produce a summary. The CA is an information-seeking service. The top-level tasks involved in the CA consists of: searching for a list of events given some restrictions (e.g., date, typology, location) and searching for information about specific types

<sup>4</sup>Recycling points are called Green Point (GP) is a location in the city where the pollutant objects are collected.

of elements (e.g., timetables and address).

Given an application, the course of these top-level tasks is carried out by one or more specific tasks. The task classification depends on the sequence of actions that the DS performs when the task is processed. We classify the tasks as follows: *SubmitForm*, *FindList* and *ObtainData*.

1. *SubmitForm*. This task consists of obtaining the value for some parameters, confirming the values and submitting them. For example, cancelling a collection is a *Submit Form* task.
2. *FindList*. The most commonly used operation in the web is to search and find a list of things, e.g., names, dates or prices. Given a specific service, one may specify some restrictions and obtain a list of objects that satisfy these restrictions.
3. *ObtainData*. The *ObtainData* task calculates a value or searches for a specific object given a set of constraints, e.g., it calculates or searches for all the information related to a specific cinema: the address, the services and the films.

Chapter 5 also describes the data and models defined for the processes of each type of task we have considered, using the examples of the two services that are integrated into the system: the CA and the LOC.

# Dialogue management

---

This chapter is focused on the dialogue manager (DM) design, which is a core module in the dialogue system (DS). The DM is responsible for controlling the interaction, interpreting the user's desires and deciding the system's next actions. This chapter describes the mechanisms that perform these tasks and the knowledge that is involved. The first section describes the DM component and its integration into the DS. Section 4.2 describes the dialogue model that the DM follows. Section 4.3 describes the communication plans. Section 4.4 describes the adaptive module.

## 4.1 Dialogue manager components

Our main goal was to develop a flexible and efficient DM that was independent of the specific service and language and could guide citizens when they accessed a web service by phone or text. We use the information state update (ISU) approach, described in Chapter 2, to design the DM in the DIGUI system. The ISU model is a formal approach to dialogue management that covers complex phenomena of dialogue. Unlike stochastic dialogue approaches, that need huge amounts of annotated corpus, the ISU model formalizes most relevant situations in dialogue. The ISU approach is appropriate for our work because it provides flexibility, adaptability and friendliness. In this approach, the dialogue model is explicit and completely separated from the task model. This separation enables the development of dialogue strategies that support more complex phenomena, such as user-initiative, feedback and clarification. Besides, the complete separation of the task and dialogue models facilitates the process of integrating new services because, even when new information about the application's tasks

has to be incorporated in the task module, the DM remains the same.

There have been several computational approaches to the ISU. The implementation of our DM, described in [Gatius and González, 2007a], follows the approach proposed in [Larsson, 2002], which is focused on the notion of “questions under discussion”.

Figure 4.1 shows the processes and the knowledge sources involved in dialogue management: the IS (dialogue context), the library of plans (communication plans), the task schema (TS, the application’s task description) and the conceptual knowledge (CK). The DM consists of three main processes: integrate, update and select. The integration process takes the interpretation from the NLU and incorporates it into the IS. The update process modifies the IS in accordance with the integrated data and the information about the TS. The update process also communicates with the TM module, and waits for the execution of the tasks and their results. The selection process decides which actions will be involved in the system’s next turn. These processes take into account the content of the IS and the CK and the results obtained from the TM in the update process. Finally, the DM also uses a user model, which uses data about different communication metrics, to decide the most appropriate agent to hold the initiative.

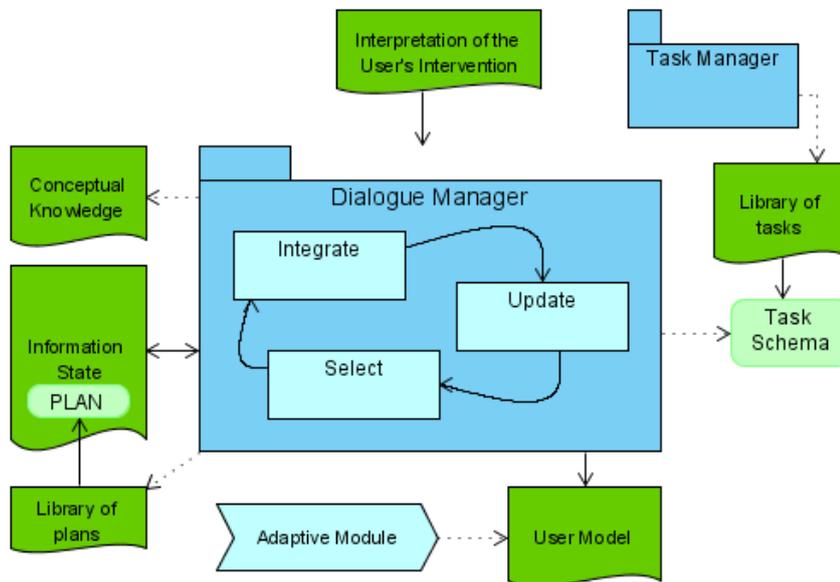


Figure 4.1: The dialogue manager

The following example illustrates the DM workflow through the turn shown in Dialogue 4.1. In this dialogue a user is looking for concerts in the Stadium on Saturday.

## Dialogue 4.1

---

U1: Which concerts are there in the stadium on Saturday?

S2: OK. There are no concerts in the Stadium on Saturday. There are several concerts on Saturday in other places. What type of music are you interested in?

---

The semantic representation of the user turn (U1) that the NLU generates is shown below:

$$(4.1) \ [type = music, location\_name = Stadium, date = Saturday]$$

The DM receives the interpretation (4.1) from the NLU. During the Integrate process, the DM identifies, within this interpretation, the specific information that is related to one or more tasks in the library of plans. The content of the CK and the IS aids this process by providing the information related to the specific domain of the applications. In this example, it is identified a task that belongs to the CA service. This task has the following two attributes among others: *queryConstraint* and *requestedData*. The DM binds the data involved in (4.1) with the data in the task as follows:

$$(4.2) \ [queryConstraints : [type = music, location\_name = Stadium, date = Saturday], requestedData : [title]]$$

The new structure in (4.2) is part of the actual task data<sup>1</sup>. Notice that the *queryConstraint* and the *requestedData* did not appear in the NLU interpretation. This information is inferred by the DM from the information in the application's task specification<sup>2</sup>. Subsequently, the Update process passes the query to the TM module, which executes the task. In this example, the TM returns no results and so the user is informed (turn S2). Then, the TM searches for alternative results (the specific algorithm used to update the query constraints is described in Chapter 5). Concretely, the TM removes the attribute “*location*” from the constraint. The updated query is as follows:

$$(4.3) \ [queryConstraints : [type = music, date = Saturday], requestedData : [title]]$$

---

<sup>1</sup>The ActualTD is an instance of the application task being executed that contains the data obtained from the user.

<sup>2</sup>The task schemas (TS) are described in Chapter 5.

Using the constraints in 4.3, the execution of the task does return results. However, the results are too many and so the DM has to ask the user for additional attributes. In order to avoid the user to give again useless data, the system searches for the useful attributes that better classify the results obtained. In this example, the type of music is the most useful attribute for classifying the results.

During the Select process the DM determines the information that will be presented to the user. In the example, the system informs the user about the failure of the first execution and then about the results in the second one. Finally, it asks for the preferred type of music (S2). The NLG module composes the sentences in the system's response.

## 4.2 The dialogue model

As mentioned in previous chapters, our DM approach follows the ISU model, which provides the tools needed to formalize the dialogue model (the information considered by the dialogue participants) and to update this information during the communication.

The ISU approach consists of the following:

1. A set of dialogue moves
2. A description of the IS
3. A set of rules for updating the IS
4. A control strategy to raise the next rule

Our implementation of the ISU takes into account that the DS has to provide access to the two types of web services that we have implemented: transactional and information. The dialogue moves, the implementation of the IS, the set of rules updating the IS and the control algorithm are described in the next sections.

### 4.2.1 The set of dialogue moves

In our system, the representation of the agent's turn consists of one or more moves, each of which is linked to semantic content. The semantic contents consist of an attribute and optionally one or more values, operators, modifiers and/or extensions. The interactions of the user and the system can be represented by one of these dialogue moves or by a combination

of them. For example, a single turn can include an implicit confirmation (feedback move) followed by a question (asking move).

DIGUI uses the set of moves described in Larson's approach plus several additional moves (marked with \*) that we found were necessary to represent additional behaviours that occur during the process of the dialogue. Moves can be seen as an extension of the dialog acts that include those actions which do not have a direct linguistic realization, and usually only the system can perform. Examples of those moves are the execution of queries and loading plans. These are the moves typified in our DM:

#### **Moves allowed by both the user and the system**

Greet. This is used at the beginning of the communication and it is associated with loading the initial plan.

Asking. This is related to raising an issue. Issues are related to the actions the system must follow to solve them, which in turn correspond to a specific plan. An asking move of the system is an issue to be raised by the DM and to be solved by the user. Usually it corresponds to the input parameters of the service tasks (which user must provide).

Answering. This concerns addressing an issue. When the move is performed by the user, it usually corresponds to the input parameters of the service; when the move is performed by the system, its content is the result of the issue that the user raised.

Feedback. This refers to the previous sentence. This approach considers basically three types of feedback: positive, negative and interrogative. Positive feedback represents an implicit confirmation of the understanding, e.g. *Understood. You want to consult the cultural agenda*. An interrogative feedback represents an explicit confirmation of understanding, e.g. *You want to consult the cultural agenda. Is that correct?*. Negative feedback represents a misunderstanding, e.g. *Pardon? I don't quite understand you*.

Pagination\*. This move is related to the presentation of the results. It takes a list of items and allows the user to browse the contents of the list. The system controls the browse

interaction. The user can select one item or move through the items.

Quitting. This indicates that an agent (the user or the system) wants to finalize the communication.

### **Moves allowed by the system**

Confirmation. This concerns the actions that have been undertaken. In our prototype only the system uses this move because it is the only agent allowed to realize actions (i.e. accessing the service). However, in other systems, such as assistants, the user can perform actions and confirm that they have been undertaken (i.e. turn on/off something).

Clarify. This 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, different services could use the same information, and the DM needs to ask the user to clarify which service or task he/she wants to access.

Reset\*. This control move is used to return to the beginning of the specified plan.

Macroaction\*. This indicates that the system must load a set of actions as a block. It differs from loading a plan in that the actions of the macroaction are also part of the plan, and they depend on the result of preceding actions.

Executing\*. Since we have separated application management from dialogue management, we use this move to represent access to the service task, regardless of its type. Each type of task generates a certain set of moves that represent the result of the execution and so the DM proceeds presenting them to the user. For example, when a transactional service is accessed, the execution updates the databases in the services. As a result, this move becomes a confirmation move and takes an attribute that indicates the status of the transaction. In contrast, when an informative service is accessed, the execution of this move consults the databases. Then, this move becomes any of the following ones depending on the possible results:

- Answering, an answer move for each attribute of the result with its values.
- Pagination, a list of items that matches a query.
- Asking, a set of ask moves to obtain more information from the user.
- Feedback, giving information about results, bad input values or access errors.

### 4.2.2 The information state

The IS represents the dialogue context and the state of the communication. Among all the structures in the IS, some of them contains information known by and shared between both agents, the system and the user; while other structures represent information known only by the system.

The following structures are used by both participants:

COMMITMENTS. This contains the information that has appeared and already been accepted in the communication. The information that one participant has expressed (the beliefs) becomes a commitment if it is not rejected by the other participant.

QUESTION UNDER DISCUSSION. This information consists of the local unsolved user goal. It includes information related to the available PLANS, as well as the information that the system has to confirm. These data are also used for ellipsis resolution.

ISSUES. This information consists of the open issues, i.e. the information that has been accommodated or optimistically grounded, and has yet to be confirmed. Once confirmed, it is stored in the Questions under discussion structure.

PAST MOVEMENTS. Both the user and the system turns are stored to solve simple references. For instance, an affirmative confirmation is related to a previous explicit confirmation question.

LAST UTTERANCE. This information consists of the last utterance processed. The information related to the agent's turns consists of the confidence level, the participant (user or system) and any errors given by the analyzer (nomatch, noinput or error) or the service

(dberror or timeout).

ACTIONS. This contains the action to be executed by any of the agents (only the system in our prototype), as well as the execution status (pending, done or error).

The following structures are only used by the system:

AGENDA. This information consists of the moves to be made in the near future. It determines the next system's moves.

PLAN and MACROACTION\*. The PLAN contains the long-term actions, which are obtained from the communication plans stored in the library of plans. The PLAN is the sequence of actions to be taken to achieve the goal of the user (and the system), which is directly related to the task under execution. The actions in the PLAN are processed at each interaction and they are loaded when the user raises an asking move. The actions in a plan may have preconditions, and hence each action is only loaded into the PLAN when its precondition is asserted. In addition, a plan can load other plans (e.g. the plan for quitting the service). When a plan references another one (using the macroaction move), it is loaded into the MACROACTION structure. This process enables the DM to know the actions that are related to a separate plan. In case the user rejects the actions taken by the system, then only the actions related to the rejected part of the plan are removed.

BELIEFS. Beliefs are either propositions obtained from the user or the results of a query. User propositions are known as beliefs until they are accepted (i.e. confirmed or not rejected) and so they become commitments.

TEMPORAL information about the user and the system. Information about the latest utterances that has not been confirmed is represented in independent data structures. These data structures consists of temporal information about the user and the system. They facilitate the development of corrective and recovery processes in case of negative feedback or confirmation.

PAGINATION moves\*. A pagination move is the result of obtaining a list of items from

the service. The information about all the items and the pagination status are stored in the PAGINATION structure, which also stores the information needed to browse and manage the results.

The following example illustrates the use of the IS structures. This example shows the content of the IS after a user's turn, and the changes applied after processing the system's turn: (i) The AGENDA is empty after the system's turn, (ii) the task has changed from "find\_ca" to "obtaindata.location", (iii) the PLAN is reduced to the remaining actions, (iv) the COMMITMENTS have been extended with the new known data, (v) the ISSUES have already been processed, and (vi) the ACTIONS have already been communicated.

---

IS Example

---

**U: Where is Aida being shown?**

AGENDA:

```
feedback ( [serviceId=ca], positive )
feedback ( [find_ca=executing] )
feedback ( [requestedData=location], positive )
feedback ( [title=aida], positive )
```

BELIEFS:

```
( [title=aida] )
( [date=[ 20090817 20090819 20090821 20090823 20090825 20090827 20090829 20090831 20090902 20090904
20090906 20090908 20090910 ] ] )
( [time=8pm] )
( [price=[ 8.75 24.25 98.25 127.75 181 ] ] )
( [location=gran teatre del liceu] )
( [channel=music] )
( [genre=opera] )
```

PLAN:

```
asking( queryConstraints )
asking( requestedData, [queryConstraints= no(title)] )
macroaction( constraintValues )
exequery( listofevents )
macroaction( listofevents )
macroaction( eventinfo )
macroaction( summaryevents )
asking( anothersearch )
```

---

Continues on next page

IS Example - Continued from previous page

---

```

reset( ca, [anothersearch=yes] )
asking( returnmenu )
reset( serviceIdentification, [returnmenu=yes] )

```

SUBACTIONS:

```

eventinfo:
  answering( price, [requestedData=price,requestedData=*) )
  answering( description, [requestedData=description,requestedData=*) )
  answering( sellpoint, [requestedData=sellpoint,requestedData=*) )
  answering( participant, [requestedData=participant,requestedData=*) )
  answering( genre, [requestedData=genre,requestedData=*) )
  answering( channel, [requestedData=channel,requestedData=*) )
  answering( location, [requestedData=location,requestedData=*) )
  answering( time, [requestedData=time,requestedData=*) )
  answering( date, [requestedData=date,requestedData=*) )
  answering( title, [requestedData=title,requestedData=*) )

```

COMMITMENTS:

```

( [queryConstraints=title] )
( [requestedData=location] )
( [title=aida] )
( [obtaindata_location = done] )

```

ISSUES:

```

( [serviceId=ca] )
( [requestedData=location] )
( [queryFocus=title] )
( [title=aida] )

```

LAST UTTERANCE:

```

Utterance:
score: 1.0
turn: user
status: correct
sentence: Where is Aida being shown?
actions:
  answering( [requestedData=location] )
  answering( [title=aida] )

```

PAST MOVEMENTS:

```

greet

```

QUESTION UNDER DISCUSSIONS:

---

Continues on next page

IS Example - Continued from previous page

---

```
( [title=aida] )
( [requestedData=location] )
( [requestedData] )
( [serviceId] )
```

ACTIONS:

```
confirmation( [obtaindata_location=done] )
```

---

**S: You are interested in the CA. I am searching. The opera Aida is playing in the "Gran Teatre del Liceu".**

AGENDA:

PLAN:

```
asking( anothersearch )
reset( ca, [anothersearch=yes] )
asking( returnmenu )
reset( serviceIdentification, [returnmenu=yes] )
```

COMMITMENTS:

```
( [queryConstraints=title] )
( [requestedData=location] )
( [title=aida] )
( [obtaindata_location = done] )
( [location=gran teatre del liceu] )
```

ISSUES:

```
( [serviceId=ca] )
```

LAST UTTERANCE:

Utterance:

score: 1.0

turn: sys

status: correct

sentence: You are interested in the CA. I am searching. The opera Aida is playing in the "Gran Teatre del Liceu".

actions:

```
feedback( [serviceId=ca], positive )
feedback( [obtaindata_location=executing] )
feedback( [requestedData=location], positive )
feedback( [title=aida], positive )
answering( location=gran teatre del liceu )
```

PAST MOVEMENTS:

---

Continues on next page

IS Example - Continued from previous page

---

ACTIONS:

---

### 4.2.3 Rules for updating the information state

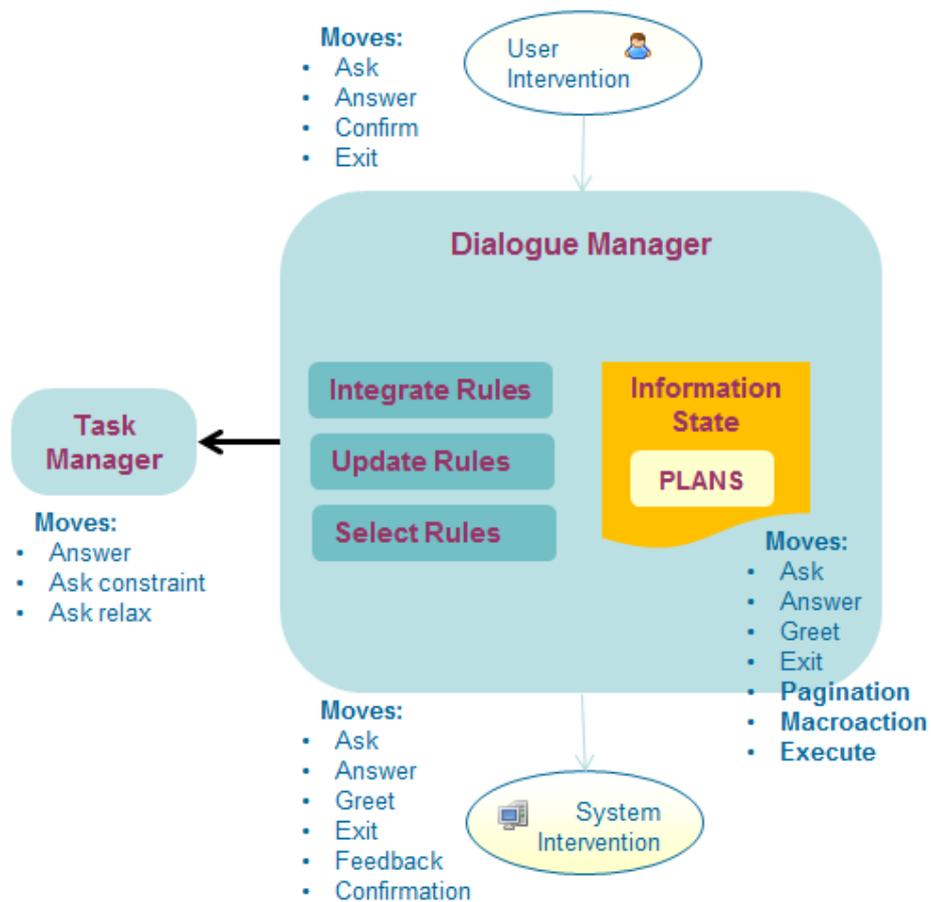


Figure 4.2: The dialogue manager flow

Figure 4.2 shows graphically the interaction among the ISU components. The DM algorithm consists of raising a set of rules sequentially and repeatedly when they are activated. These rules are grouped into three classes: Integrate, Update and Select.

The processes of integrating the content of one turn and updating the IS with that content are applied to both the user's and the system's turns. These processes consist of a set of ordered rules that sequentially process each piece of information in the utterance and incorporate the

data into the IS structure. There are several types of rules, for instance to detect the user rejection is related to system inputs and to process the results of the task execution is related to system outputs.

The selection of the content for the next turn applies only to the system's interventions. The Select rules decide the next dialogue moves and their content. A specific algorithm establishes the order in which the rules are raised. Examples of selection rules are updating the AGENDA with the actions in the PLAN or with feedback moves.

#### 4.2.4 The control algorithm

A pseudo-code of the main algorithm is shown in figure 4.3. The first steps consist of the initialization of the data structures and the system variables. The activation of the rules starts when the representation of the user's turn is sent to the DM. The object utterance consists of the set of moves performed by the user. The `update algorithm` processes the utterance and updates the IS. When the `update algorithm` finishes, the next actions to be performed are placed in the AGENDA. Then, the `select algorithm` decides the system's next moves, which in turn generate the object utterance containing the system's intervention. This set of moves is used to call the LG module, which generates the sentences (language-dependent) of the system's response. The algorithm finishes when one of the agents makes a quitting move.

The function `execute plan` in the `update algorithm` manages the execution of the tasks. The subsequent process differs when different types of tasks are processed (see Chapter 5).

When form submissions are processed, the DM considers three possible results: done, roll-back (not possible to do) and error. The DM updates the move containing the requested execution, adds the information about the execution result, and places the move in the Beliefs. Then, other rules will fix the moves for communicating the result to the user.

When queries for obtaining data are processed, the DM considers four types of results: no results, one result, a list of results or too many results. If there are no results, a new move containing the execution move with this information is stored in Belives. If there is only one result, then the same move will be created and stored in ACTIONS. Since an item can contain more than one attribute, the DM selects which information is displayed to the user, depending on the plan and the user's answers. If the service returns a list of results, they are stored in a pagination move that, in turn, is stored in ACTIONS. The `update algorithm` prioritizes the

rules related to pagination while there are moves in the PAGINATION structure. When the user selects one item from the list, the information related to this item is stored in Belives, as in the case of only one result.

Finally, the **select algorithm** selects an action item from the AGENDA. Then, it selects a move that performs this action and adds it to the set of the system's next moves.

**Algorithm DialogueManager** (plans\_libraries, resources, Channel, Lang)

```

NextMoves ← {} // Initialize 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
display answer

repeat
  utterance ← wait_user_utterance // The user utterance is the NLU interpretation of the
  // sentence, either written or transcribed from voice

  NextMoves ← usr_process( utterance )
  answer ← sys_process( NextMoves ) // NextMoves generates system answer and utterance
  display answer // Returns to the FrontEnd the Output
until ( state ← stop ) // Only the system can change the state when a quit
  // move is processed

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

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 4.3: Pseudo-code for the DM algorithm

### 4.3 The communication plans

The DM uses communication plans to (i) recognize the user's intentions and (ii) determine the next actions that the system has to perform to satisfy the user's needs.

Traditionally, in plan-based systems, plans are generated dynamically during the communication. An example is the use of plan recognition techniques from the artificial intelligence

area to recognize the user's intention. Once the DM has recognized the user's goal, the system can optimize the plan to achieve it, e.g. taking into account 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, or to be optimized. Since the current web services supported by DIGUI are not very complex, the DM we have developed does not need complex reasoning capabilities for planning and plan recognition.

In our system, there is a plan for each service task that is considered to be the user's goal. There are more plans than services because a service can perform more than one task. Particularly, in the two services that are currently supported by DIGUI (Section 3.3) there are three plans for the CA service and three plans for the LOC service, plus one plan for the initial turn.

The plan defines the information that the system needs from the user (asking and answering moves), and the information that the system needs from the service. This information is mainly related to the input and output parameters of the service's tasks, but also other issues as confirmations. The plan also determines the relations between all the moves, by means of preconditions and/or the creation of separate plans.

A communication plan can be decomposed into simple and complex actions<sup>3</sup>. In the current implementation of the DM, the simple actions that appear in the communication plans are:

- Asking: the system asks the user for the information needed to perform the task.
- Answering: the system gives information to the user.
- Confirm: the system asks the user to confirm some data.
- Service access: the system accesses the service to perform a transaction or execute a query.

In the communication plan, each action is related to a particular parameter and contains additional information about the parameter, which is highly useful in some particular procedures of the DS modules. This information basically consists of:

---

<sup>3</sup>complex actions are named macroactions and they are a reference to another plan

- The arity of the parameter, which indicates whether the parameter allows just one value (unary) or more than one (n-ary).
- The default value, which is used to avoid asking the user for most usual values.
- The precondition of an action, which is related to the values of other parameters in the plan.
- The type of value, e.g. boolean, string, numeric or date among others. This is useful in the recognition process. For instance, to enable general grammars to recognize dates.

This is an example of an action in the LOC plan:

action asking address unary
-----------------------------

“*action*” identifies the new dialogue action.

“*asking*” identifies the type of action.

“*address*” indicates the attribute that must be asked.

“*unary*” indicates that the user must provide just one value.

In our latter studies, we analyzed the content of the plans and how they cope with the users expectations during an evaluation. Then, we designed general schemas in order to generalize the plans and automate the process of creating plans from the web services. This work and more details about the plans and the task schemas are described in Chapter 5 and Appendix B.

## 4.4 The adaptive module

As stated in the overview of the DS (Chapter 3), friendliness and flexibility are two of the main goals in the design of our DS. In order to achieve these objectives, we have also developed an adaptive module, which makes the system more collaborative and adaptable to different types of users and channels.

An important aspect in the design of a DS is to determine the degree of the initiative. In the design of system-initiative dialogues, the system’s messages have to be more directed and detailed. In contrast, user-initiative systems use more open messages giving to the user the freedom to guide the conversation. The initiative of a DS defines which agent guides the

conversation (and holds the initiative), and its degree varies from one system to another. For example, tutor or task-oriented systems guide the interaction until the objective is achieved. Nevertheless, the user can raise subdialogues for clarification, verification or correction of specific aspects of the task. From this point of view, the user is called to have a certain degree of initiative, although the system has control of the communication. Alternatively, in recommendation systems mainly the user holds the initiative. These systems can infer proposals or actions that are not mentioned or raised by the user, and propose them. Nevertheless, the user is the person who finally decides which action to perform or which plan to follow. Any other type of interaction in the middle of these two is called mixed-initiative with different degrees.

To increase the adaptability of the system to the user's expertise and the application complexity, our DS dynamically adapts the dialogue strategies (initiative and confirmation policies). Although the speech recognition confidence in spoken DSs is usually the most useful data for determining how well the DS is performing, other dialogue information can be used. We have incorporated the independent adaptive module into our DS. It takes into account cues about the interaction performance during communication to determine the degree of the initiative. Nevertheless, the DS strategy is always considered mixed-initiative as the user can, at any time, decide to select a new task and the system will always guide the user to introduce the required information. The use of different types of messages moves the task-initiative or dialogue control closer to either the system or the user. We mainly consider two degrees: open and directed. The DS adapts the system's messages to the degree of initiative indicated by the adaptive module.

#### 4.4.1 Obtaining cues

The adaptive module considers information about the communication interactions to finally determine whether an open or a directed system's message is more appropriate. In order to state which information from the communication interaction is useful we studied a corpus of dialogues. In these dialogues different types of users (different skills and ages) access different types of web services.

Two people manually labelled the corpus of dialogues. For each interaction, they decided when a system's open message may be more appropriate than a directed message. The

agreement between them was considered substantial, according to the kappa index resulted: 0.633804<sup>4</sup>.

We followed the methodology proposed in [Chu-Carroll and Nickerson, 2000] and integrated the adaptive model in DIGUI. We selected a set of data about the communication performance that could determine the most appropriate system's message. This methodology uses the Dempster-Shafer theory, which allows the combined effect of two or more pieces of observed data. At each dialogue turn, the resulting combination of the dialogue data determines how well the communication is progressing.

Table 4.3 shows the most significant cues we used to model the initiative shifts and the amount of evidence given by each cue. The first four cues in the table are related to special situations that occur when processing the user's turn. The first cue corresponds to the system's errors and provides most evidence for a shift to a directed system message. The second cue consists of the user's turn of asking for help and provides a moderate evidence for the use of a directed system message. The third and fourth cues are related to the system's expectations about the next user's turn. When the user gives information that the system does not expect, an open system message seems more appropriate; whereas if the user gives more than the information the system expects, both open and directed messages are appropriate. Finally, the last cue indicates that the system response should be an open message when the user gives new relevant information.

We also considered the following cues, but they did not provide enough evidence to decide which participant should have the task-initiative: the system turn consisted of asking for the same data, the system asked for the same data more than twice, the previous task had been completed, the user's answer partially fits what the system expected, the system only partially understood the user's answer, and the user's answer gave values for more than one new attribute.

To obtain the cue values modelling the dialogue we divided the tagged corpus into 8 sets. Then, we evaluated the system's performance at adapting the messages in the way that the people tagging the corpus considered most appropriate using an 8-fold cross-validation. The resulting model performed in the same way as human in 73.5% of cases. When open system messages were used, the performance was the same as human in 66.55% of cases.

---

<sup>4</sup>Values of kappa above 0.6 can be considered substantial agreement when data are not balanced, as it is our case.

<i>Cue description</i>	<i>Directed message</i>	<i>Open message</i>	<i>Neutral</i>
System error	0.21	0.02	0.75
Ask for help	0.17	0.00	0.82
Not expected	0.19	0.27	0.53
More than expected	0.22	0.19	0.58
New relevant information	0.17	0.31	0.50

Table 4.3: Amount of evidence given by each cue for shifting to directed/open mode of initiative or keep neutral

#### 4.4.2 Adapting the degree of initiative

Once the DM has decided what to respond, it has to decide how to express it. Directed system’s messages are explicit about the information the system needs from the user at each stage of the communication. Open system’s messages suggest that the user introduces the required information, but not as strongly.

Dialogues 4.2, 4.3 and 4.4 are examples of dialogues in which the user asks for information about the opera “Aida”.

In Dialogue 4.2, all the system’s messages are directed and explicit about the information requested. As a result, the user turn is very short (U1). The user writes the name of the play wrongly, and the system does not understand the user. We note that the following turn of the system (S2) is very similar to the previous one (S1). Then, the user notices the error and writes the name of the play correctly.

In Dialogue 4.3 all the messages are open. They invite the user to take the initiative. The user’s sentence is longer, and it also expresses more concisely what he/she is looking for (U1). After writing the name of the event incorrectly, the next system is still open, and the user answers with the same sentence, but with the mistake corrected.

In Dialogue 4.4 the system begins to use an open message (S1). Then, the system notes the mistake in the user’s sentence, and it adapts the communication strategy to use a directed message in the following turn (S2). Note that the user does not use the long sentence again (U1), but only the specific data (U2). Intuitively, this behaviour seems more natural.

Dialogue 4.2

---

S1: Please, describe the event by giving me information such as the title, the place, the date or the event type.

---

Dialogue 4.2 - Continues on next page

Dialogue 4.2 - Continued from previous page

---

U1: iada

S2: I am sorry. I do not understand you. Please, describe the event by giving me information such as the title, the place, the date or the event type.

U2: aida

S3: The event's title is Aida. It takes place in the Gran Teatre del Liceu.

---

Dialogue 4.3

---

S1: May I help you?

U1: Where iada takes place

S2: You want information about the cultural agenda. What kind of information are you looking for?

U2: Where aida takes place

S3: The event's title is Aida. It takes place in the Gran Teatre del Liceu.

---

Dialogue 4.4

---

S1: May I help you?

U1: where iada takes place

S2: You want information about the cultural agenda. Please, describe the event by giving me information such as the title, the place, the date or the event type

U2: aida

S3: The event's title is Aida. It takes place in the Gran Teatre del Liceu.

---

## 4.5 Summary

DIGUI architecture includes a user model that allows the system to support both open and directed initiatives. The open mode of interaction makes the conversation more natural. Since the dialogue model deals with richer interactions, the dialogue system can achieve friendly communication for different types of services and users.

Our dialogue manager follows the Information State Update approach. In this approach, the Information State contains specific information about the dialogue context. Then, a set of rules integrates the content of each agent's turn and updates the content of the Information State. This dialogue model supports richer conversation, since it can deal with more complex phenomena than systems that use simpler models. Although the current dialogue manager does not cover every phenomenon that appears in human-human or human-machine conversations, its design and formalization mean that it can be upgraded, i.e. new rules, structures or processes can be introduced so that it can include more features. For instance, we have introduced the use of conceptual knowledge, but we have not taken great advantage of it. Further work could include a more in-depth study of the conceptual representation.

In addition, the adaptive module dynamically calculates when the system should guide the interaction and when the user can do this. Hence, the system's messages are more directed when the module detects a communication problem. Directed messages are designed so that the user gives specific responses. Open messages invite the user to freely express his/her intentions. To determine the degree of initiative of the system, the adaptive module uses metrics related to the number of errors and the content of the user responses (e.g. the user's verbosity). This module could be improved by adding more features to the user model. For instance, under the voice mode of interaction, if the model took into account channel noise, the system could avoid the use of inappropriate messages.

Furthermore, our design also decouples the knowledge involved in the dialogue from the task knowledge. Specifically, our dialogue manager uses communication plans to guide the interaction. In DIGUI, these communication plans are obtained from the task specification. Some systems use sophisticated methodologies to extract the dialogue structure for a specific application either from the databases or from the service specification. We consider that the adoption of these kinds of methods could enhance the process of integrating new services into

the system, as this could generalize the creation of plans for other kinds of services. Moreover, it is also possible to define a specific language to formalize the plans, as other systems do, e.g. an adapted logic language. The use of formal languages increases the mechanization and adaptability of the system, as well as its complexity and the need for specific skills.

# Task management

---

The data related to the task are handled by the task manager (TM) module. The TM processes the results and the information that is needed to access web services. This chapter describes the TM. In particular, it discusses the schemas that are used to guide the interaction and access the web services, the structures that represent all the application related information (i.e. the web services), and how the TM processes them. The structure of the tasks determines most of the system behaviour, and the system must consider all the phenomena that are represented, so that they can be propagated to the system response.

## 5.1 Generalization of the service structure

In our system design, we aimed to ease the process of integrating a new service into the dialogue system. For this purpose, we collected a set of dialogues during an evaluation of the system, as described in Chapter 7. Using this dataset, we studied the structure of the dialogues for each type of task and for different types of communication phenomena (this study was described in [González and Gatiús, 2010]). For instance, we studied the use of explicit vs. implicit confirmation strategies. In this case, the system rather than the user chooses the interaction flow (even under open initiative mode). We also studied how to respond to the user in these kinds of human-computer interactions. The study of the collected dialogues can be found in Appendix B.

### 5.1.1 Service plans

As described in Section 4.3, our system has a plan for each service task. The plans represent how to achieve the users' goals. Plans are organized as top-level tasks that correspond to the main operations of the web service. As described in the service typology (Section 3.3), the top-level tasks can then be decomposed into smaller tasks. In addition, depending on how each task has to be implemented, it is recorded as: `SubmitForm`, `ObtainData` or `FindList`.

A plan is a set of ordered actions for guiding the DM to achieve a specific goal. The simplest plan in our DS is the initial plan, which guides the DM to obtain the specific service that the user intends to use. Figure 5.1 shows this plan. Specifically, line 3 identifies the plan: "serviceIdentification" is the name and "init" is a DM control command. Then, line 4 corresponds to an action move to ask the user the value of the attribute "serviceId". Line 5 indicates that there are no more actions to perform. And finally, line 6 indicates the end of the plan.

```

1: #
2: #initial service
3: plan serviceIdentification init
4: action asking serviceId unary
5: endactions
6: endplan
7: #

```

Figure 5.1: The 'initial' plan

We have defined general schemas to represent and process the tasks (and plans) for the two services studied in this thesis, namely transactional and informative services. Our proposal facilitates the generation of plans for new services.

### 5.1.2 Transactional service: general schema

The following procedure, in figure 5.2, is the general schema that represents the steps that the DM needs to follow to access a transactional service. The LOC is an example of a transactional service.

The LOC service has three top-level tasks: to arrange a collection, to cancel a collection and to provide information about Green Points (GP). The cancellation and collection tasks

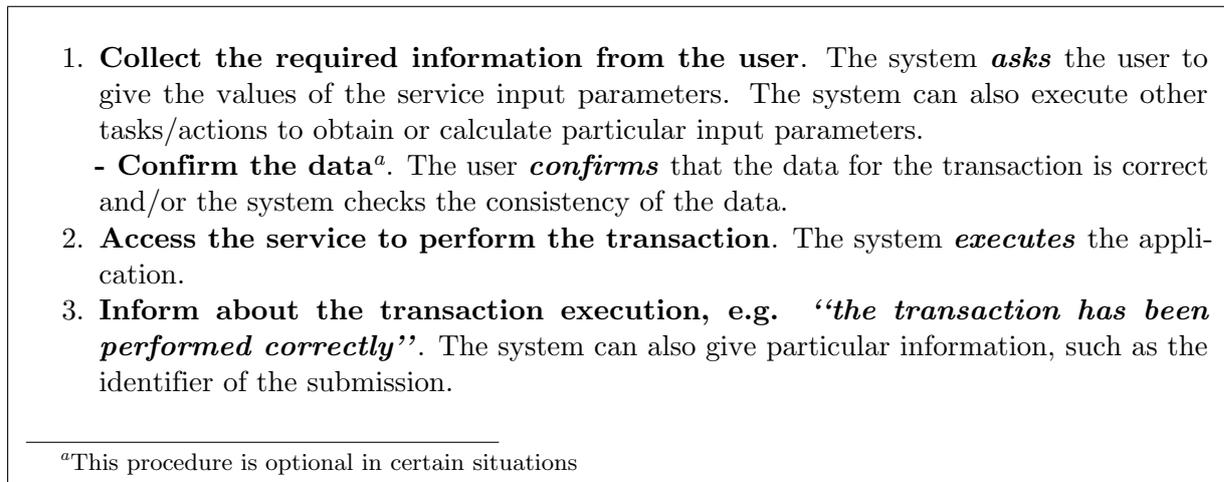


Figure 5.2: General schema for transactional services

register some data in the service’s database. Hence, they are recorded as SubmitForm tasks. In contrast, task used to obtain the collection date and to provide information about the GPs are recorded as ObtainData tasks. Both SubmitForm and ObtainData tasks follow the schema above, although ObtainData tasks do not confirm the data before execution.

For instance, in Dialogue 5.1 a user accesses the collection task to arrange the collection of bulk paper (U1). Sequentially, the system asks the user for the data needed to submit the transaction. First, the system asks the user to confirm the collection date (S3), which depends on the address (S2). In this turn, there is also implicit confirmation of the collection address. Then, the system asks the user to confirm the price of the collection (S5), which depends on the volume of paper (S4). In this turn, there is also implicit confirmation of the volume of the paper. Finally, the system submits the collection and gives the transaction number to the user (S6).

## Dialogue 5.1

---

S1: Welcome to the Automatic Platform. How may I help you?

U1: I want to arrange for paper to be collected from the University.

S2: What is the collection address?

U2: Diagonal 30.

S3: The collection will be on Wednesday at Diagonal, 30. Is that OK?

---

Dialogue 5.1 - Continues on next page

Dialogue 5.1 - Continued from previous page

---

U3: Yes.

S4: OK. What is the approximate volume of paper in cubic metres?

U4: About 30 cubic metres.

S5: The price of the collection of 30 cubic metres is 150 euros. Do you agree?

U5: Yes

S6: OK. I am registering the collection. The transaction has been done correctly. Your collection number is 12345. Do you want to return to the main menu?

U6: No thanks

S7: Thanks for using this Automatic Platform. Good bye.

---

Figure 5.3 shows the plans for the LOC service. The structure of the plans for the collection and cancellation tasks is quite similar (both are SubmitForm tasks). Lines 22 to 40 correspond to the actions carried out in the collection task. Lines 42 to 56 correspond to the actions in the cancellation task. In both tasks, the first actions are those required to obtain the input parameters. Thus, they are mainly “asking” moves. Then, the system performs additional actions to obtain extra input parameters. The three last actions in both tasks are directly related to the execution of the SubmitForm. The system confirms that all the data has been correctly understood before accessing the service to perform the transaction. Finally, the system presents the results (lines 33-35 and 49-51).

The cancellation task is very simple. The only parameter required is the identifier of the collection, which is provided when the collection is registered, or the address of the collection (lines 45 and 46). Using these data, the system accesses the service to obtain information about the collection (line 48). Before the collection can be effectively cancelled, the user must confirm the collection data. If the confirmation is positive, the system accesses the service again to cancel the collection. Finally, it informs the user about the submission status <sup>1</sup> (line 51).

In contrast, the collection task requires several input parameters: the address, the date and the volume or the specific objects. The system delegates obtaining the date to another task (line 25). The actions related to the volume and the objects have conditions. These

---

<sup>1</sup>The submission status is confirmation that the cancellation has been made or information about any particular error.

conditions separate specific actions for company users from those for private users.

The ObtainData tasks are similar to the SubmitForm tasks. However, ObtainData tasks do not update the service's databases. Instead, they only query specific data and present them to the user. Thus, ObtainData tasks do not require confirmation of the input parameters. The procedure is as follows. First, the system obtains the data for the input parameters. Then, it accesses the service to obtain the data. These data are either a calculated value (e.g. calculate the price as a function of the volume) or an item from the service's application (e.g. a collection, a location or an event). Finally, the system presents the data to the user. The results are given together with the input parameters that were used (implicit confirmation). Subsequently, the user can repair any misunderstood value or ask for more information.

In the LOC service, four sections correspond to ObtainData tasks. The first section groups the set of actions needed to obtain the collection date (lines 24-26). The second one deals with the set of actions needed to obtain the collection that has to be cancelled (lines 44-48). The third one is concerned with the set of actions needed to check that none of the objects are pollutant (lines 30-32). Finally, there is the top-level task to obtain information about the GPs (lines 11-13).

### 5.1.3 Informative services: general schema

The informative service we have studied is an information-seeking system. The user's intention when he/she accesses an informative service is to obtain specific information, given various sets of constraints. In most cases, the DM guides the user to describe the information that he/she requires. This information is used to constrain the search. Then, the results are presented to the user. A specific characteristic of this type of service is that the system uses an underspecified set of constraints to drive the interaction, rather than a defined user's goal (which can be broken down into tasks and subtasks).

Hence, the main task of an informative service is to guide the user to access the information. In addition to the main task, we propose three top-level tasks in this service. We classify these top-level tasks according to the type of results obtained from the application. Some examples of results are: information about an item, a list of items and a summary of the retrieved items. The following procedure, in figure 5.4 is the general schema, followed by the DM, used to

```

1:
2: #LOC SERVICE
3:
4: plan loc serviceId
5: action asking loctype unary
6: endactions
7: endplan
8:
9: #information
10: plan information loctype
11: action asking address unary
12: action executing gpinfo unary
13: action answering gpinfo unary
14: action asking anotherSearch boolean
15: action reset information unary commitment anotherSearch boolean yes
16: action asking returnmenu boolean
17: action reset serviceIdentification unary commitment returnmenu boolean yes
18: action quitting
19: endactions
20: endplan
21: #
22: #collection
23: plan collection loctype
24: action asking address unary
25: action executing collectiondate unary
26: action answering collectiondate unary
27: action asking agreedate boolean
28: action asking userType unary commitment agreedate boolean yes
29: action asking volume unary commitment userType unary company
30: action asking object multiary commitment userType unary private
31: action executing pollutant multiary
32: action answering pollutant multiary commitment pollutant multiary *
33: action asking confirm boolean
34: action executing transactionId unary commitment confirm boolean yes
35: action answering transactionId unary commitment confirm boolean yes
36: action asking returnmenu boolean
37: action reset serviceIdentification unary commitment returnmenu boolean yes
38: action quitting
39: endactions
40: endplan
41: #
42: #cancellation
43: plan cancellation loctype
44: {
45: action asking transactionId unary
46: action asking address unary
47: }
48: action executing collection unary
49: action asking confirm boolean commitment collection *
50: action executing cancelled unary
51: action answering cancelled unary
52: action asking returnmenu boolean
53: action reset serviceIdentification unary commitment returnmenu boolean yes
54: action quitting
55: endactions
56: endplan
57: #

```

Figure 5.3: 'LOC Service' Plan

1. **Collect the requested information.** In this step, the user specifies which information he/she wants (*requestedData*).
2. **Collect the information needed to restrict the search.** The system asks the user for the particular characteristics of the elements that he/she seeks (*queryConstraints* and *constraintValues*).
3. **Access the service.** The system accesses the web services through a specific module. The execution of the application sometimes involves updating the constraints. For example, the query might be relaxed or default values might be used.
4. **Present the results.** The items that satisfy the query are presented. If too many elements are obtained, then the module presents a summary of the data to the user. Besides, the system establishes the parameters needed to constrain the next search and asks the user for these values. If only one object is obtained, all the data related to this item are displayed to the user. If the system obtains a list of objects, it presents them to the user, together with a partial description. In addition, the system asks the user to select a specific item from the list.

Figure 5.4: General schema for informative services

achieve the user's goal.

Figure 5.5 shows the general procedure that the informative service follows. A main task precedes the three top-level tasks. It corresponds to the two first steps in the aforementioned general procedure and consists of the actions needed to obtain data for the query: i) the parameters for constraining the query (*queryConstraints*), ii) the values of these parameters (*constraintValues*) and iii) the output parameters (*requestedData*). The flexibility of the general plan for information-seeking services allows the user to shift from one task to another. For instance, as illustrated in Dialogue 5.2, when an item (U2) is selected from a list of results (S2), the system shifts from the task that provides a list of results to the task that gives certain information about a specific item (S2). The general process is as follows: each time the constraints are updated, the system processes the service plan from the beginning, generates a new query search, and accesses the service.

The process of obtaining the query constraints is complex, as they are not gathered in a predetermined order. After the user's turn, the system determines whether a complete query can be generated. Otherwise, it determines which data are missing and need to be obtained from the user. If the service's definition includes default values, then the system can use them

to complete the query. Moreover, the parameter values that appeared in previous turns can also be reused.

#### Dialogue 5.2

U1: And are there any theatre performances for children today?

S2: OK. I am searching. There are 2 events. Results follow:

1: Titelles,

2: Contacontes.

Select one of the results or change the search criteria....

U2: What time does Contacontes begin?

S3: OK. The event Contacontes begins at 12 a.m.

U3: Give me all the information

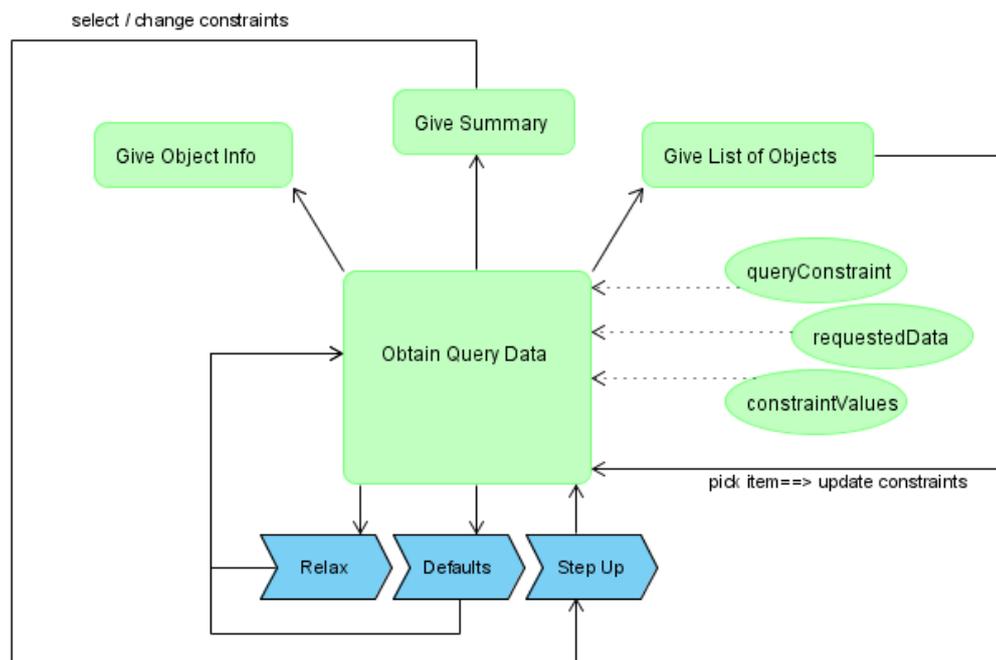


Figure 5.5: General workflow for informative services

Next, we describe the use of the general schema of informative services in the CA service. Figure 5.6 shows the complete plan for the CA service. Lines 5 to 21 correspond to the task that controls the flow of the service. This task obtains the query parameters, accesses one of the three top-level tasks and reinitiates the task that ends the communication. Then, lines

24 to 32 correspond to the actions of obtaining the specific values for the input parameters. Lines 36 to 40 correspond to the actions of browsing a list of items. Lines 44 to 56 correspond to the actions of presenting the information related to a particular event. Finally, lines 60 to 64 correspond to the actions of presenting a summary of results to the user.

In our implementation, the messages that request data from the user to restrict the search are at the beginning of the plan (lines 6-8). When all the data needed for the query have been obtained, the system accesses the service (line 9). Then, it follows the specific plan that is related to the type of results obtained. If there are no results, the DM can automatically relax the constraints and execute the task again. The constraints can be relaxed in two ways. In the query level, the system selects and removes one or more of the attributes contained in the query. In the parameter level, the system selects and updates the value for one or more of the attributes. The system includes general knowledge for relaxing commonly used data, e.g. dates and locations. In addition, the specification of the service tasks has to include the information needed to relax the specific parameters of the service.

Dialogue 5.2 shows an example in which the system presents the results in turn (S2) and the user picks up the item “Contacontes” in turn (U2).

When the results from the web service are a list of items, they are processed using the move pagination in the plan (line 37 of the plan in figure 5.6). This action allows the system to define the maximum number of results to present and the number of results that are displayed at each turn. It is implemented as a loop that ends in one of the following three situations: i) there are no results to display, ii) the user changes the constraints and iii) the user picks up an item from the list (line 38). In the last two cases, the constraints change and the system must therefore reinitiate the task flow.

When only one item is retrieved, for instance in turn (U2) in Dialogue 5.2, the system displays the specific information requested or all the available information (line 12). This task is a set of answer actions, one for each attribute of the available items. In the specific case of the cultural agenda we have implemented, there is a task for the events (lines 45-54) and another for the locations (omitted due to lack of space, it follows the same structure as the task for events). Each answer action contains a condition that determines when the action is raised. For example, the condition in line 45 states that the system will give the price value

when either the requestedData contains the name of the proper attribute (the price) or when all the information is required (here, the \* means all the information).

The answer summary action (line 61) contains all the information related to the data that is obtained after the execution of the application. The system will provide concise information for the user if the NLG module is capable and has the required rules and messages. Otherwise, the system will provide general information. The system can guide the user by suggesting the data that he/she may provide. Once a user has provided the new constraints, the system executes the application again.

## 5.2 The task manager

To facilitate the integration of new services into the system, we designed a general algorithm for the task manager. Then, we created specific task structures and procedures. We decoupled these task structures from the specific domain of the task and the specific content. The task structures represent the data involved in a specific type of task. The task schema is the content assigned to a task structure for a specific application (and domain).

### 5.2.1 Task representation

This section describes the data structures that are used to represent the web services tasks. One of the main goals when designing these structures was to facilitate the adaptation of the DS to different types of applications. Consequently, we represented in different knowledge bases the services task specifications and the data related to the task in progress. The task specification is represented in the task schema (TS) and the task information involved in dialogues is stored in the task data (ActualTD).

The TS has been designed to model the information related to a specific type of task. Three different models of TS correspond to the three types of tasks considered in this thesis: SubmitForm, ObtainData and FindList. Each TS defines the specific data of the task that it represents: e.g. parameters, constraints and conditions. A TS for a specific service consists of: the web service access task; the available requests for each task (input parameters); the information that each task returns (output parameters); and further requirements and conditions related to all the data involved. This information, which is part of the service

```

1:
2: #CA SERVICE
3:
4: # Query Data
5: plan ca
6: action asking queryConstraints multiary
7: action asking requestedData multiary default title commitment queryConstraints multiary title noext
8: action macroaction constraintValues
9: action executing findlist_ca multiary
10: {
11: action macroaction listofevents
12: action macroaction eventinfo
13: action macroaction summaryevents
14: }
15: action asking anothersearch boolean
16: action reset ca unary commitment anothersearch boolean yes
17: action asking returnmenu boolean
18: action reset serviceIdentification unary commitment returnmenu boolean yes
19: action quitting
20: endactions
21: endplan
22:
23: # auxiliary subaction for obtaining the queryConstraints values
24: plan constraintValues
25: action asking price multiary commitment queryConstraints multiary price
26: action asking location unary commitment queryConstraints multiary location
27: action asking date multiary commitment queryConstraints multiary date
28: action asking genre unary commitment queryConstraints multiary genre
29: action asking channel unary commitment queryConstraints multiary channel
30: action asking title unary commitment queryConstraints multiary title
31: endactions
32: endplan
33: #
34:
35: # List of Objects
36: plan listofevents
37: action pagination findlist_ca multiary commitment list unary findlist_ca listindex requestedData
38: action reset ca unary commitment selectitem unary *
39: endactions
40: endplan
41: #
42:
43: # Object info
44: plan eventinfo
45: action answering price multiary commitment requestedData multiary price,all
46: action answering description unary commitment requestedData multiary description,all
47: action answering sellpoint multiary commitment requestedData multiary sellpoint,all
48: action answering participant multiary commitment requestedData multiary participant,all
49: action answering genre unary commitment requestedData multiary genre,all
50: action answering channel unary commitment requestedData multiary channel,all
51: action answering location unary commitment requestedData multiary location,all
52: action answering time multiary commitment requestedData multiary time,all
53: action answering date multiary commitment requestedData multiary date,all
54: action answering title unary commitment requestedData multiary title,all
55: endactions
56: endplan
57: #
58:
59: # Summary
60: plan summaryevents
61: action answering summary findlist_ca
62: action reset ca
63: endactions
64: endplan
65: #

```

Figure 5.6: “CA service” plan

resources described in Appendix C, is stored in XML files. The set of XML files compounds a library of tasks. Hence, the process of executing a specific task consists of instantiating and processing a specific TS, according to the information contained in the ActualTD.

ActualTD represents the specific data for the ongoing task execution. The system needs to know the TS to process the information contained in the ActualTD. Each type of task is processed by a specific algorithm. These algorithms are very simple. The algorithms for the SubmitForm and ObtainData tasks search for a suitable list of parameters and check that no conflicts among the parameters prevent the system from accessing the service. The algorithm for the FindList process is only aware of the number of results. It organizes them so that other procedures can correctly process the results. Examples of procedures that the FindList uses are: selection of default values and selection of the information that has to appear in the presentation of a list of items.

### 5.2.2 The processes involved in task management

Task management involves two processes. As shown in Figure 5.7, the first stage is the supervision and completion of the task (that is, to complete the content of the ActualTD). The second stage consists of accessing the web service and processing the results according to the content of the ActualTD.

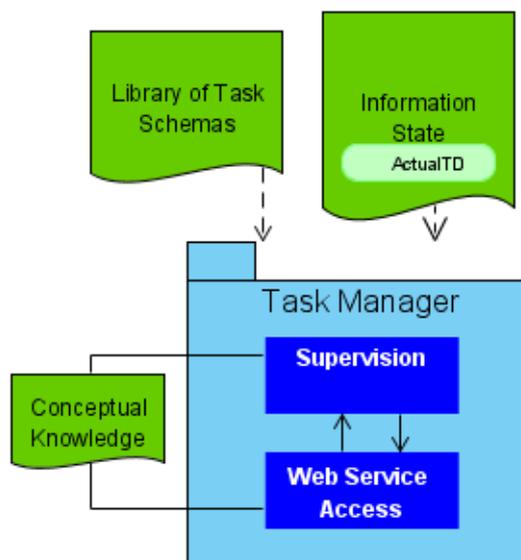


Figure 5.7: The task manager architecture

Figure 5.8 shows the workflow for the TM. During the communication, and once the user's turn has been interpreted, the DM identifies which service and task the user wants to execute. Then, the TM instantiates and loads the proper TS for the particular task. Next, the TM updates the ActualTD content, considering the data that appear in the dialogue (ActualTD) and the TS of the task. If possible, the system accesses the service using the data given for the particular task. Finally, it obtains the results.

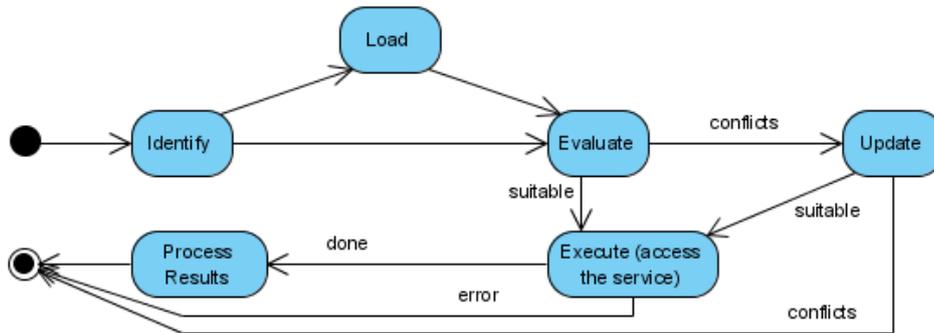


Figure 5.8: The execution flow of the task manager

Dialogue 5.3 serves to illustrate the TM stages: identify, load and update the task. In this dialogue, the user is looking for concerts in the Stadium. The TM needs to update the constraints given by the user to give a suitable response.

#### Dialogue 5.3

---

S1: Welcome to the Automatic Platform. How may I help you?

U1: Which concerts are there in the Stadium on Saturday?

S2: OK. There are no concerts in the Stadium on Saturday. There are several concerts on Saturday in other places. ...

---

#### Identify the service task

Once the communication has started and the first turn of the user has been interpreted, the DS has to identify the task and the service that the user wants to access. The TM receives the data from the NLU module and the dialogue context. To identify the service, the system needs to know which services are available and how to differentiate them.

As described in Chapter 3 in relation to the system architecture, knowledge about the domain of the application is represented in a general repository. This general conceptual

base, named conceptual knowledge (CK), contains information about the service domain concepts, the attributes and the relations among them. The domain concepts and the attributes are related to the parameters in the service tasks. They are also related to the semantic representations obtained from the NLU. These bidirectional relations enable the TM to easily identify the task that the user wants to use.

Figure 5.9 represents the connections generated among the ActualTD, the CK and the TS for Dialogue 5.3. The ActualTD contains the interpretation of the user sentence (U1), which is obtained after the execution of some processes in the NLU and the DM. The FindList CA class represents a simplification of the TS of the task. It contains all the parameters related to the task. The rest of the boxes represent the information stored in the CK, which enables the connection between the TS and the ActualTD.

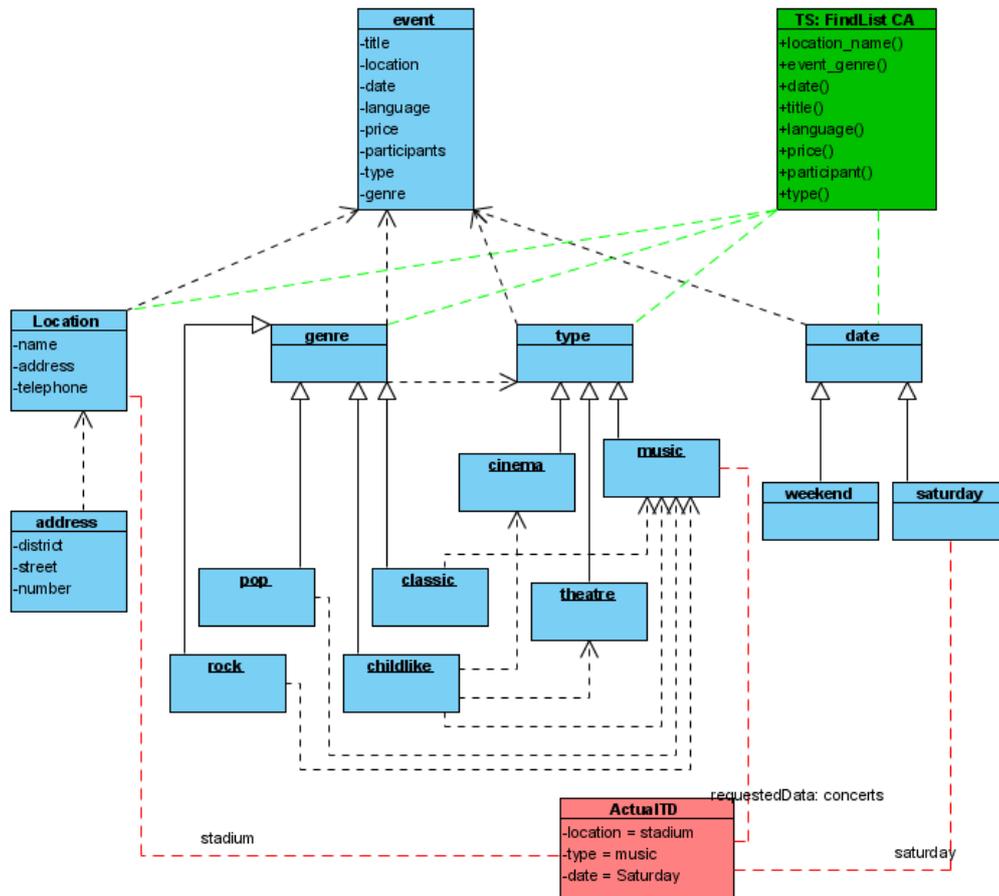


Figure 5.9: Information used by the TM when processing Dialogue 5.3

### Load the service task

Once the TM has identified the web service and the specific task that the user wants, it loads all the related information into the data structures. This process consists of two steps. First, the TM identifies the type of task and creates the corresponding object instance: SubmitForm, ObtainData or FindList. Then, the data related to the attributes of the task are incorporated into the instance. As a result, this process generates an specialization of the data provided (in the ActualTD) for the task that is being executed (the object instance of the task). These data consist basically of the task parameters and their constraints. For example, Figure 5.10 shows the specialization of the ActualTD for the U1 sentence in Dialogue 5.3. The system binds the FindList schema for the task FindList\_CA with the particular content of the ActualTD.

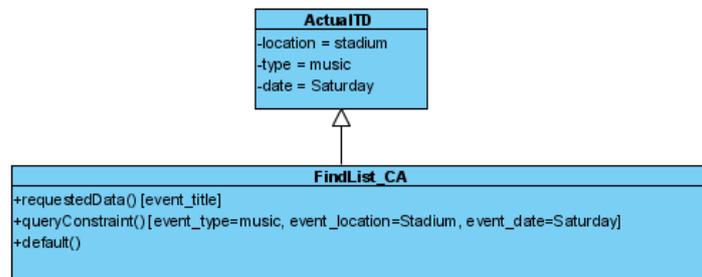


Figure 5.10: Example showing the data used by the TM

### Update the data and access the service

The ActualTD is updated at each turn of the dialogue. The information is updated following these sequential steps:

1. Evaluate the ActualTD. The TS defines the attributes that the user must provide to execute the task. If not all of them are available, then the user is asked for the missing data. In addition, the TS defines constraints among attributes. These constraints apply at different levels: attribute value, attribute relations or overall task. The TM has to consider all these constraints to update the parameters.
2. Update the parameters. The evaluation of the ActualTD can trigger a set of constraint conflicts. The use of different mechanisms updates the values of the parameters, for instance it relaxes or deletes them. The function of these mechanisms is to solve constraint conflicts and enable the execution of the task.

3. Execute the task. The result of the task execution is classified according to the number of results. If there are too many results, the system has to suggest to the user how to step up the query's constraints. If there are no results, the system has to decide how to relax the constraints.

The next dialogue actions depend on the data resulting from the execution of the TM. We divide these actions into four (non-exclusive) groups: request the user for more information, given that there are missing data; handle non-compatible constraints when the value of at least one attribute is not compatible with a constraint in the task specification; update the constraints when the system has been executed but the value of at least one attribute needed to be changed; and show the results if the service was correctly accessed.

For example, as we saw in Dialogue 5.3, the TM executes the query taking into account all the information given by the user. However, as no results are obtained, the TM updates the constraints (relax) and then executes the service again. With these new constraints, the system finds several results, namely concerts in other places on Saturday, and shows these results.

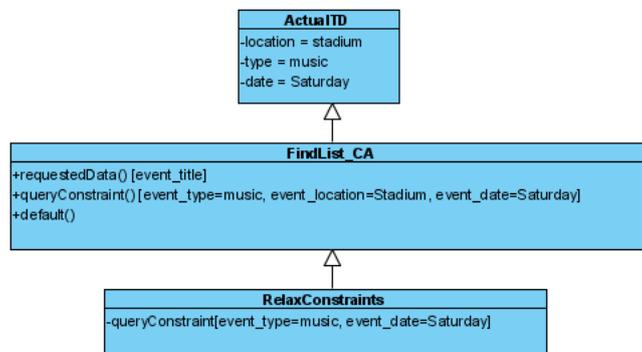


Figure 5.11: Example showing the updated FindList\_CA

Figure 5.11 illustrates how the content of the ActualTD is updated after processing the U1 sentence in Dialogue 5.3. This process produces the system sentence S2 in Dialogue 5.3. In this example, the first task execution does not produce any results. Next, the TM relaxes the constraints, deletes the values for the parameter location and executes the task again. In this case, the execution returns a set of results that can be classified according to different parameters, e.g. the location or the genre. This produces the second part of sentence S2 where

the system gives alternative results. As we saw in Chapter 4, in this example the dialogue manager decides to ask the user for the genre that he/she is interested in.

Another example is shown in Dialogue 5.4, in which the data given by the user are considered both incomplete and incompatible with the required parameters. A task constraint is broken because the washing machine cannot be collected, since it is pollutant and must be delivered at a GP (incompatible). In addition, the system cannot access the collection task because the user's address is needed (incomplete).

#### Dialogue 5.4

---

S1: Welcome to the Automatic Platform. How may I help you?

U1: How can I get rid of my washing machine?

S2: I am sorry, electrical appliances cannot be thrown away on the street. You may drop it off at a Green Point. Tell me the address.

---

### 5.2.3 The algorithms that process the tasks

As indicated in the previous section, task processing requires knowledge of the specific TS of the task and the ActualTD. Every task of a specific type shares common features and can be processed and updated following the same schema. Our goal is to decouple the specific data of the applications contained in the task schemas from the processes in the task manager. To achieve this goal, we developed the task structures (Section 5.2.1) that cover the range we considered and facilitate the deployment of new services in the dialogue system.

This section describes the specific algorithms we have designed to process each type of task. These algorithms are based on the data structures used to store the specification of each task (the task schemas). Figure 5.12 gives the UML diagram for these data structures. In this design, the main structures are the *Parameter* and the *ParameterList*. The rest of the structures correspond to the functions of the parameters and the relations among them.

Concretely, the *Parameter* is the piece of data that can be obtained from the user and holds the constraints that can be applied to it. The *ParameterList* is a list of parameters. In the SubmitForm tasks, it is used to list the complete set of required input parameters.

In the FindList tasks, the *ParameterList* has a set of parameters that the user can ask for (*requestedData*). For example, in our implementation of the CA service, the *ParameterList* has the parameters event, location and date. As shown in the UML diagram, the *ParameterList* can also hold a list of constraints over all the *Parameters* in the list. The *Condition* represents a constraint related to one parameter. In our implementation, these are Boolean expressions that are defined according to their values and relations with other parameters. For instance, the *requestedData* cannot contain the same parameter as the *queryConstraint*. The *queryConstraint* is used in the FindList and ObtainData tasks. It contains the list of the allowed set of *Parameters* that can be used to constrain a search. Hence, the *queryConstraint* is a list of *ParameterList*, in which each *ParameterList* is the minimum set of *Parameters* needed to obtain values from the service.

The study of the services discussed in Appendix B also gives the specific content set for each service task.

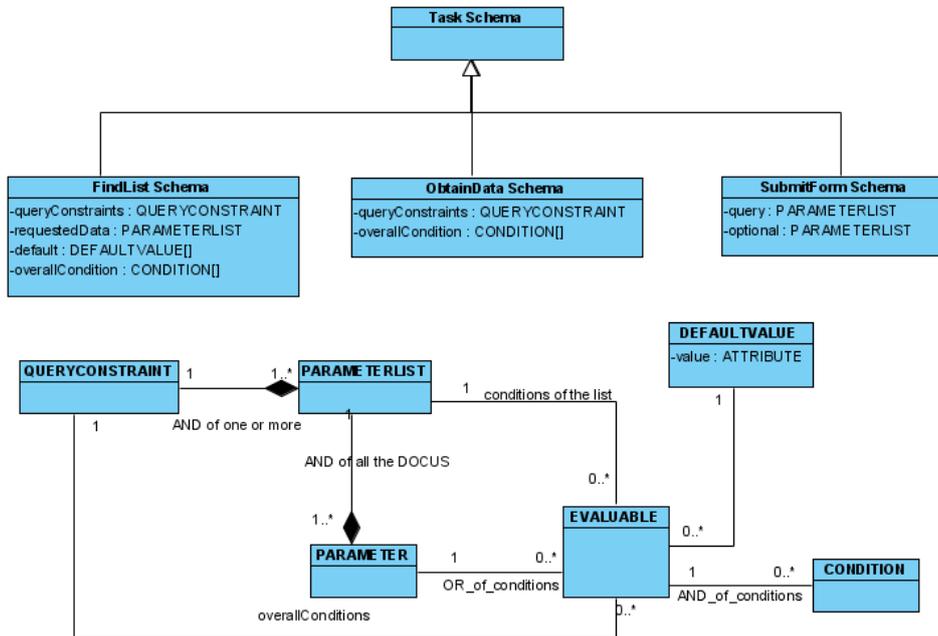


Figure 5.12: UML class diagram of the task schemas

### The algorithm for SubmitForm tasks

Figure 5.13 shows the pseudocode for the SubmitForm algorithm. When all the parameters in the list have been obtained, the system evaluates their values. If no parameter is missing

and no constraint is broken, then the system evaluates the conditions related to the task. At this point, the system can update or remove parameters (in particular, the optional ones), if needed. The current algorithm consists of evaluating mandatory parameters and then optional ones, and removing those that are not suitable. When unsuitable parameters are removed, a new set of parameters is obtained and the system has to evaluate them again. When a list of parameters is suitable for executing the task, the system accesses the service and submits the transaction. Finally, the updated set of parameters and the results are returned to the system. The next actions of the system might show the changes in the constraints.

**Algorithm** ProcessSubmitForm (TS, ActualTD)

```

set c ← getConstraints ( TS, ActualTD )           // obtains the parameters
repeat
  set qc ← c
  for each parameter (P) in ParameterList (PL)    // checks the required parameters have been obtained
    check_if_missing( qc, P )
  end for

  set pl ← {}
  if ( !missings )
    check_if_conflicts( qc, PL )                   // checks conflicts related to the ParameterList
    if ( !conflicts ) set pl ← combine( qc, PL )   // combines the qc-values & the PL-parameters
  end if

  if ( !Empty( pl ) )                             // checks conflicts in the TS & updates/removes incompat.
    set p ← check_overall_conflicts( pl, qc, TS, ActualTD ) // generates a new qc
  end if
until ( qc == c )                                // a list of parameters was correctly obtained

if ( !Empty( pl ) )
  set result ← execute_SubmitForm( pl, TS )       // executes the SubmitForm
end if

update control attributes ( result, pl )          // returns the control to the system

```

Figure 5.13: Algorithm for processing SubmitForm tasks

### The algorithm for ObtainData tasks

The ObtainData task has the *queryConstraint* attribute which consists of a set of *ParameterList*. To execute the ObtainData task, the algorithm looks for at least one list of parameters that is suitable for executing the task. Every constraint related to the parameters is also evaluated using the data content in the ActualTD. In particular, the ObtainData algorithm, as shown in figure 5.14, processes these data. The current algorithm implementation

evaluates the sets of parameters sequentially and executes the query using the first suitable *ParameterList* found. The following parameters are not evaluated.

**Algorithm** *ProcessObtainData* (TS, ActualTD)

```

set c ← getConstraints ( TS, ActualTD )           // obtains the parameters
repeat
  set qc ← c
  for each parameter_List (PL) in constraints (QC) // evaluate the list of parameters in the queryConstraints
    for each parameter (P) in ParameterList (PL) // checks the required parameters have been obtained
      check_if_missing( qc, P )
    end for

  set pl ← {}
  if ( !missings )
    check_if_conflicts( qc, PL )                 // checks conflicts related to the ParameterList
    if ( !conflicts )
      set pl ← combine( qc, PL )                 // combines the qc-values & the PL-parameters
      break for                                 // a suitable list of parameters is found
    end if
  end if
end for

if ( !Empty( pl ) )
  set c ← check_overall_conflicts( pl, qc, TS, ActualTD ) // checks conflicts in the TS & updates/removes incompat.
  // generates a new qc
end if
until ( qc == c )                                // a list of parameters was correctly obtained

if ( !Empty( pl ) )
  set result ← execute_ObtainData( pl, TS )      // executes the ObtainData
end if

update control attributes ( result, pl )         // returns the control to the system

```

Figure 5.14: Algorithm for processing *ObtainData* tasks

### The algorithm for *FindList* tasks

The *FindList* task has the *queryConstraint* attribute, which consists of a set of *ParameterList*, and the *requestedData* attribute, which is a single list of parameters. Every *ParameterList* in the *queryConstraint* represents a minimum set of parameters to query the service, whereas the list of parameters in the *requestedData* represents the list of items that the service can return. To execute the *FindList* task, the algorithm looks for at least one list of parameters that is suitable for executing the task. The *FindList* algorithm is shown in figure 5.15.

The current algorithm implementation evaluates the sets of parameters sequentially and executes the query using the first suitable *ParameterList* that is found. The relaxation rules

are applied for a specific set of parameters. Then, the algorithm processes the parameters for the *requestedData*. These are then evaluated and updated, taking into account the selected *queryConstraints*. Finally, when the *requestedData* is also suitable, the *FindList* can be executed.

**Algorithm ProcessFindList** (TS, ActualTD)

```

set c ← getConstraints ( TS, ActualTD )           // obtains the parameters for the query constraints
repeat
  set qc ← c
  for each parameter_List (PL) in constraints (QC) // evaluate the list of parameters in the queryConstraint
  for each parameter (P) in ParameterList (PL) // checks the required parameters have been obtained
    check_if_missing( qc, P )
  end for

  set pl ← {}
  if ( !missings )
    check_if_conflicts( qc, PL ) // checks conflicts related to the ParameterList
    if ( !conflicts )
      set pl ← combine( qc, PL ) // combines qc-values & PL-parameters
      break for // a suitable list of parameters is found
    end if
  end if
end for

  if ( !Empty( pl ) ) // checks conflicts in the TS & updates/removes incompat.
    set c ← check_overall_conflicts( pl, qc, TS, ActualTD ) // generates a new qc
  end if
until ( qc == c ) // a list of parameters was correctly obtained

set r ← getRequested ( TS, ActualTD ) // obtains the parameters for the requested data
repeat
  set qr ← r
  set pl_r ← {}
  check_if_conflicts( qr, PL ) // checks conflicts related to the ParameterList
  if ( !conflicts ) set pl_r ← combine( qr, PL ) // combines qr-values & PL-parameters

  if ( !Empty( pl_r ) ) // checks conflicts in the TS & the queryConstraint, & updates/removes in
  set r ← check_overall_conflicts( pl_r, qr, TS, ActualTD, pl ) // generates a new qr: relax and/or use default parameters
  end if
until ( qr == r ) // a list of parameters was correctly obtained for the requested data

if ( !Empty( pl ) && !Empty( pl_r ) )
  set result ← execute.FindList( pl, pl_r, TS ) // executes the FindList
end if

update control attributes ( result, pl, pl_r ) // returns the control to the system

```

Figure 5.15: Algorithm for processing FindList tasks

### 5.3 Summary

Our DM uses communication plans to represent the steps that must be followed to access web services. These plans are used to detect the user intentions, guide the dialogue interaction, query the application, and decide on the system responses. We have studied the most appropriate representation of the application knowledge (tasks) to facilitate the generation of these plans, and the design and development of the processes related to them. We have defined the general schema for two types of web services, and the task models for the tasks that they involve. In addition, we have used this method to integrate two different applications into our dialogue system, which improved the engineering process of the whole system. The resulting architecture facilitates the integration of other applications into the system, since the task models can be easily extended and adapted. Besides, new tasks models could be defined for other types of applications.

There is room for further improvements in the task management. For instance, the algorithm to generate the summary of results has not been designed, and the algorithm to relax the constraints and the selection of the default values are simple rules. We have focused on the task structure rather than the domain knowledge, which can be studied in more detail. Hence, there are several future lines of study. We believe that the interaction in information-seeking services could still be more efficient with a richer domain knowledge representation, so that the results obtained after the search can meet the user's needs. This is an interesting research line, particularly if we extend the system so that it can also access other online services. In addition, the algorithms that process the tasks are actually very simple. Other systems use more general AI techniques, such as reasoning, planning or constraint resolution. Finally, other strategies used to accelerate the dialogue design can be studied, such as the system described in [D'Haro, 2009]. In this thesis we have focused on the structure of the tasks, but other strategies can be designed that take into account the contents of the applications.

# Natural language modules

---

One of the challenges in developing DSs for multiple applications is to facilitate the generation and/or adaptation of linguistic resources for new applications and domains. The problem of adapting linguistic resources to new applications is even more challenging in DSs that support several languages. The most common way to adapt a DS to a new application is to generate the new linguistic resources that are needed to express the concepts and actions related to the new application.

However, application-restricted resources are expensive to develop and difficult to reuse. For this reason, many approaches isolate the different types of knowledge that are involved in the communication: the domain knowledge, the knowledge that controls the interaction and the linguistic knowledge. An appropriate representation of this knowledge facilitates the adaptation of the system to different applications, languages, modes of communication and types of users. In most well-known DSs, the linguistic knowledge and domain specific restricted knowledge are presented separately from the general processes involved in communication, i.e. analyzing the user's sentences, determining the next actions and generating the system's responses. Several publications on preliminary versions of the system studied the types of knowledge and how they can be represented to improve different aspects of the communication process (e.g. [Gatius and González, 2004a], [Gatius and González, 2005b], [Gatius and González, 2005c]).

Our DS design can support two modes of interaction: speech and text mode (see figure 6.1). Since we wanted to focus on dialogue and task management, we only developed the text mode subsystem. The resources used by the text-based subsystem are similar to those used in

the VoiceXML standard. Thus, they can be easily reused and integrated into voice-dependent components. Concretely, the text analyzer that we have integrated uses grammars and lexicon structures that are similar to those used in VoiceXML applications, following the standard SRGS. Technically, the analyzer models the speech recognizer when it performs perfectly. This approach was published in [Gatius et al., 2007a].

The DS also addresses multilingualism. The design of the linguistic modules is general and language independent, so as new languages can be easily integrated into the system. Although the current online implementation of the system is only available in Spanish, we developed similar resources for Catalan, English and Italian in a previous version of the DS that was developed for the HOPS project.

This chapter describes the two natural language modules included in our DS: the natural language understanding (NLU) and the natural language generator (NLG). Although the two linguistic modules have been specially adapted for the DS described in this thesis, they could be integrated into other systems.

## 6.1 Obtaining linguistic resources

Well-organized dialogues improve efficiency in communication because they help users to understand the information the service needs from them. For this reason, as seen in previous chapters, the plans and the tasks that are used in the dialogue are obtained from application and domain knowledge.

In our approach, the application knowledge is used to generate the dialogue resources (the plans) and the linguistic resources (grammars, lexicons and system messages) needed for both modes of interaction in different languages, [Gatius and González, 2005d]. As mentioned in previous chapters, the application knowledge involved in communication basically consists of the input and output parameters of the several tasks that the application performs plus other additional information, as constraints or data types. The application knowledge is modelled as data schemas with specific slots. Each slot contains the set of allowed parameters, which are described by attributes and the relations among them. These attributes correspond to the specific information about the parameter that is requested from (or given to) the user. Hence, the dialogue plan and the information to be provided to the users are obtained from

the model that represents the application knowledge tasks.

The linguistic resources associated with the attributes that represents the information that is required in several applications (e.g., dates and addresses) are easily identified and reused across applications. For the rest of the application's attributes, specific linguistic resources are generated. The reason is that the use of application-restricted lexicons and grammars improves accuracy and efficiency, specially through the speech channel, because the restricted resources reduce the amount of possible interpretation in the ASR. In addition, restricted resources avoid the production of interpretations that are inconsistent with respect to the application specification.

Domain knowledge can also be used to generate the linguistic resources adapted to a specific application. For example, in our system we used a taxonomy of furniture, which was extracted from IKEA<sup>1</sup> to build the lexicon for the LOC service. At run time, this taxonomy would have been used to detect whether specific furniture is pollutant or not, but unfortunately this information was not included.

Moreover, domain knowledge can be used to avoid asking the user difficult questions whose answer can be inferred from the conceptual representation. For instance, domain knowledge helps the system to detect hypernyms or hyponyms in the user's interventions (as was described in [Milward and Beveridge, 2004]), particularly if this knowledge is represented in taxonomies or ontologies.

## 6.2 Natural language understanding

The NLU module generates a computational interpretation of the user's turn in two phases: the analysis and the interpretation. Figure 6.1 shows the data structures and the components involved in processing the user's turn.

The analyzer produces a semantic representation of the user's turn and the interpreter completes it by considering the domain knowledge and the dialogue context (basically, information related to the application task that is being processed). Both the analyzer and the interpreter are independent modules. The analyzer has been adapted from a previously existing parser (described in [Gatius, 2001]) and the interpreter has been specifically developed

---

<sup>1</sup><http://www.ikea.es>

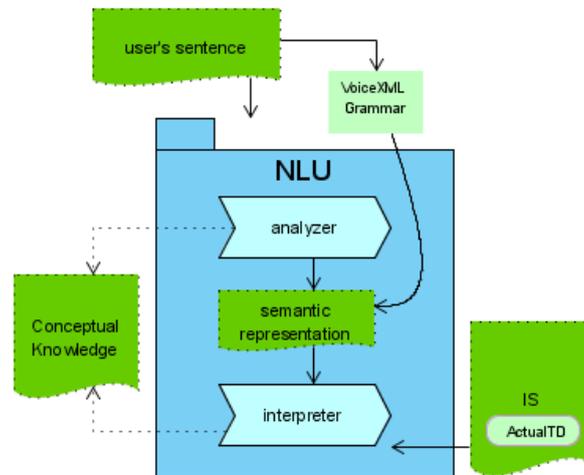


Figure 6.1: The natural language understanding architecture

for our DS.

The main reason for incorporating the analyzer was that it had been designed to process user's turns when they interact with different applications. The analyzer uses semantic grammars and lexicons. It has been used in a prototype in which a grammar and lexicon were generated semi-automatically from the application's conceptual representation. It proved more efficient than using general linguistic resources.

As mentioned before, the HOPS-DS supports text (through the web) and voice (over the telephone) input. Similar application-restricted grammars that incorporate conceptual knowledge are used by the two modes. A VoiceXML-based front-end processes the voice mode. The formalism of the application-restricted grammars and lexicons can be easily converted into VoiceXML grammars. The difference is that the output of the voice front-end is not processed by the analyzer, but the semantic representation is directly passed to the interpreter. This facilitates the use of common resources for the two modes that are supported in our design.

The two differentiated steps involved in processing the user's turns are explained below, following the example in Dialogue 6.1. In this example, a user calls from the university and wants to arrange for a huge amount of paper to be collected.

#### Dialogue 6.1

---

S1: Welcome to the Automatic Platform. How may I help you?

---

Dialogue 6.1 - Continues on next page

Dialogue 6.1 - Continued from previous page

---

U1: I want to arrange for paper to be collected from the University.

S2: OK. What is the approximate volume of paper in cubic metres?

U2: About 30 cubic metres.

S3: What is the collection address?

U3: Diagonal 30.

---

In S2, the system asks the user to state the volume of paper to collect. The semantic analyzer returns the following representation of the input:

$$(6.1) \ [number = 30, unit = m3]$$

Then, the interpreter builds the representation shown below:

$$(6.2) \ [answer[volume = 30]]$$

The interpreter infers from the context of the previous system turn (S2) that the number given by the user corresponds to the volume of the paper to be collected, which is a concept that is related to the task under execution.

Next, the system asks the user for the collection address. The semantic representation of the user intervention in turn (U3) is:

$$(6.3) \ [street = diagonal, number = 30]$$

Finally, using the context knowledge the interpreter completes the representation of the user turn:

$$(6.4) \ [answer[address[street = diagonal, number = 30]]]$$

In this case, the number given by the user belongs to a more complex structure, the address, and it is related to the street name.

### 6.2.1 The analyzer

The syntactic-semantic analyzer is a left-corner parser with top-down filtering that is written in Prolog. The parser is domain and language independent. It performs the syntactic and semantic analyzes in parallel using the specific domain knowledge that is encoded in the grammar and lexicon. The main advantage of this approach is that semantic considerations can be used to reduce the number of possible syntactic structures to be considered. In order to integrate the semantic analysis into the parser, the application knowledge is associated with the grammar rules and the lexical entries. The semantic analysis is based on lambda calculus. The semantic information is associated with each rule to indicate the order of interpretation of its constituents. The meaning representation for a sentence is created once all its constituents' parts have been recognized. As shown in figure 6.2, in order to generate the semantic representation, the analyzer needs to know the user's sentence, the language being used and the information expected by the DM (i.e. the system's actions). The parser selects the proper linguistic resources (language-specific grammars and lexicons) as long as the language identification is established earlier in the dialogue.

The main contribution of the present work in relation to the parser is the development of the linguistic resources in different languages. Nevertheless, more detailed information about the parser, several tests and results can be found in [Gatius and González, 2006a].

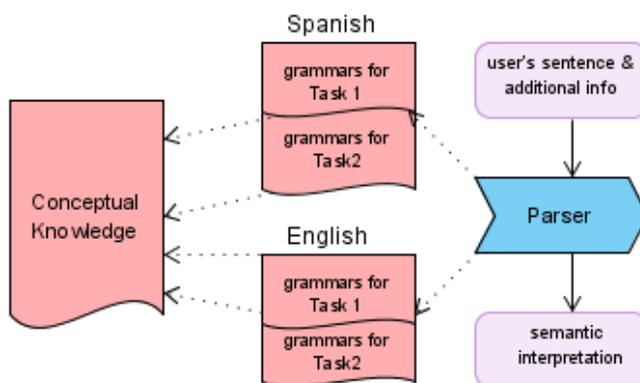


Figure 6.2: The data used by the analyzer

The language-specific grammars are built from the description of the application tasks and their parameters. Separate grammars and lexicons are developed for each language. In order to improve the efficiency of the process we incorporated dynamic entries, in which

representation and associated semantic representation are set at run-time. The use of dynamic entries reduces the number of lexical entries to be considered and allows the user to easily introduce new values. There are two different types of dynamic entries: those representing open values, such as numbers and dates; and those associated with a list of values, such as location names or object names. These entries can easily be changed when the application is updated.

Besides, the parser incorporates a garbage mechanism that eliminates all the words in the sentence that are not covered by the linguistic resources. This mechanism enables the parser to deal with spelling mistakes or sentences that are not covered by the grammars. However, it decreases efficiency, because it introduces ambiguity. To limit the number of possible interpretations, the parser uses knowledge from the dialogue context. Specifically, the parser knows the values that the DM expects to obtain from the user. The information related to the last system turn is used to give priority to the grammar rules in relation to the questions and answers of the system. As a result, the analyzer returns a single interpretation of the user's intervention based on the context of the dialogue.

To figure out the general performance of the parser, we analyzed the data from the corpus we collected during an evaluation (see Chapter 7 for further details about the implementation of the parser and the corpus). Table 6.2 shows the average number of turns per dialog, the average number of parser errors, the average ratio of parser errors per turn and the percentage of errors that were due misspellings. As can be seen, open dialogues tend to be longer than directed ones; and the parser performs better under directed mode than under open mode as indicates the number of errors per dialog and also the ratio of errors due to misspellings. The reason is that under open mode the user tends to give longer sentences, which are more difficult to parse. Instead, under directed mode, the user answers using fewer words or even just keywords without context, which also provokes ambiguities in specific situations. The adaptive module uses open mode unless too many errors occur, in which case it switches to directed mode (it happens in 8.0% of the turns). When using the A mode, it is possible to reduce the number of turns of the dialogues and the number of errors with respect to the UI mode.

Initiative	Num. dialogues	Avg. turns per dialogue	Avg. parser errors/dial.	Errors/turn	Spelling
Directed	52	9.4	0.85	0.07	10.3%
Open	14	15.1	3.43	0.20	13.3%
Adaptive	66	9.7	1.71	0.17	2.9%

Table 6.2: Parser Errors

## 6.2.2 The interpreter

The interpreter generates the final representation of the user's turn. As shown in figure 6.3, it disambiguates and completes the semantic interpretation of the user's sentence using the information about the dialogue context (information state), the conceptual knowledge and the schema of the actual task. The interpreter also adds the dialogue move associated with each piece of information in the interpretation.

For example, the sentence U1 in Dialogue 6.1, “*I want to arrange for paper to be collected from the University*”, generates a dialogue move “ask”, which contains the identification of the task, and a dialogue move “answer” that contains the rest of the data (shown in (6.5)). The turns U2 and U3 generate a dialogue move “answer” that contains the semantic interpretation.

(6.5)

`[ask[taskid = collection], answer[location = university], answer[throwingobject = paper]]`

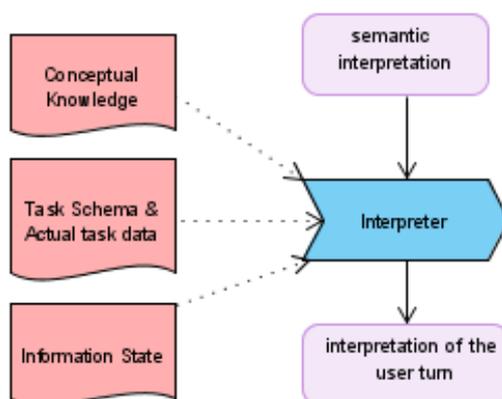


Figure 6.3: The data used by the interpreter

The application knowledge is represented in the CK as a set of concepts described by their attributes. In order to obtain the information from the user's turn required to execute the query (i.e. the identifier of the web service's task and its parameter values) some correspondence must be found between the concepts and attributes of the application task and the lexical entries used to express them.

For example, in the semantic representation (6.2) ("*About 30 cubic metres*", from turn U2 in Dialogue 6.1), the data related to the task state that volume is an attribute of the task "collection" (Chapter 5). The conceptual knowledge about the application states that the value of the attribute "volume" has to be a number (Appendix C). And finally, the context of the application states that the system asked the user to introduce the volume (a parameter of the task). Hence, the integer value given by the user is related to the attribute associated with the task under execution, which is waiting for a value of integer type.

Finally, the interpretation of the user turn is extended with the dialogue move. This last phase of the interpretation facilitates the integration of the NLU output into the DS's information structures, which is performed by specific rules in the DM. Among the types of moves that the DM uses (described in Chapter 4), the interpreter can only identify the subset representing the user actions:

Greet: initiates the communication and loads the initial plan.

Asking: indicates that the user is raising an issue. The task schemas describe the possible task (questions) that the user can ask (raise); and the plans indicate the actions that the system must follow to solve them. Each asking move of the user corresponds to an issue that the DM must solve; and an issue corresponds to the web services tasks.

Answering: corresponds to the values given to the input parameters of the task.

Feedback: indicates understanding or acknowledgment of the previous turn. It can take one of the following values: positive, negative or interrogative (more elaborated feedback, such as retraction or confirmation of the system turn).

Pagination: allows browsing a list of elements.

Quitting: indicates the desire to finish the communication.

### 6.3 Natural language generation

The NLG module outputs a natural language sentence that is based on the content of the system's turn. This representation is given by the DM and augmented with additional information, as the degree of initiative provided by the Adaptive module. The NLG process consists of two phases: first, the response generator decides what to say to the user (using the DM output) and in which order the information has to be communicated. Then, the surface generator provides the final sentences that express the system's intervention. Figure 6.4 illustrates the NLG architecture and the knowledge involved in its processes.

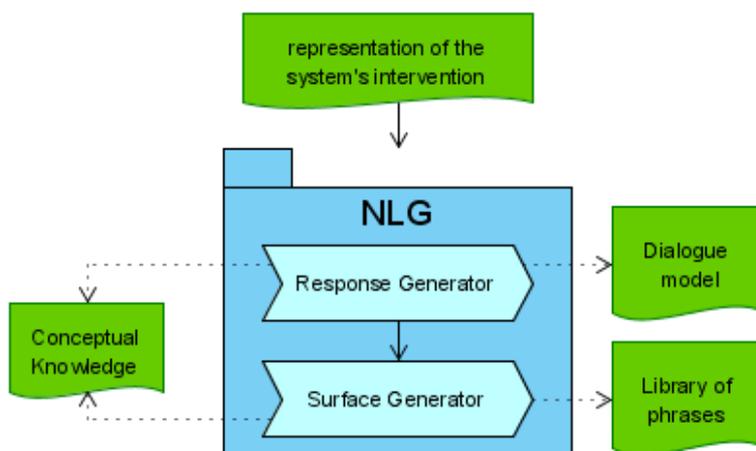


Figure 6.4: The natural language generator architecture

During the development of the general linguistic resources needed to generate the system's responses we collaborated with a linguist. We adapted the syntactic-semantic taxonomy described in [Gatius, 2001], and inspired by the generalized upper model of [Bateman et al., 1994], to semi-automatically generate all the phrases that the system needs to express the communication moves required for the specific applications we implemented. When the DS is adapted to a new web service, all the linguistic resources involved in the service tasks are automatically generated in all the languages that are supported and manually supervised. The generalized upper model was a generalization of a previous upper model, described in [Bateman et al., 1990], to different languages that organizes the information for supporting natural language processing requirements. The KPLM system, described in [Bateman, 1994], is an example of the application of this model to the NLG task.

DIGUI produces messages in three languages: Catalan, Spanish and English. A preliminary

version of the system produced also messages in Italian. The appropriate representation of the linguistic structures may limit the language structures that are generated when the system is adapted for a new service. For each language, each class in the syntactic-semantic taxonomy is associated with the linguistic structures (patterns) that represent the communication moves: ask the user for specific data, explicitly or implicitly ask for confirmation of a value and present results. The patterns contain semantic, syntactic and morphological information.

When the system is adapted to a new service, the task's parameters must be classified according to the syntactic-semantic taxonomy. Each class is associated with the linguistic structures needed for its realization (i.e., attributes in the class of\_quantity are associated with structures expressing a value and, optionally, its unit). Then, for each language supported, each class has to be linked to the proper lexical entries. Finally, the system messages are automatically generated in each language by combining the general patterns associated with each taxonomy class with the particular lexical entries.

The phrases that are generated are stored in libraries. Each one of the generated phrases expresses a particular move and a particular task parameter, although several task parameters appear in more than one service, such as telephone, address and date. Sentences for these parameters are reused across services.

### 6.3.1 The patterns

The general patterns represent several linguistic forms in which the dialogue moves can be expressed. Hence, we can generate different phrases using different patterns to express the same meaning in different forms, which makes the system more flexible and friendly. We also classified the patterns into two categories of initiative:

Open: the pattern generates an open question/phrase.

Directed: the pattern generates a directed question/phrase.

Mid: the pattern generates a question/phrase that can be used indistinctively for the two modes of initiative (Open and Directed).

This classification helps the system to select the most appropriate phrases to follow a specific mode of initiative. During the second phase of the NLG, the system selects the specific phrases that will appear in the system's response. The NLG also has mechanisms to avoid repeating the same phrases (i.e. it does not use the same phrases when it needs to repeat a

question because the user did not answer). We used paraphrasing and synonyms to obtain more sentences and to enable the system to be more flexible.

Figure 6.5 shows the English patterns for asking about the type of the event. Figure 6.6 shows an excerpt of the total set of 96 phrases that are generated for the communication move “ask-ca.event.type”<sup>2</sup> using the English lexicon and the concepts and attribute taxonomy associated with the task.

Additionally, a set of 20 patterns are related to a specific move, but unrelated to any parameter or particular service, although they are used by all of them. We addressed these moves separately and linked them directly to their linguistic representation. Thus, they do not have to be generated for each application. Those moves are shown in figure 6.7.

The Spanish resources consist of 32 different rules implemented using 336 patterns and the linguistic expressions for 81 concepts, attributes and values. After processing the patterns, the generated resources consist of 254 different dialogue moves expressed by 2466 phrases and 242 lexicon entries.

In order to figure out the appropriateness of the messages, we marked the ambiguous messages appearing in a corpus we collected during an evaluation of the system. As shown in table 6.3, the percentage of inappropriate messages under the open mode is higher than under the directed mode. Nonetheless, the messages under the adaptive mode adjust better to the communication needs, as the lower percentage of inappropriate messages indicates.

<b>Initiative</b>	<b>Avg. turns</b>	<b>Avg. inappropriate messages</b>	<b>% inappropriate messages</b>
Directed	9.4	0.33	2.72%
Open	15.1	0.86	3.72%
Adaptive	9.7	0.24	2.22%

Table 6.3: Appropriateness of the messages

## 6.4 Summary

The natural language components of DIGUI deal with various domains in several languages. We aimed to design components that can: tackle multilingualism; be used in voice mode of interaction; be easily adapted to other applications and domains.

<sup>2</sup>This move corresponds to the action of asking about the type of the event in the CA service

```

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]?)
            (What <event_type> [verb_wish_q](: [music], [theatre], [cinema], [art] or [appointment])?)
            (What <event_type> [verb_wish_prep_q](: [music], [theatre], [cinema], [art] or [appointment])?)
  Directed ([verb_request] [art] <event_type> [verb_wish_s])
  Directed ([verb_request] what <event_type> [verb_wish_s])
  Directed ([verb_choose] [art] <event_type>: [music], [theatre], [cinema], [art] or [appointment])

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

[q-word] --> what | which
[verb_copulative_sg] --> is
[art] --> art
[verb_wish_s] --> you are looking for | you are interested in | you have in mind | you want to see

```

Figure 6.5: The rule *ask* < *event\_type* >, pattern for the class *event\_type* and several lexical forms

```

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 6.6: Some examples of phrases for *ask* < *event\_type* > move of total 96 phrases

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
Option_Statement -->	You can choose one or more options
Goodbye -->	Goodbye   Bye
Confirm-User_sentence -->	You said [@user_phrase]

Figure 6.7: Messages associated with general unrelated moves

DIGUI incorporates a natural language parser to perform a deep syntactic and semantic analysis of the user's messages. The parser, which is language-independent and domain-independent uses domain-restricted grammars, lexicons and taxonomies.

The natural language generator uses semi-automatically generated phrases for each language that the system supports. To generate the most appropriate system messages for a specific service, the generator component uses a syntactic-semantic taxonomy, which relates the parameters of a specific service task to the linguistic patterns that represent different dialogue moves.

Moreover, these linguistic patterns cover two types of interaction: open and directed. Open messages are more appropriate for expert users (user-initiative dialogues), while directed prompts are more suitable for novice ones (system-initiative dialogues).

The generation of linguistic resources is a hard task that requires specialized skills. In our system, we have developed robust modules that can run with very basic resources. Indeed, the patterns and lexical entries developed for the language generator can be used to generate baseline for the domain-restricted grammars that are used in the processing module. Such

a mechanism would facilitate the process of developing the resources for a new domain or another language.

Moreover, we could enhance the communication and the user perception of the system by increasing the collaboration between the modules. For instance, the system could address coreference resolution or ellipsis in its message generation. In the current implementation this issue is only addressed in a few situations.

Finally, the system architecture has a modular and flexible design. This characteristic allows the system modules to be upgraded easily. As an example, the system performance could be improved by replacing the current natural language modules with more sophisticated ones.



# Implementation and evaluation

---

This chapter describes the implementation of the DIGUI prototype and the evaluation we carried out to study the main achievements of our design decisions.

We aimed to develop a friendly dialogue system in which it is easy to integrate new web services. The next sections in this chapter provide details about the implementation of each module of the system and the resources it uses. This gives an idea of the effort needed to develop the resources for a new web service. Moreover, we evaluated the degree of satisfaction of the users who interacted with the system. For this purpose, we collected a corpus of dialogues and several measurements of the modules' work, such as the parser errors, the initiative mode used and the system messages. Finally, we analyzed this corpus to assess the use of the adaptive initiative strategy and to plan the generalization of the task schemes.

## 7.1 Implementation

DIGUI can be accessed through the web site: <http://www.lsi.upc.edu/~nlp/digui/>. The basic configuration is a web chat that allows the user to access two services. A screenshot of the graphical interface is shown in figure 7.1. The main view of the chat has several links that display useful information: the description of the scenario (during the tests), the operations available in the LOC service, the locations and events available in the CA, contextualized help and some hints about the capabilities of the system. Moreover, besides the text area of the chat, there is a list of several basic commands: help, cancel, menu and bye.

The system was developed using Java J2EE 1.4, although the analyzer that we integrated was implemented in Prolog. Currently, the system runs on Java EE 5. The textual web chat

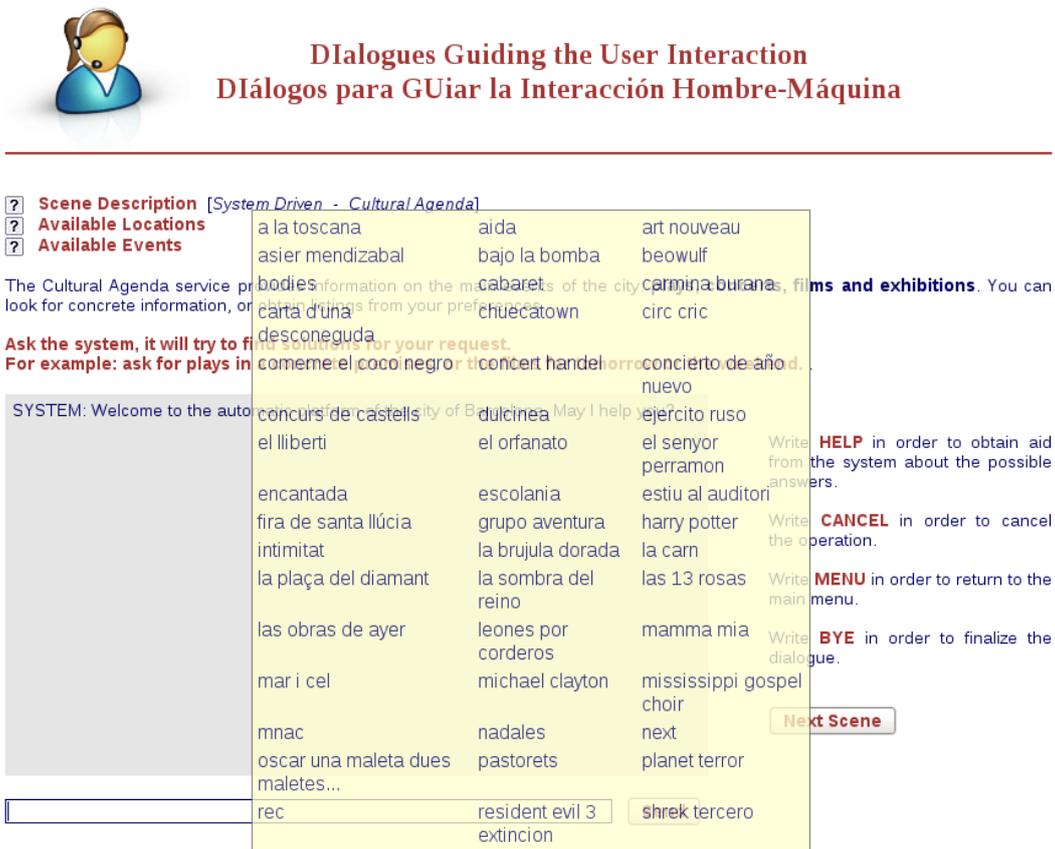


Figure 7.1: Screenshot of the DIGUI prototype

is a Servlet that runs on Tomcat 5.5. When a user accesses the chat, the system initiates a new session with the DS, displays the first system message and waits for the user inquiry. Figure 7.2 shows a schema of the DS deployment.

The analyzer is a Prolog program that runs on the server. It runs on a SWI-Prolog environment and uses the HTTP support library to communicate with the rest of the system. In other words, the analyzer runs independently of the rest of the system and can be accessed using a browser. The parser answers the HTTP requests with XML files that store the analysis results. Currently, the Spanish version of the NL resources is the most highly developed. Table 7.1 shows its components. The use of dynamic entries facilitates the introduction of proper nouns by the user. In addition, the dynamic entries are displayed in help screens during the dialogue, as shown in figure 7.1. The static resources, such as grammars and lexicons, are only loaded once when the parser is started.

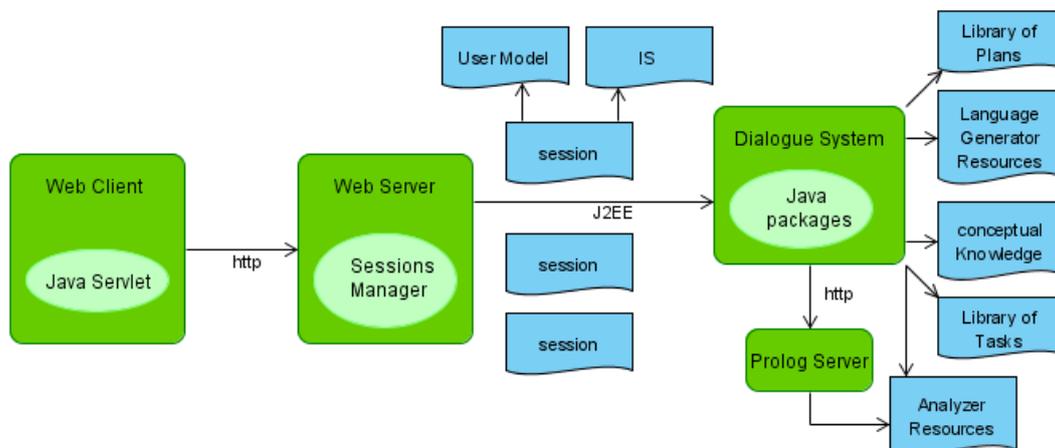


Figure 7.2: Schema of the DS deployment

Components of the parser	Amount
Grammatical rules (reachable as top-most elements)	253 (184)
Lexical entries (for both general and domain-specific grammars)	2,241
Lexical entries that express dates	67
Dynamic furniture names	266
Dynamic location names	1,005
Dynamic participant names	541
Dynamic street names	16,173
Dynamic title names	92

Table 7.1: NL-resources of the analyzer

The rest of the modules in the system are JAVA packages. When the DS initiates a new session it creates all the structures needed to store the session context, such as the IS and the user model. The permanent resources<sup>1</sup>, such as the plans, the conceptual knowledge and the task library, are only loaded once when the DS has been initiated.

The current version of the DM consists of 27 integration rules, 25 update rules (18 for plan execution and 7 for accommodation), 27 selection rules (which select the system's next actions) and 120 plan commands (39 actions in the CA, 33 in the LOC and 48 shared actions).

The conceptual knowledge is currently represented using a taxonomy that is written in plain text, in which the concepts are described by attributes. The two services use a total of 37 concepts. These are organized into trees according to the different types of relations (belongs-to, attribute-value and depends-on). For instance, *private* and *company* have a

<sup>1</sup>The corpus of dialogues that was generated and the linguistic resources used are available on the website: <http://www.lsi.upc.edu/~mgonzalez/digui/resources.htm>.

relation of type *attribute-value* with the concept *usertype*.

The TM uses a repository of XML files that contain the task’s specification of the web services. The current implementation consists of one XML file that represents the tasks of each service (described in Chapter 5). Additionally, different files contain information that is related to the domain concepts. Specifically, there are two files for general resources, three for CA resources and four for LOC service resources.

The NLG module generates a message that takes into account the information obtained from the task execution, the system’s response actions, the conceptual knowledge and the user model information. This message is embedded in an XML page and delivered to the text client, which in turn displays the message to the user. The NLG resources consist of the linguistic resources involved in generating the patterns and the language resources that are generated for the two services using these patterns. Table 7.2 shows the Spanish version of these resources.

Pattern generation:	Amount
Patterns related to dialogue moves	32
General syntactic rules	336
Structures related to concepts, attributes and values	81
Language resources generated for the two services:	
Set of system moves	254
Phrases for expressing the system moves	2466
Lexical entries for expressing the attributes’ values	242

Table 7.2: NL-resources of the surface generator

Appendix C gives a detailed description of the organization of resources, how they are structurally stored, their content, and how the system processes them.

## 7.2 The evaluation

In our evaluation of the system, we aim to reveal whether the system design and the adaptive module in particular helped us to achieve our main goals: friendly communication and portability. First, we compared performance when different dialogue strategies were used and when different types of services were accessed. Then, we measured the improvement in the system when the linguistic resources were adapted by experts using a corpus of domain dialogues. For this purpose, we carried out two separate experiments.

In the first experiment, the system messages used by the NLG module were generated automatically using the pattern, rules and lexicons. They were then analyzed and, with the collaboration of a linguist, the general patterns, the general grammar rules and the lexicons were adapted to the specific service tasks and their parameters. The grammar rules and lexicon used in the prototype include general ones plus those generated by the general patterns from the application specification, which were then manually adapted.

On completion of the first experiment, we obtained a corpus of dialogues. We used this corpus to develop the adaptive module and to improve the language resources, i.e. to correct the main failures in the parser and to extend the specific domain-restricted grammar and lexicon. We classified the system messages according to the degree of initiative and corrected inappropriate ones by updating the patterns and rules. The new set of messages was peer-reviewed (minor revisions were needed). The improved linguistic resources were then used in the second experiment.

As described in the next sections, we used the PARADISE framework to measure the overall system performance and to compare both experiments. Then, we annotated the system messages in the corpus, to compare the effect of the adaptive module.

### 7.2.1 Methodology

The web address of the DS was sent via e-mail to several university distribution lists. Web experiments enable more data to be gathered than in traditional laboratory-based experiments, at virtually no cost. As no trips or appointments are involved, voluntary participation is much easier to achieve. The main drawback of web tests is that the experimenters lose control over the environment in which the experiment is carried out, and the participants' anonymity may make it difficult to assure the quality of results. Hence, to encourage participants to reveal their e-mail addresses and to commit themselves to the experiment, we raffled an MP4 player.

We configured the system that was used in the experiments to store all of the data required for the evaluation: the user and system turns, the semantic representation of the sentences, the traces of the interpretation and the dialogue component, the system's moves, the initiative used, and the timing (the time spent by the analyzer and the DM components and the time needed by the user at each turn).

Access to the application back-end was simulated (we developed simple databases and

programs), even though these web services already exist. The main reason for this was to keep the behaviour of the services under control. For instance, when the LOC service was accessed, we simulated an error that registered the collection of objects at a fixed ratio of 1 : 3. We also stored the registered collections in files so that we could cancel a collection if any of the users attempted this operation. Obviously, in this case, we could not use the real application without the collaboration of the entity that runs the service. This problem does not apply to the cultural agenda service. However, we did not access real online CA services because further mechanisms, out of our scope, are needed to connect the system with them. For instance, a wrapper can be used to communicate with online services, and the system would benefit from it by automatically extracting the list of events and probably the task structure. Our CA service contains 60 different events, with information about their type, dates, schedules, locations, participants, prices, sales points and description. We wanted to use semantic web techniques to undertake this task, as we had studied the use of ontologies to facilitate the reuse of the system resources with good results. Unfortunately, the study and development of these techniques is beyond the scope of this thesis.

Fifty-three volunteers were recruited for the first experiment, but only 30 of them completed it. These volunteers were randomly divided into two groups: one group accessed the version using only open messages and the other group accessed the version using directed messages. Subjects were asked to access each of the two web services twice (LOC and CA). To assess satisfaction, participants were asked to complete a questionnaire, immediately after each dialogue on their immediate impressions of several features: their overall impression, the appropriateness of the system's interventions, performance, friendliness, usefulness and future use. Each subject accessed the two services in a random order. We collected 104 valid dialogues.

For the second experiment, the volunteers were asked to access each of the two web services twice. During one of the trials, the adaptive module was activated and both services were accessed. In the other trial, the system chose between directed or open messages and again both services were accessed. Each subject accessed the two versions of the system and the services in a random order. The questionnaire had to be filled in at the end of each mode of interaction, instead of right after each dialogue. Ninety-one people accessed the

test. Sixty-three volunteers finished the first part of the experiment (two dialogues and a questionnaire), but only 33 completed both parts (four dialogues and two questionnaires): 26 people accessed the directed messages plus the adaptive version, and 7 accessed the open messages and the adaptive version <sup>2</sup>.

All tests were performed in Spanish, as it was assumed that this was the first language of all users. Table 7.3 summarizes the set of dialogues and questionnaires collected during both experiments. The first column corresponds to the first experiment, in which questionnaires had to be completed right after each dialogue. The second and third column correspond to the second experiment, in which questionnaires were required at the end of a mode test. Column 2D1Q corresponds to the questionnaires collected from users who completed at least two dialogues and one questionnaire. Column 4D2Q corresponds to users who completed the whole test. To compute the figures used to compare the different systems and strategies described in the next section, we only used data obtained from the users who completed all of the stages of the experiments.

Initiative Mode	First Experiment	Second Experiment	
		2D1Q	4D2Q
Directed	53	45	26
Open	51	17	7
Adaptive	-	67	33
Total	104	129	66

Table 7.3: Number of questionnaires collected during both experiments

### Metrics and trial results

The dialogues were analyzed according to the four performance features proposed in the PARADISE evaluation framework, as described in [Walker et al., 1997]:

1. System usability: the user's satisfaction.
2. Task success: the user gives the system all the information needed to perform the web service task.
3. Dialogue quality: the system errors and the appropriateness of the system messages.
4. Dialogue efficiency: the duration and number of turns.

---

<sup>2</sup>Therefore, most of the people who accessed the open messages version of the system did not complete the experiment.

### 7.2.2 A comparison of the linguistic resources: first vs. second experiment

We measured the system usability from the questionnaires that the users were asked to complete, as discussed in the previous section. Table 7.4 shows the set of questions and scores for the first and second experiments, in which the rating scale ranged from 1 (strongly disagree) to 5 (strongly agree). The second version of the system, which included improved language resources and the adaptive module, increased the average score of all the questions. The Mann-Whitney test showed that the increase was significant for 6 out of 10 questions. The average increase was also significant, which reinforces the score increase for the first (overall impression) and eighth (usefulness) questions.

The increase in scores for Question 3 was probably due to improvements in grammars and lexicons. The increase in scores for Questions 5 and 6 was most likely the result of the adaptive module. Finally, the increase in scores for Question 7 could be attributed to the revision of the phrase patterns.

The score increase for Questions 2 and 4 was not statistically significant. These two questions are related to the information displayed to the user and specifically, to the presentation of the results and the generation of the system's response. In fact, there were no major differences in these aspects in the two versions of the system. Finally, the last questions were related to the user's willingness to use the system.

Average score for the 10 questions (1..5)	1st	2nd	Mann-Whitney
1. My overall impression is good	3.11	3.58	0.0080
2. The system provided the desired information	3.35	3.59	0.2488
3. You felt understood by the system	2.90	3.32	0.0346
4. You knew at each turn what the system expected from you	3.39	3.74	0.0535
5. You perceived the dialogue as pleasant	3.25	3.68	0.0308
6. You found the system's help appropriate	3.37	3.83	0.0097
7. You were able to understand the system without any problems	3.62	4.09	0.0060
8. You found the system was useful for accessing web information	3.19	3.64	0.0296
9. In the future, you would use the system again	3.20	3.42	0.3171
10. You would prefer to access this system by phone	2.48	2.91	0.0584
Average for questions 1 to 9	3.26	3.65	0.0184

Table 7.4: Questions scores related to individual interactions: 1st vs. 2nd experiments.

We used the independent sample t-test to compare the two experiments on the grounds of

task success and dialogue quality and efficiency. Table 7.5 summarizes the results obtained. As in the usability test, the second experiment resulted in better performance. In general, there were fewer errors and inappropriate system messages. Intuitively, there were fewer interactions (as there were fewer errors) and the user spent less time on each turn (as the messages were clearer).

To summarize, the improvement due to the linguistic resources and the adaptive module was statistically significant on the grounds of dialogue quality (system errors) and usability (questionnaire test).

Performance features	First experiment	Second experiment	p-value
Task success (%)	82%	83.3%	0.783
Avg. system errors/dialog	2.37	2.19	0.003
Avg. ambiguous messages/dialog	0.297%	0.154%	0.085
Avg. user turns/dialog	6.42	6.10	0.469
Avg. time (seconds)/dialog	124	102	0.222

Table 7.5: Evaluation of efficiency: 1st vs. 2nd experiments

### 7.2.3 A comparison of the initiative strategy

We also used the independent sample t-test to compare the dialogues using open messages, directed messages and the adaptive mode (in the second experiment). Dialogues with directed system messages had significantly fewer system errors than the ones using open messages, mainly because more directed messages resulted in shorter and predictable user interventions, which were easier for the analyzer to process.

We obtained similar results when we compared directed messages and the adaptive version using the paired sample t-test. Adaptive dialogues had significantly more errors, as shown in table 7.6. Significantly more time was required to complete the tasks in the adaptive dialogues, even though the number of turns was very similar. This is because the adaptive version used mainly open messages that resulted in longer user sentences, which take longer to be written.

We also used the paired t-test to compare open messages and the adaptive version. The results are shown in table 7.7. The number of dialogues using open messages was low (7 users, 14 dialogues), which makes the comparison difficult, as shown by the 100% task success.

Dialogues involving the adaptive version had a significantly lower number of ambiguous messages. Although this measure could contribute to the higher system usability of the adaptive version, it does not contribute to a higher rate of task success. The fact that less time is taken when the system uses open messages only than when the adaptive mode is active could also have an impact on the poor number of examples. For the sake of comparison, we spanned the set of dialogues with the excluded<sup>3</sup> ones and used the independent t-test to compare both modes of interaction again. This resulted in 158 seconds and 6.50 turns for the open messages and 133 seconds and 5.65 turns in adaptive mode. The p-values were 0.538 and 0.338 respectively.

Paired-t test directed vs. adaptive	Directed	Adaptive	p-value
Task success (%)	82%	80%	0.766
Avg. system errors/dialog	0.749	1.273	0.038
Avg. ambiguous messages/dialog	0.1434	0.1320	0.872
Avg. user turns/dialog	6.192	6.198	0.990
Avg. time (seconds)/dialog	73.5	103.8	0.037

Table 7.6: Directed messages vs. adaptive messages

Paired-t test open vs. adaptive	Open	Adaptive	p-value
Task success (%)	100%	91.67%	0.339
Avg. system errors/dialog	1.827	1.284	0.381
Avg. ambiguous messages/dialog	0.2675	0.0275	0.021
Avg. user turns/dialog	6.217	5.486	0.409
Avg. time (seconds)/dialog	81.0	94.7	0.602

Table 7.7: Open messages vs. adaptive messages

## 7.2.4 A comparison between transactional and informative services

We compared the performance of the DS when the two web services<sup>4</sup> were accessed (table 7.8). The only significant difference was related to the number of errors. There were more errors related to interpreting the user answer for the object collection service, in which the specific data (objects and addresses) were harder to recognize. In contrast, there were significantly

<sup>3</sup>When we include the users who only finished 2 dialogues and 1 questionnaire, the set was comprised of 18 users and 39 dialogues.

<sup>4</sup>The informative service CA is the cultural agenda and the transactional service LOC is the large objects collection

more errors due to dialogue management in the informative service than in the transactional service. Indeed, this was even more obvious when the dialogues were peer-reviewed, and it motivated the study and design of the task manager and the presentation of the database results.

Paired-t test LOC vs. CA	LOC	CA	p-value
Avg. parser errors/dialog	1.09	0.58	0.018
Avg. DM errors/dialog	0.10	0.33	0.014
Avg. ambiguous messages/dialog	0.09	0.19	0.124

Table 7.8: Transactional vs. Informative services

Finally, we also analyzed several aspects of tasks and services using the data collected in the evaluation. First, figure 7.3 shows that the dialogue duration and number in the transactional and informative services, were similar on average (the t-test for the number of turns resulted in a p-value of 0.95). The number of tasks executed<sup>5</sup> within each dialogue was also similar (the p-value was 0.93). However, the number of task executions (service access) was significantly different (p-value=0.0). Specifically, it was higher in the informative service CA (3.67) than in the transactional service LOC (1.63). Indeed, in the LOC service, the database was only accessed more than one when the user was looking for green points. Table 7.9b shows the task success rate (TSR). The TSR was slightly higher in the LOC service, but the difference was not statistically significant (p-value=0.68). Table 7.9b also shows that all the errors that occurred during the dialogues were more or less equally shared out between both services. However, the types of errors differed. Errors in the NLU module were more frequent in the LOC service, probably because it is difficult to recognize the street and the object names. In contrast, the number of errors that occurred in the dialogue manager and in accessing the database was higher in the CA service. Indeed, these tasks were more complex in the case of the CA.

Figures 7.4a and 7.4b show the turn in which the LOC and CA services accessed the databases, respectively. The histograms show that in the CA service, most of the accesses occurred on the second and third turns. These early accesses can be explained by the ability of the informative service to query the service with a minimum set of data. In contrast, most

<sup>5</sup>A new task starts every time that the user returns to the main menu.

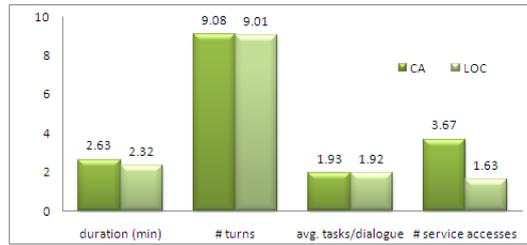


Figure 7.3: The dialogue duration in minutes and the number of turns, tasks and database accesses per dialogue.

CA	LOC, collection
Requested data	Task identifier
Query constraints	Objects
Constraint values	Collection address
	Date
LOC, cancellation	LOC, information
Task identifier	Task identifier
Collection identifier	Collection address
Confirmation	

(a) Input parameters in both services

	CA	LOC
TSR	75.25%	77.98%
Error	49.11%	50.89%
NLU	37.56%	62.44%
DM	77.78%	22.22%
DB access	92.31%	7.59%

(b) Perc. service errors

Table 7.9: CA vs. LOC figures

of the service accesses in the LOC service occurred on the fourth and fifth turn. In this case, the system cannot access the database until it has gathered all the data.

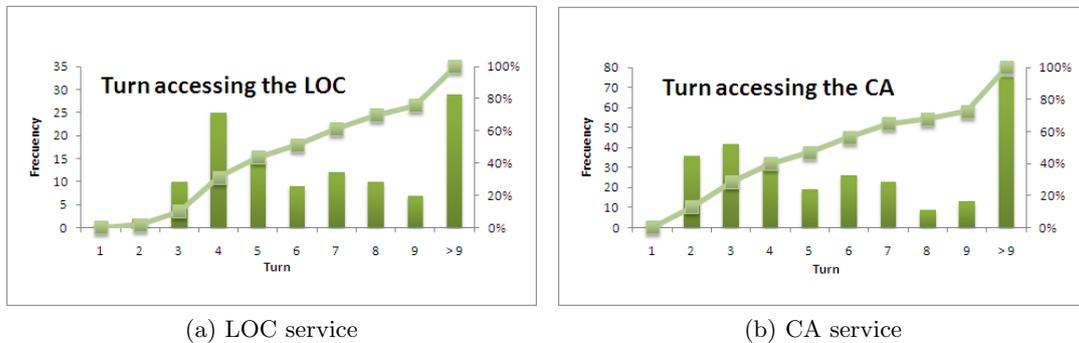


Figure 7.4: Turn on which the system accesses the database

## 7.2.5 Additional discussion

The experiments showed that improving the linguistic resources and integrating the adaptive module resulted in significantly fewer errors and higher user satisfaction. However, it did not improve the task success. In fact, task success rates were already high when the automatically generated system messages were used. The experiments also show that users prefer directed

system messages probably because they result in less errors. Indeed, the performance of the parser decreased when the user was allowed to openly express his/her intentions. Although we could have used more powerful natural language tools, we wanted to keep the portability to voice based systems. The grammars we used are compatible with those in Voice-XML platforms, even though we cannot simulate the errors produced by the ASR.

An evaluation of the prototype developed for the HOPS project<sup>6</sup>, which included an earlier version of the language and dialogue modules described in this thesis and a voice interface provided by LOQUENDO, compared figures for the system when either the text or the voice input were used. Figure 7.5 and 7.6 show some remarkable results taken from data collected in the city of Barcelona for voice and text input, respectively. More users considered that the system did not follow the scenario in voice mode, even though both modes of interaction used the same system. Regarding the error recovery strategy, more than 50% of phone users said that the system could not give a relevant answer. In contrast, less than 25% of the chat users thought the same. Finally, the third figure is related to the generation of system answers. The information given by the system did not make sense to almost 50% of the phone users. In contrast, the information made some sense to over 75% of the chat users.

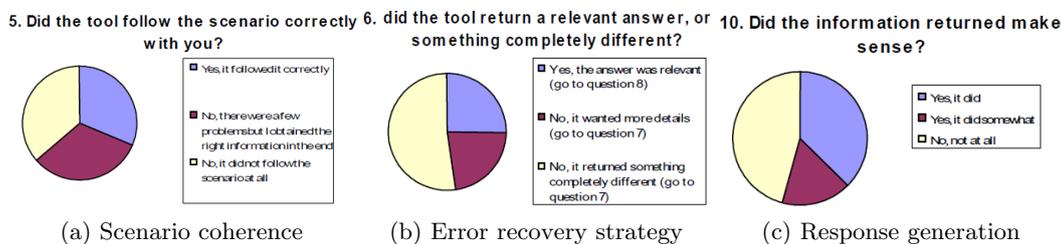


Figure 7.5: Results collected from users of the voice channel

The experience with previous prototypes of the system suggests that the use of the phone to communicate with an automatic system produces a certain level of stress. This may be because users consider that the system cannot empathize with them. This is supported by figures obtained from an earlier evaluation within the HOPS project. Figure 7.7 shows how users from Barcelona who used the voice mode of interaction answered the question “*Did you feel comfortable and relaxed in speaking to the HOPS tool?*”. As can be seen, more than 50% of the users did not feel relaxed. The use of the adaptive strategy could also result in

<sup>6</sup>Each municipality taking part in the HOPS project evaluated of the dialogue system. Among all the cities, 240 participants were recruited, 960 dialogues were produced, and 480 questionnaires were collected.

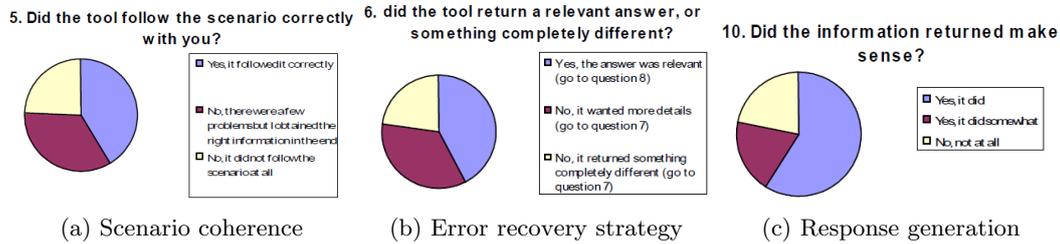


Figure 7.6: Results collected from users of the text channel

significantly better perception of the system in voice mode, as demonstrated in the case of text input, particularly if we take into account the fact that the voice input produces more comprehension errors.

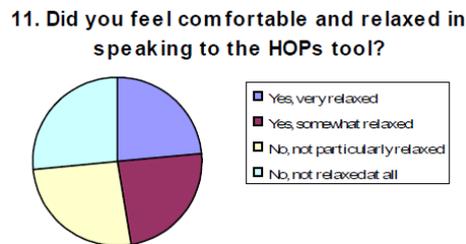


Figure 7.7: Did you feel comfortable and relaxed in speaking to the HOPS tool?

More experiments can still be carried out. For instance, we could use a general-purpose parser instead of the dedicated one that we designed. The use of general parsers avoids the need to adapt the resources. However, it also decreases the comprehension performance. More user features could be studied, as a detailed user model would enable the system to better handle a wider range of user features and we could simulate users to study the performance of the system strategies (as in [González et al., 2010]). Another interesting experiment would be to compare users' perceptions when they use the voice channel, which usually introduces more comprehension errors, in the latest version of the system. Further work could also include the study of multimodal systems and how to adapt the dialogue system to combine several modes of interaction simultaneously.

# Conclusions

---

We have developed a modular and flexible dialogue system that can access different types of web services in several languages. Each component of our system is independent. Application- and language-specific data are completely separated from the general processes involved in communication. To facilitate the adaptation of the system to different domains, explicit models that represent the knowledge involved in the communication are used: the task models for the service, the user model and the dialogue model. General conceptual and linguistic bases are also used. We studied an appropriate representation of the linguistic resources to facilitate adaptation to new services and languages. The particular contribution of this thesis is the integration of innovative technologies and models that have proven successful in other systems, within a dialogue system that guides the user to access web services.

The dialogue manager component uses an explicit dialogue model that is application and language independent. This model follows the Information State Update approach, which has proven very useful for dealing with different aspects of the dialogue. It is also useful for the various types of applications that give rise to the different types of interactions, such as form-filling or information-seeking dialogues. Our particular contribution to this approach has been to adapt it to dialogues that take place when the user accesses two different types of applications: transactional and informative.

Informative services need to guide the user through the available information and transactional ones need to guide the user to give values for specific parameters.

Transactional services incorporate a specific type of move, which is the submission. Therefore, they require explicit confirmation of specific input parameters, regardless of the confidence

level or the specific initiative strategy. Once data have been submitted, users only expect confirmation that the transaction has been successful. At this stage, they do not require additional information from the system.

Dialogues that are devised to access informative services are more complex. The volume of data that can be accessed in informative services may be high, so specific strategies are required. At any time, the user can change the information that has been requested or just add new information. In addition, the system can ask the user to change the constraints and the results can be presented in several ways. Hence, the dialogue manager handles continuous updates of the user goal. However, this does not necessarily mean that the user rejects what the system understood or the data provided in the previous user turn.

Our dialogue system can also adapt the degree of initiative to the communication needs. We developed and integrated a specific module in which a user model dynamically computes a set of metrics related to the communication. These metrics are used to adjust the degree of initiative in the system responses, which endows the system with some friendliness.

Dialogue management in DIGUI follows the ISU approach, which supports user-initiative interaction. However, the basic functions of the dialogue system become more complex when mixed- and user-initiative dialogues are addressed, which detracts from the flexibility and portability of the system. The DIGUI dialogue manager uses communication plans to represent the steps that must be followed for the web services to adapt to the system. These plans are used to detect the user intentions by interpreting his/her interventions, and to decide the next system actions. The communication plans are explicit in our system and are derived from the task specification. Our main contribution in this line has been to study the most appropriate representation of the dialogue and application knowledge, to facilitate the development of communication plans. We have defined general plans and general schemas (task models) to assist the generation of communication plans for two web services. The general schemas were incorporated into the task manager. Although we have defined the task models for two service types, they could be extended to tackle more specific features. Furthermore, other new task models could easily be defined for other types of applications using the same approach.

The task manager uses the task schemas to help DIGUI to determine the system response,

which mainly depends on the results that are retrieved from the services. The information in the task schemas is used to determine when and how to acquire more reliable constraints from the user, when and how to manipulate the user's constraint to obtain more suitable results, and the most appropriate presentation of results. This is an interesting line of research, which has been approached from the perspective of the language generator in the past. Recent studies show the importance of the results presentation strategy, e.g. [Paksima et al., 2009]. The problem can also be addressed as part of, or in collaboration with, the task management, e.g. the Content Optimizer in [Varges et al., 2009b]. The task schemas used by the task manager are obtained from the task specification, which includes mainly input and output parameters we focused on in this thesis. Nevertheless, in other approaches, the system retrieves the results directly from databases. In this case, the task and/or the dialogue structure can be built from the database schema. For instance, [D'Haro et al., 2009] describes an end-to-end platform and method for semi-automatically generating human-machine dialogue applications. Although the general approach of the dialogue system differs from that of DIGUI, the study of the method used in this platform and the dialogue representation could result in better models and increased understanding of the task and dialogue structure.

We also developed a natural language generation module for our dialogue system, and we adapted an existing natural language understanding module. Most dialogue systems use simple paraphrasing and hand-crafted sentences to generate the system messages. Our system uses a pattern-based mechanism, in which the patterns for expressing the system's potential dialogue moves are generated semi-automatically when a new service is incorporated. A syntactic-semantic taxonomy is used to facilitate this task. In addition, although the current version of the system is only available through the text channel, this strategy has proved useful when applied to a VoiceXML-based environment. Hence, in this research line, we have designed linguistic resources that are compatible with those described in the VoiceXML specifications. This means that the prototype can easily be adapted to voice-mode (and could deal with its specific paradigms, as defined in the VoiceXML standard). Moreover, our system addresses multilingualism. The semantically structured linguistic knowledge decouples language-dependent from language-independent processes. This approach makes it easy to adapt the system to other languages.

## 8.1 Further studies

The modular organization of the system means that new modules can be incorporated or existing ones replaced. Further studies could be carried out on different aspects of the system.

In the current version, the task manager uses simple algorithms that match the parameters and conditions of the task with the content of the IS. However, more complex algorithms could be incorporated so that the system intelligently solves query constraints, including constraint relaxation, as shown in [Rieser and Lemon, 2009].

For this purpose, the system would benefit from the combination of task models and ontologies to improve portability to other applications and domains, as shown in [Milward et al., 2005]. In the resources, we have defined a possible design to represent conceptual knowledge, which is currently based on a simple taxonomy. In fact, we benefited from the use of ontologies to represent the application knowledge in preliminary prototypes of the system. We implemented a preliminary version of this design using OWL ontologies, as the tools and resources for inferring new knowledge from the data and for reasoning about it are available for these formalisms. Due to the complexity of these technologies, we considered them beyond the scope of this study. Nevertheless, we studied the advantages of using ontologies in the two web services considered in this thesis. In particular, we found that ontologies that properly represent time and space enrich interaction with the system. For instance, specific constraints that can be relaxed follow the relation “belongs-to”, e.g. in the consideration of a district when the user gives an address. We also considered how to relax the dates and found a strong dependence on the domain context. For example, when the user gives a day of the week, the system can relax the constraint and consider the entire week or the weekend.

Another example of how ontologies can improve the performance of the system can be found in the taxonomy of objects and their materials. The system could infer whether an object is pollutant by searching the materials in its parts. We studied how to obtain linguistic resources from domain ontologies. As a result, we generated part of the lexicon for the LOC from an existing furniture taxonomy that can be accessed on the web, [Gatius and González, 2005d].

The dialogue system could also be improved by incorporating machine learning techniques.

Current research on dialogue systems addresses the use of machine learning techniques in several aspects of the dialogue system, such as dialogue management, task structure or the generation of the system responses.

In the task management field, the system would benefit from the incorporation of techniques for automatic extraction of the task structures. This feature would make the system flexible enough to access other online services. For instance, we could consider using powerful toolkits, such as the W4F described in [Sahuguet and Azavant, 2001]. This toolkit is a rapid designer and generator of wrappers that could be used to access different web services.

To integrate new applications into DIGUI, specific resources that feed the modules need to be developed: grammars, lexicons, patterns and tasks. Although several parts of these resources can be developed semi-automatically, the bindings among resources (concepts and attributes) still need to be set manually. Some platforms facilitate this task by integrating into a single environment the design and development of all the resources. This is the case of ACTIVE, a single unified platform that creates end-to-end intelligent assistants. Guzzoni et al. [2006] describe the platform architecture and functionalities and show how to use the ACTIVE platform to model and deploy a natural language assistant that can retrieve information about restaurants and movies. This platform uses ontologies to arrange processing elements, i.e. the concepts, attributes and relationships of every item in the application domain. The ontology serves as the platform's execution environment.

Hence, the research line related to ontologies will be important in future studies. Ontologies can assist the procedures of the system modules and help the developers to design and create a unified development environment for the dialogue system.

POMDPs have been used in most recent publications on the next generation of dialogue systems, specifically those on dialogue managers (e.g. [Williams, 2005]), and even in combination with the ISU approach. Hence, the study and development of user simulators has emerged from the need to train such dialogue systems. In this line, we can study the corpus of dialogues that we collected, to establish a better user model. The aim is to better understand the range of user behaviours from which statistical systems can learn better dialogue strategies. Additional features can also be incorporated into the user model that would benefit other modules, such as the user's verbosity, cooperativeness or linguistic slang (e.g. [Griol et al.,

2009]).

Furthermore, improvements in the system's behaviour must be reflected in the system's responses. Hence, the natural language generator has to handle sophisticated system actions. Otherwise, the user will not be aware of the real capabilities of the system. Consequently, improvements in the dialogue manager imply the enrichment of the syntactic-semantic taxonomy used to generate the system answers. Although our design of the linguistic resources has proven sufficient for our purposes, there is room for further improvement in this field. The natural language understanding and generation components would benefit from sharing the language resources. This could help to address reference resolution or to use the same slang as the user. Such features would make the system even more natural and friendly to the user. The taxonomy and lexicons used to generate the linguistic structures for the natural language generator can also be used to generate grammars in the natural language understanding module. However, the resources that we generated would probably need to be extended.

To summarize, we have developed a completely operational dialogue system prototype, which can be accessed in text mode through the web and offers friendly communication for different types of services and users.

# Examples

---

The following examples show the main actions and data involved when the system processes the user's turns. DM indicates an action performed by the dialogue manager. TM indicates an action performed by the task manager. PLAN indicates the actions stored in the plan in the information state. TASK indicates the content of the task structures. AGENDA indicates the data that the system's turns produced. NLU indicates the result of the natural language understanding module in the last user's turn.

In the first example A.1 a user wants to fix a collection of some furniture, including a fridge. After registering the collection, the system suggests to look for a green point where to leave the fridge.

In the second example A.2 the user is looking for some information about a museum. In this example the user needs to recover from a system error.

In the last example A.3 the user navigates the information about a set of events. This example illustrates how the system changes the constraints of the search at each turn.

### A.1 Large Objects Collection Example

*S1: Welcome to the Automatic Platform. How may I help you?*

DM: loadPlan(init)

AGENDA greet(), ask(serviceid)

---

***U1: I want to arrange a collection.***

NLU: ask(serviceid, loc), ask(loctype, collection)  
 DM: loadPlan ( collection )  
 TM: loadTask ( collection )  
 PLAN: [ asking(address), executing(collectiondate), answering(collectiondate),  
 asking(agree-date), asking(userType), asking(volume), asking(object),  
 executing(pollutant), answering(pollutant), asking(confirm),  
 executing(transactionId), answering(transactionId) ]  
 TASK: Collection [parameters: ]

---

***S2: OK. Tell me the address.***

AGENDA: feedback(loadplan, collection), ask(address)

---

***U2: Jordi Girona street, 3.***

NLU: answer(address, [street(jordi\_girona), number(3)])  
 DM: execute ( collectiondate )  
 TM: loadTask ( collectiondate )  
 execute ( collectiondate )  
 PLAN: [ executing(collectiondate), answering(collectiondate), asking(agree-date),  
 asking(userType), asking(volume), asking(object), executing(pollutant),  
 answering(pollutant), asking(confirm), executing(transactionId),  
 answering(transactionId) ]  
 TASK: CollectionDate [parameters: [address, [street(jordi\_girona), number(3)]],  
 result: [date, next(wednesday)]],  
 Collection [parameters: ]

---

***S3: The collection will be the next Wednesday at 8pm at Jordi Girona street, 3.  
Is it ok?***

AGENDA: answer(date, next(wednesday)), ask(agree-date)

---

**U3: yes**

NLU: acknowledge(positive)  
 TM: update ( collection )  
 PLAN: [ asking(userType), asking(volume), asking(object), executing(pollutant),  
 answering(pollutant), asking(confirm), executing(transactionId),  
 answering(transactionId) ]  
 TASK: Collection [parameters: [address, [street(jordi girona), number(3)]],  
 [date, next(wednesday)] ]

---

**S4: OK. Tell me if you are a private party or you call from a company.**

AGENDA: feedback(understood, positive), ask(userType)

---

**U4: what?**

NLU: answer(help, userType)  
 DM: loadPlan ( help(userType) )

---

**S5: Collections are free of charge for citizens. Companies must pay a fee for the collection. Are you a private party or a company?**

AGENDA: answer(help, userType), ask(userType)

---

**U5: I'm a private**

NLU: answer(usertype, private)  
 TM: update ( collection )  
 PLAN: [ asking(object), executing(pollutant), answering(pollutant),  
 asking(confirm), executing(transactionId), answering(transactionId) ]  
 TASK: Collection [parameters: [address, [street(jordi\_girona), number(3)]],  
 [date, next(wednesday)], [userType, private] ]

---

**S6: OK. Which objects are you throwing away?**

AGENDA: feedback(understood, positive), ask(object)

---

***U6: A fridge, a table and four chairs***

NLU: answer(object, [(1,fridge), (1,table), (4, chair)])

DM: execute ( pollutant )

TM: loadTask ( pollutant )  
execute ( pollutant )

PLAN: [ answering(pollutant), asking(confirm), executing(transactionId),  
answering(transactionId) ]

TASK: Collection [parameters: [address, [street(jordi\_girona), number(3)]],  
[date, next(wednesday)], [userType, private], [object, [(1,table), (4, chair)]] ]

---

***S7: I am sorry, electrical appliances cannot be thrown away in the street. You may drop it off at a Green Point. The Collection data is: One table and four chairs, next Wednesday at 8pm at Jordi Girona street, 3. Do you agree?***

AGENDA: answer(pollutant, fridge), ask(confirm)

---

***U7: yes***

NLU: acknowledge(positive)

DM: execute ( collection )

TM: execute ( collection )

PLAN: [ executing(transactionId), answering(transactionId) ]

TASK: Collection [parameters: [address, [street(jordi\_girona), number(3)]],  
[date, next(wednesday)], [userType, private], [object, [(1,table), (4, chair)]]],  
result: [transactionId, 12345] ]

---

***S8: OK. I am registering the collection. The transaction has been done correctly. Your collection number is 12345. Do you want to find the nearest Green Point to Jordi Girona street, 3?***

AGENDA: answer(submit, done), answer(transactionId,12345),  
ask(loadtask, information)

---

**U8: yes thanks**

NLU:       acknowledge(positive)  
 DM:       loadPlan ( information )  
 TM:       loadTask ( information )  
           execute ( information )  
 PLAN:     [ *asking(address)*, *executing(gpinfo)*, *answering(gpinfo)* ]  
 TASK:     ObtainGP [parameters: [address, [street(jordi\_girona), number(3)]],  
           result: [gp, GP] ]

---

**S9: OK. I am searching. The nearest Green Point to Jordi Girona street, 3 that collects electrical appliances is Les Corts-Pedralbes, at Av. Esplugues / Gran Capità.**

AGENDA: feedback(loadplan,information), answer(gpinfo, GP)

---

**U9: And where is the Green Point of Vallcarca?**

NLU:       ask(loctype, information), answer(district, vallcarca)  
 DM:       reraise ( information )  
 TM:       update ( information )  
           execute ( information )  
 PLAN:     [ *executing(gpinfo)*, *answering(gpinfo)* ]  
 TASK:     ObtainGP [parameters: [address, [district(vallcarca)]],  
           result: [gp, GP] ]

---

**S10: OK. I am searching. The nearest GP is Collserola, at C. Collserola, 2. Sortida 6 Ronda de Dalt.**

AGENDA: feedback(loadplan,information), answer(gpinfo, GP)

## A.2 Cultural Agenda Example 1

### *U1: I want information about the Museum of History*

NLU: ask(serviceid,ca), answer(location,bcn\_history\_museum)  
 DM: loadPlan ( ca )  
       execute ( findlist\_ca )  
       loadPlan ( eventinfo )  
 TM: loadTask ( obtainLocation )  
       execute ( obtainLocation )  
 PLAN: [ answering(name), answering(address), answering(description), answering(phone),  
       answering(facilities), answering(web) ]  
 TASK: ObtainLocation [ parameters: [name, bcn\_history\_museum],  
       result: [location, LOCATION] ]

---

*S2: OK. I am searching. The Barcelona City's Museum of History is at Plaça del Rei. The telephone number is 93 256 21 22. The web address is <http://www.museuhistoria.bcn.es/>*

AGENDA: feedback(loadplan,ca), answer(name,bcn\_history\_museum), answer(phone,"93 256 21 22"),  
       answer(address,"Plaça del Rei"), answer(web,"http://www.museuhistoria.bcn.es/")

---

### *U2: No, I want to know where is the Museum of History of Catalonia.*

NLU: feedback(rejection), answer(location, cat\_history\_museum), ask(address)  
 DM: reject  
       reraise ( findlist\_ca )  
       execute ( findlist\_ca )  
       loadPlan ( eventinfo )  
 TM: update ( obtainLocation )  
       execute ( obtainLocation )  
 PLAN: [ answering(address) ]  
 TASK: ObtainLocation [ parameters: [name, cat\_history\_museum],  
       result: [location, LOCATION] ]

---

**S3: OK. I am searching. The Museum of History of Catalonia is at Plaça de Pau Vila, 3.**

AGENDA: feedback(reraise,eventinfo), answer(location(address), “Plaça de Pau Vila, 3”)

---

**U3: Which facilities there are?**

NLU: ask(facilities)

DM: reraise ( eventinfo )

PLAN: [ answering(name), answering(address), answering(description),  
answering(phone), answering(facilities), answering(web) ]

TASK: ObtainLocation [ parameters: [name, cat\_history\_museum],  
result: [location, LOCATION] ]

---

**S4: Facilities information is not available. The Museum of History of Catalonia is at Plaça de Pau Vila, 3. The telephone number is 93 225 47 00. The web address is <http://www.en.mhcat.net/>**

AGENDA: answer(facilities,[]), answer(name,cat\_history\_museum),  
answer(address,“Plaça de Pau Vila, 3”), answer(phone,“93 225 47 00”),  
answer(web,“<http://www.en.mhcat.net/>”)

### A.3 Cultural Agenda Example 2

*U1: Where are there events for children?*

NLU: ask(serviceid,ca), ask(address), answer(genre,childlike)  
 DM: loadPlan ( ca )  
       execute ( findlist\_ca )  
       loadPlan ( listofevents )  
 TM: loadTask ( findlist\_ca )  
       execute ( findlist\_ca )  
 PLAN: [ pagination(findlist\_ca), reset(ca) ]  
 TASK: FindList\_CA [ queryConstraints: [(genre,childlike)],  
       requestedData: [location] ]  
       [ result: [list, LOCATION(8)] ]

---

*S2: OK. I am searching. There are 8 places. These are the results:*

*1: Plaça del Rei,*

*2: Plaça Catalunya,*

*3: Plaça Lessesps,*

*...*

*Select one of the results to search events in the place, or change the search criteria.*

AGENDA: feedback(loadplan,ca), answer(pagination(location), (list,LOCATION(8)))

---

*U2: 1.*

NLU: answer(selectitem,1)  
 DM: reraise ( ca )  
       execute ( findlist\_ca )  
       loadPlan ( listofevents )  
 TM: update ( findlist\_ca )  
       execute ( findlist\_ca )  
 PLAN: [ pagination(findlist\_ca), reset(ca) ]  
 TASK: FindList\_CA [ queryConstraints: [(genre,childlike), (location,pl\_rei)],  
       requestedData: [title] ]  
       [ result: [list, EVENT(2)] ]

---

**S3: OK. I am searching. There are 2 events. Results follow:**

**1: Titelles,**

**2: Contacontes.**

**Select one of the results or change the search criteria.**

AGENDA: feedback(reraise,ca), answer(pagination(title), (list,EVENT(2)))

---

**U3: What time does contacontes begin?**

NLU: ask(time), answer(selectitem,2)

DM: reraise ( ca )

loadPlan ( eventinfo )

TM: loadTask ( obtainEvent )

update ( obtainEvent )

PLAN: [ answering(time) ]

TASK: ObtainEvent [ parameters: [name, contacontes],

result: [event, EVENT] ]

---

**S4: I am searching. The event Contacontes begins at 12a.m.**

AGENDA: feedback(reraise,ca), answer(event(time), "12:00:00")

---

**U4: Give me all the information**

NLU: answer(requestedData,all)

DM: reraise ( ca )

loadPlan ( eventinfo )

PLAN: [ answering(price), answering(description), answering(sellpoint),

answering(participant), answering(genre), answering(channel),

answering(location), answering(time), answering(date), answering(title) ]

TASK: ObtainEvent [ parameters: [name, contacontes],

result: [event, EVENT] ]

---

**S5: OK. The event Contacontes is at Plaça del Rei. It begins at 12a.m. It is free, childlike and suitable for people of all ages ...**

AGENDA: answer(price,free), answer(genre,childlike), answer(channel,theater),

answer(location(address),"Plaça del Rei"), answer(time,"12:00:00"),

answer(title,contacontes) ]

---

***U5: and are there theater plays for children today?***

NLU: ask(title), answer(channel,theater), answer(genre, childlike),  
 answer(date,today)

DM: reraise ( ca )  
 execute ( findlist\_ca )  
 loadPlan ( listofevents )

TM: loadTask ( findlist\_ca )  
 execute ( findlist\_ca )  
 update ( findlist\_ca )  
 execute ( findlist\_ca )

PLAN: [ relax((channel,theater), (genre, childlike), (date,today)),  
 pagination(findlist\_ca), reset(ca) ]

TASK: FindList\_CA [ queryConstraints: [(genre,childlike), (date,today)],  
 requestedData: [title] ]  
 [ result: [list(2), EVENT] ]

***S6: OK. I am searching. There are no theater plays for children today. There are 2 films for children. Results from 1 to 2 follows:***

***1: Ice Age at Central Cinema,***

***2: Night at the museum at Central Cinema.***

***Select one of the results, or change the search criteria.***

AGENDA: feedback(reraise,ca),  
 answer(listofevents(0), ((channel,theater), (genre, childlike), (date,today))),  
 answer(pagination(title), (list(2),EVENT))

***U6: give me all the theater plays***

NLU: ask(title), answer(channel,theater)

DM: reraise ( ca )  
 execute ( findlist\_ca )  
 loadPlan ( summaryevents )  
 reraise ( ca )

TM: update ( findlist\_ca )  
 execute ( findlist\_ca )

PLAN: [ answering(summary,findlist\_ca), reset(ca),  
 asking(queryConstraint), asking(requestedData),  
 macroaction(constraintValues), executing(findlist\_ca)

TASK: FindList\_CA [ queryConstraints: [(channel,theater)],  
 requestedData: [title] ]  
 [ result: [list(MAX), ()] ]

---

***S7: OK. I am searching. There are too many theater plays. Please, give me other information about what are you looking for.***

AGENDA: feedback(reraise,ca), answer(summary, list(MAX), ()),  
 ask(queryConstraint)

---

***U7: theater plays today***

NLU: answer(channel,theater), answer(date,today)

DM: execute ( findlist\_ca )  
 loadPlan ( summaryevents )  
 reraise ( ca )

TM: update ( findlist\_ca )  
 execute ( findlist\_ca )

PLAN: [ answering(summary,findlist\_ca), reset(ca),  
 asking(location(district)), executing(findlist\_ca) ]

TASK: FindList\_CA [ queryConstraints: [(channel,theater), (date,today)],  
 requestedData: [title] ]  
 [ result: [list(5), summary(location(district))] ]

---

*S8: OK. I am searching. There are too many theater plays today. Which district are you interested in?*

*There are 5 events at Poblenou District.*

*There are 3 events at Ciutat Vella District.*

*There are 2 events at Sarrià District.*

*There are 3 events at Gràcia District.*

*There are 2 events at Les Corts District.*

*Select a district or change the search criteria.*

AGENDA: feedback(reraise,ca),

answer(listofevents(MAX), (channel,theater), (date,today)),

answer(summary(location(district))), ask(location(district))

# Analysis of the structure of the tasks for accessing web services

---

This appendix analyzes the structure of the dialogue that takes place when the user accesses the web services. Concretely, we analyze the different types of information involved in the dialogue: application and linguistic knowledge, general dialogue mechanisms and information about the user. In order to achieve an appropriate representation of this information, we have studied the interaction with two different web services: LOC (transactional) and CA (information).

## B.1 The transactional service LOC

The LOC service allows the citizens to arrange a collection for their old furniture. It also allows the companies to get rid of large amounts of rubbish. This service has three top-level tasks: to arrange a date for a collection, to cancel a collection and to obtain information about Green Points (GP). The latter is a single task, whereas to request and to cancel a collection are composed tasks, i.e. they need the execution of several smaller tasks.

To arrange a date for a collection consists of two tasks. First, the system asks the user the collection address and obtains a date for the collection. Second, the system asks the user for additional input data and executes the transaction (i.e. the collection is registered).

To cancel a collection requires also the execution of two tasks. First, the system retrieves the collection from the application. In the second task, the system orders to remove the collection.

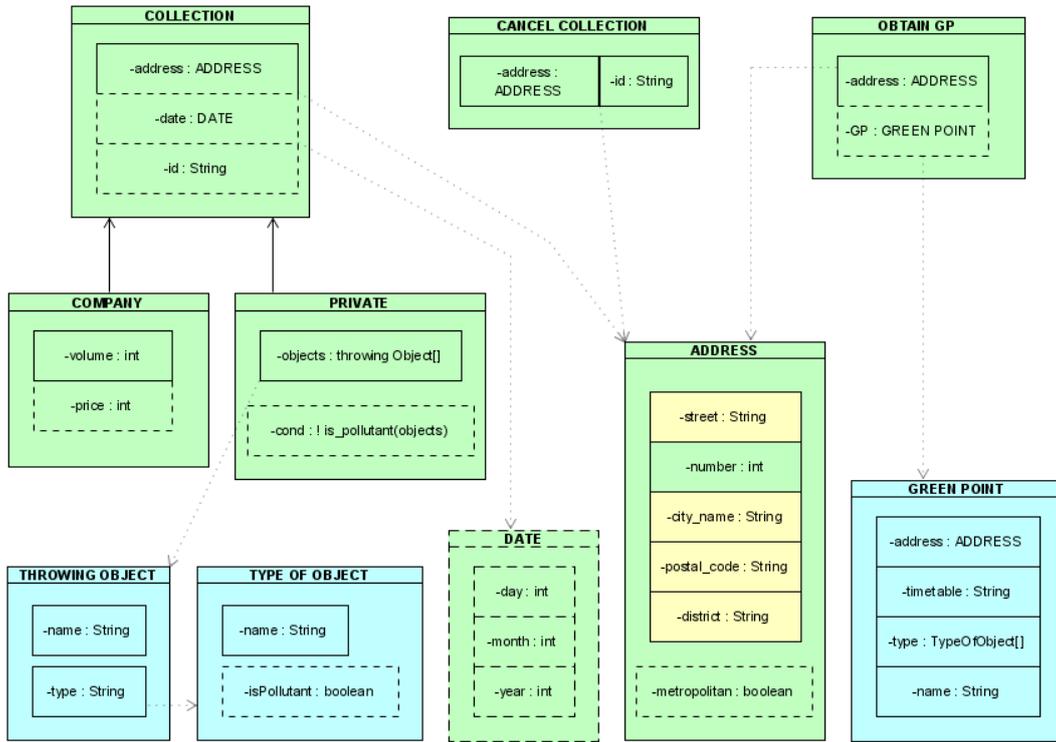


Figure B.1: The LOC model

Figure B.1 shows the tasks and information involved in the LOC service. We represent the three main tasks as objects. The attributes of each object represent the input and output parameters of the tasks. The dotted boxes represent the data that is calculated by the system. Additional objects are used to represent the complex structures (e.g. addresses and GPs). The dotted arrows indicate which object represents the type of the attribute.

### The tasks

The object `COLLECTION` represents the data involved in the task that arranges collections. In this task, all the users must give the collection address and the type of collection: private and company. In case the collection is demanded by a company, the user must also give the volume of the objects. In case the collection is demanded by a private party, the user must also give the list of objects and the system checks that none of the objects is pollutant. The service calculates the following data: the date of the collection, the price of the collection (in case of companies' collections) and the collection identifier.

The object `CANCEL_COLLECTION` represents the data involved in the task that cancels

collections. The user must give either the address or the collection identifier in order to identify which collection to cancel.

The object OBTAIN\_GP represents the task that gives information about the GPs. The user gives an address and the system looks for a GP near to the given address.

### **The data**

The object DATE, composed of day, month and year, represents the date for the collection.

The object ADDRESS is composed of the following attributes: street, number, city, postal code and district. In the LOC service the user must provide a street and a number. The city, the postal code and the districts are optional values that can be asked to the user or calculated by the system, depending on the specific application. Moreover, a specific implementation of the application can include a process to check the validity of an address, i.e. it corresponds to a street in the city. In our system, this information is stored in the conceptual knowledge. We considered information about locations a resource widely used in public administration services.

THROWINGOBJECT collects information about an object being thrown. It has two attributes: type and name. The name of the object is given by the user, whereas the system determines the type of the object (in the semantic interpretation process). In case the system does not recognize the name of the object, it asks the user to provide a more general description of the object. Since in our specific application the pollutant objects cannot be collected, the conceptual knowledge includes a classification of pollutant objects.

The GREEN POINT has the following information: the name of the place, the address, a description of the timetable and a list containing the type of objects that the recycling place can collect. All the data is part of the knowledge of the application. The web service allows the user to search a GREEN POINT near to a given address. How to determine whether a place is near to another place can vary from one application to another. In our application, the collection takes place inside a city, and so we consider that one place is near to another if they belong to the same district.

**The dialogue example**Dialogue B.1

---

U1: I want to set up a collection.

S1: OK. Tell me the address.

U2: Jordi Girona street, 3.

S2: The collection will be the next Wednesday at 8pm at Girona street, 3. Is it ok?

U3: yes

S3: OK. Tell me if you are a private party or a company.

U4: what?

S4: Collections are free of charge for citizens. Companies must pay a fee for the collection.

Are you a private party or a company?

U5: I'm a private

S5: OK. Which objects are you throwing away?

U6: a fridge, a table and four chairs

S6: I am sorry, electrical appliances cannot be thrown away in the street. You may drop it off at a Green Point. The collection data is: One table and four chairs, next Wednesday at 8pm at Jordi Girona street, 3. Do you agree?

U7: yes

S7: OK. I am registering the collection. The submission has been done correctly. Your collection number is 12345.

Do you want to find the nearest Green Point to Jordi Girona street, 3?

U8: yes

S8: OK. I am searching. The nearest Green Point collecting electrical appliances is Les Corts-Pedralbes, at Av. Esplugues / Gran Capità.

U9: And where is the green point of Vallcarca?

S9: OK. I am searching. The nearest Green Point is Collserola, at C. Collserola, 2. Sortida 6 Ronda de Dalt.

---

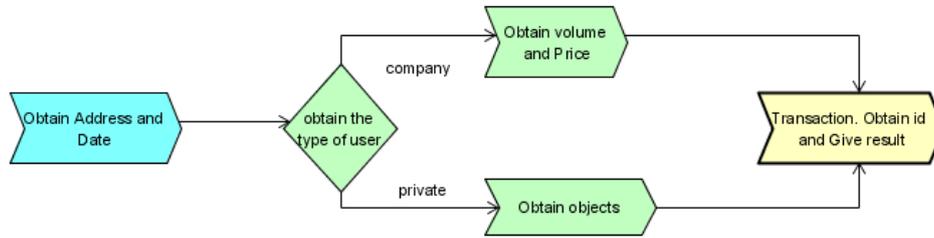


Figure B.2: The Collection flowchart

### B.1.1 The task “Collection”

Figure B.2 shows the flow for the task Collection. This chart shows only dependencies among the data involved in the object COLLECTION, and none related to errors and conditions. This task can be splinted into two parts. In the first part, the user gives the data for the collection and the system gives the date of the collection. In the second part, the system obtains the data that depends on the type of the user. In case of the companies collection, the system calculates the price of the collection. In case of private parties collection, the system confirms that none of the objects to be thrown up are pollutant. Finally, the system submits the collection data and assigns an identifier to it.

In order to enable the integration of this task into the DS, we must consider further situations such as i) the possible errors during the execution of the task, ii) the conditions defined in the application and iii) to avoid unexpected states during the communication. In consequence, additional stages need to be incorporated in the flow of the task shown in figure B.2.

Figure B.3 shows the flowchart for the same task Collection. In contrast with previous chart, this one has been adapted to the interaction within a DS. So, this chart also shows the flows related to errors and conditions. We differentiate four parts. In the first one the system obtains the date and requires the user agreement. We consider that it is more efficient to inform the user about the date and to ask for the agreement before going on with the next task (and asking for more data). In case of companies collection, the system follows up with another task to obtain the price of the collection. The user has to agree the price of the collection in order to proceed with the submission. In case of a private collection, the system follows up with the task that checks that none of the objects in the collection is pollutant.

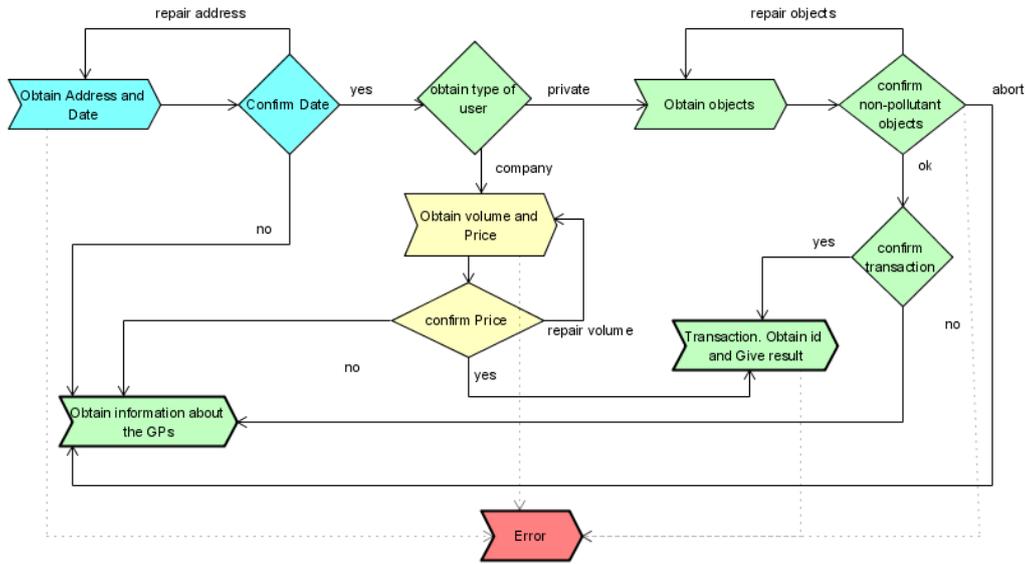


Figure B.3: The Collection flowchart adapted to the DS

Finally, in the last task, the collection is registered.

Note that whenever has sense, the system redirects the flow to the task that gives information about the GPs. This is a dialogue strategy of the application flow.

Besides, the global error recovery policies inform the user about any failure and decide the next system's actions. As can be seen in figure B.3, there are several conditional paths in the flowchart.

### Options in task Collection:

#### Confirm the date:

This conditional state appears in the adaptation to the DS. It represents an agreement of the data given and calculated by the system. At the same time, it represents an implicit confirmation that the system correctly understood the values given by the user. These two actions are always performed together every time that the next system's action is not a transaction. In consequence, three different paths must be considered. Dialogues B.1, B.2 and B.3 show different examples of user's answers. In turns (S1-S2) in dialogue B.1 the system continues with the normal execution of the task. In dialogue B.2 the system overpasses all the following actions and goes to an alternative state. The question in the system's answer indicates that the system is trying to guide the user to the task that gives information about

GPs. In dialogue B.3 the user corrects the system's error. In this case, the DM mends the content of the IS. Then, the task is executed again with the new values and obtains a new date for the collection.

Dialogue B.2

---

S1: The collection will be the next Wednesday at Girona street, 3. Is it ok?

U1: No. Can't be on Monday?

S2: I am sorry. You can call again the next week or go to a Green Point. Do you want to find the nearest Green Point to Girona street, 3?

---

Dialogue B.3

---

S1: The collection will be the next Wednesday at Girona street, 3. Is it ok?

U1: No, I said Jordi Girona street, 3

S2: The collection will be the next Tuesday at Jordi Girona street, 3. Is it ok?

---

Select the type of the collection:

This conditional state already appears in the application description. It consists of selecting a value for the collection type. Turns (S3-S5) in dialogue B.1 and B.4 show the different answers of the system. Moreover, in the first example the user asks for help (U4). The system answers accordingly (S4).

Dialogue B.4

---

S1: Are you a private party or a company?

U1: I'm a company

S2: OK. Please, tell me the volume of the objects in cubic metres?

---

Confirm the objects are non-pollutant:

This is a conditional state that does not appear in the application flowchart (figure B.2), but it is defined in the task specification. Pollutant objects cannot be collected because they cannot be thrown away in the street. Instead, they must be dropped off in any of the recyclable places in the city. The system may confirm that none of the objects is pollutant before going on with the execution of the task. If the confirmation is negative, the system

notifies the violation of the condition, and the user has the chance to accept the condition or to abort the execution of the task. In turn (S6) in dialogue B.1 the system recognizes the fridge as an electrical appliance, which cannot be thrown in the street, and the user is informed.

Agree the price:

This state requires an agreement with the data calculated by the system. In the same turn, the user implicitly confirms that the system understood correctly the values given previously. There are three different paths after the system obtains the price. First, the user accepts the price of the collection and it is then registered. Second, the user rejects the price of the collection. And third, the user rejects the price and repairs part of the information. In this case, the DM updates the values and calculates the price again. Dialogue B.5 shows an example.

Dialogue B.5

---

S1: The price of the collection of 20 cubic metres is 100 euros. Do you agree?

U2c: No, I said 12 cubic metres.

S2c: OK. The price of the collection of 12 cubic metres is 60 euros. Do you agree?

---

Confirm the submission:

This conditional state is part of the submission and it appears in the adaptation to the DS. Before submitting definitively (i.e. updating the databases), the system confirms that all the data that will be used have been understood correctly. This conditional state always precedes the execution of a SubmitForm task, which is only executed if the user agrees. Turns (S6-S7) in dialogue B.1 shows an example.

**Errors in task Collection:**

Obtaining the date:

There are two reasons the system cannot give the user the date of the collection: because the address does not exist and because there has been an error in the execution. In such cases the user cannot agree with the date of the collection and the execution flow cannot continue. Dialogue B.6 shows an example. The system informs about the error and gives an alternative to continue the dialogue.

---

Dialogue B.6

---

S1: I am sorry, it has been an error while obtaining a date for the collection. You can call again the next week or go to a Green Point. Do you want to find the nearest Green Point to Girona street, 3?

---

Submitting the registration:

If the execution of the transaction fails, then the system must clearly inform the user and abort the execution of the task. Consequently, it is not possible to give the user a submission identifier.

Obtaining the price:

If the system cannot calculate the price of the collection, it is still possible to continue the dialogue if the user agrees with the unknown price. In the example, the system informs about the error and gives the user the chance to continue.

---

Dialogue B.7

---

S1: I am sorry, it has been an error while obtaining a price for the collection. The collection data is ... Do you want to continue?

---

Consulting the objects:

It is possible that the application does not know the type nor the name of the objects. In this case, the system cannot consult the type of the objects. This is the case in dialogue B.8, where the system gives the user a recommendation about the objects.

---

Dialogue B.8

---

S1: OK. Remember that pollutant objects cannot be thrown in the street. The collection data is ... Do you agree?

---

Submitting the registration:

If the submission fails, the collection is not registered and the system cannot obtain the identifier. Hence, the flow cannot continue and the system must inform the user.

### B.1.2 The task “Obtain\_GP”

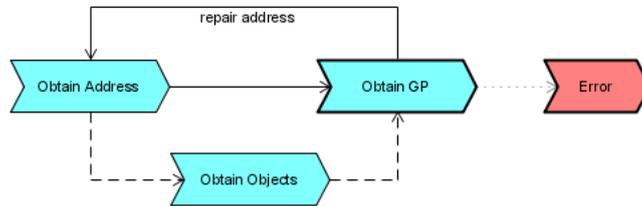


Figure B.4: The Obtain\_GP flowchart

This is a simple task where the system gives the nearest GP to a given address. In order to pick up the nearest GP, the system needs to know the district of the given address and the existing GPs. The DS stores the information about the GPs, the streets of the city and their districts in the conceptual knowledge of the application. Besides, some of the GPs specify which objects do they collect since not all of them are able to collect every type of objects. However, this information is usually not required and the system uses it only when the user gives a list of objects or other extra information. Figure B.4 shows the flowchart for the task. Dashed lines represent optional paths.

Turns (S7-S9) in dialogue B.1 are examples of interaction within this task. The system explicitly asks whether to use a previously given address. Then, the system looks for the GP, taking into account the extra parameter (fridge). In the next turn, the user gives a new address and the system executes the task Obtain\_GP again. In this new execution the system uses the new address and does not use the optional parameter.

#### Errors in task Obtain\_GP

##### Obtaining the GP:

The task to obtain GPs does not consider any conditional state. However, it considers possible errors during the execution of the task. Concretely, we consider that an error can occur while searching for the GP. For example, when the application does not recognize the address or when there is a failure accessing the service. In such cases, the system cannot continue with the normal flow. Instead, it informs the user about the error and gives an alternative, which is generated by other system mechanisms.

### B.1.3 The task “Cancel\_Collection”

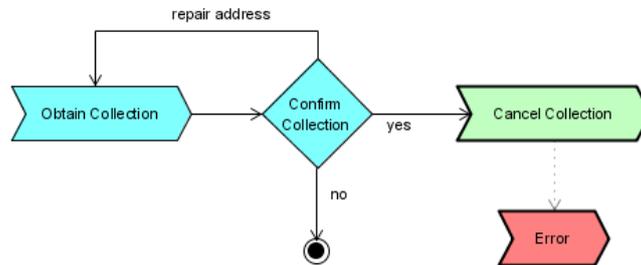


Figure B.5: The Cancel\_Collection flowchart

Figure B.5 shows the flow, already adapted to the DS, for cancelling a registered collection. To cancel a collection consists of two tasks. In the first one, the system identifies and retrieves the collection data. In the second part, the system orders the cancellation of the collection.

#### Options in task Cancel\_Collection

##### Confirm the collection:

As can be seen in figure B.5, there is a conditional state after the selection of the collection data. In this state, the system gives the user the information retrieved about the collection. The user must confirm that the data correspond to the collection that he/she wants to cancel. Although this type of confirmations are usually done implicitly, the next step in the flow is the execution of the SubmitForm, which requires explicit confirmation.

We consider three different paths in this conditional state. First, a misunderstanding during conversation makes the system to retrieve a wrong collection. In this case, the user repairs the misunderstanding and the DM updates the parameter’s values and executes the task again. Second, the user agrees with the retrieved collection. And third, the user rejects the cancellation of the collection. In this last case, the DM decides how to continue the dialogue since the application flow does not define it.

Dialogue B.9 shows an example. The user gives the address, which is misunderstood by the system, and the user repairs the data. Finally, the user confirms the cancellation data and the system submits it.

Dialogue B.9

---

S1: Give me the identifier of the collection or the picking up address.

U2: Jordi Girona 3

S2: OK. I am searching. I am sorry, there is no collection at C/ Girona, 3. Do you want to return to the main menu?

U3: no, I said C/Jordi Girona, 3.

S3: OK. I am searching. Do you want to cancel the collection on Tuesday at C/ Jordi Girona, 3?

U4: yes.

S4: OK. I am cancelling the collection. It has been successfully done! Do you want to return to the main menu?

---

### Errors in task Cancel\_Collection

#### Retrieving the collection:

The system cannot obtain the collection. This can occur because the system cannot recognize the address or the given identifier. Another reason is an internal failure of the service. The system cannot continue with the flow because the user cannot confirm that the retrieved collection is the correct one.

#### Submitting the cancellation:

The collection is not cancelled when the submission fails. The system cannot continue and must inform the user that the cancellation did not succeed.

## B.2 The informative service (CA)

In the information tasks the input and output parameters are more flexible than in other type of tasks. The CA service allows the citizens to consult the cultural events in the city and to get information related to the events, e.g. the places where they are played. The specific CA application we have implemented consists of three top-level tasks: to search for a list of items that satisfy specific constraints (e.g. date, place and/or time), to obtain information about a specific event and to obtain information about a specific location. In figure B.6 these three

tasks are represented as objects. The attributes of each object represent the parameters for accessing the service. The other objects in the figure represent other data related to the tasks and their parameters.

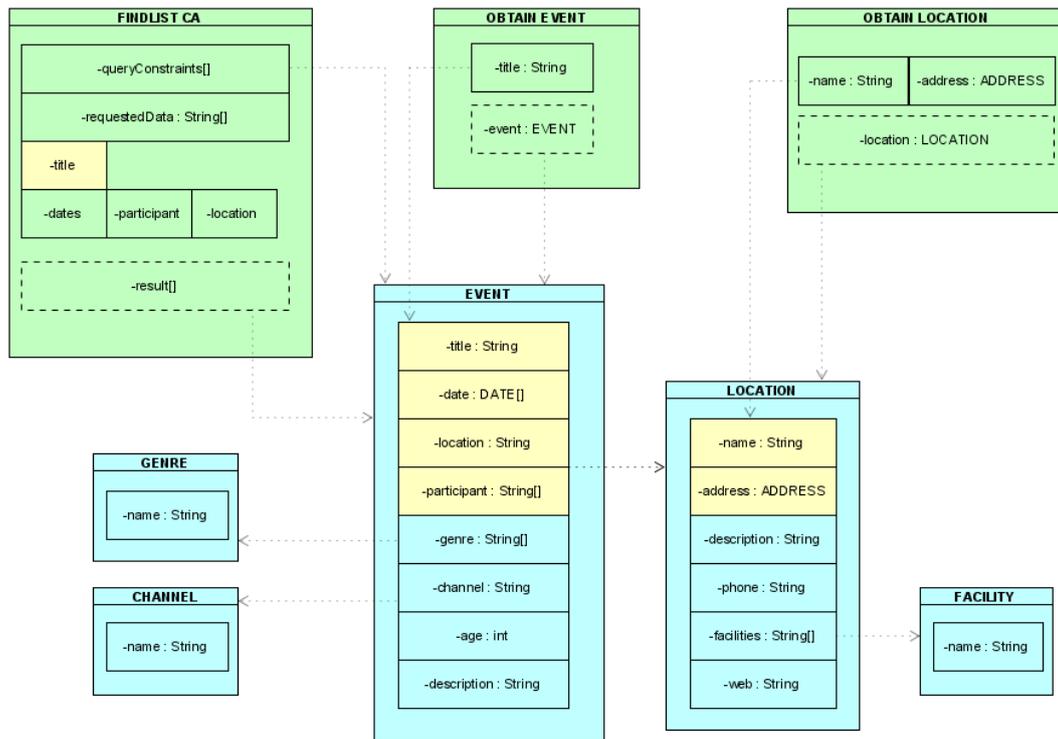


Figure B.6: The CA model

### The tasks

The task OBTAIN\_LOCATION allows to obtain all or part of the information available about a specific location. The task OBTAIN\_EVENT allows to obtain all or part of the information available about a specific event. In these two tasks the user provides the name of an item and the information that is looking for. For instance, in dialogue B.10 the user wants to know the facilities of a specific museum. The system searches the item and gives the information requested.

The object FINDLIST\_CA represents the information involved in the task that gives a list of items. The queryConstraints attribute represents the information that the user can give to constrain the search. The requestedData is the information that the user wants to obtain. In our particular implementation the allowed requestedData are: the title of an event (what),

the place where is played (where), the dates the event is available (when) and the people that takes part in the event (who).

### The data

The objects LOCATION, ADDRESS and FACILITY are part of the specific knowledge of the service. Having the information about facilities defined as the application knowledge allows the user use this data in the constraints. Similarly, the user can search for events being played in a specific district, which is an attribute of an address.

The object EVENT has the following information: title, date, location, participant, genre, channel, age and description. The first four correspond to the data that can be used in the requestedData. In consequence, the user can ask for a list of titles, a list of dates, a list of locations or a list of participants.

The GENRE and the CHANNEL belongs to the specific conceptual knowledge of the service. In consequence, the system can classify and filter the events using these attributes.

### The dialogue examples

#### Dialogue B.10

---

U1: I want information about the Museum of History

S1: OK. I am searching. The Barcelona City's Museum of History is at Plaça del Rei. The telephone number is 93 256 21 22. The web address is <http://www.museuhistoria.bcn.es/>

U2: No, I want to know where is the Museum of History of Catalonia.

S2: OK. I am searching. The Museum of History of Catalonia is at Plaça de Pau Vila, 3.

U3: Which facilities are there?

S3: Facilities information is not available. The Museum of History of Catalonia is at Plaça de Pau Vila, 3. The telephone number is 93 225 47 00. The web address is <http://www.en.mhcat.net/>

---

#### Dialogue B.11

---

U1: where are there events for children?

S1: OK. I am searching. There are 8 places. Results follow:

---

Dialogue B.11 - Continues on next page

Dialogue B.11 - Continued from previous page

---

1: Plaça del Rei,

2: Plaça Catalunya,

3: Plaça Lesseps,

...

Select one of the results to search events in the place, or change the search criteria.

U2: 1.

S2: OK. I am searching. There are 2 events. Results follow:

1: Titelles,

2: Contacontes.

Select one of the results or change the search criteria.

U3: what time does contacontes begin?

S3: OK. The event Contacontes begins begins at 12a.m.

U4: give me all the information

S4: OK. The event Contacontes is at Plaça del Rei. It begins at 12a.m. It is free, childlike and suitable for people of all ages ...

U5: and are there theater plays for children today?

S5: OK. I am searching. There are no theater plays for children today. There are 2 films for children. Results from 1 to 2 follows:

1: Ice Age at Central Cinema,

2: Night at the museum at Central Cinema.

Select one of the results, or change the search criteria.

U6: give me all the theater plays

S6: OK. I am searching. There are too many theater plays. Please, give me other information about what are you looking for.

U7: theater plays today

S7: OK. I am searching. There are too many theater plays today. Which district are you interested in?

There are 5 events at Poblenou District.

There are 3 events at Ciutat Vella District.

---

Dialogue B.11 - Continues on next page

Dialogue B.11 - Continued from previous page

There are 2 events at Sarrià District.

There are 3 events at Gràcia District.

There are 2 events at Les Corts District.

Select a district or change the search criteria.

### B.2.1 The task “Obtain\_Location”

The task Obtain\_Location gives information about a specific place. The DS stores the information about the locations in the conceptual knowledge of the specific service. The locations are identified by the name, but also by the address, since only one location can have a concrete address. So, The user can give either the address or the name as an input parameter (queryConstraints). Additionally, the user can indicate which specific information wants about the location. This is an specialization of the task. In such case, the NLU and the NLG modules have to support the specialization. The former has to understand the constraint, and the latter has to generate an answer answer with the concrete information instead of the all the data. Figure B.7 shows workflow of the task. The dotted lines represent the optional path where the user indicates which specific information is looking for. The user can, repeatedly, change or repair the values used to execute the task. This task does not require confirmation of the parameter’s values. However, if any error occur when accessing the service, the system has to inform the user.

Dialogue B.10 is an example of this task. In this dialogue, the user gives the name of a museum and the system gives the information related to the museum found.

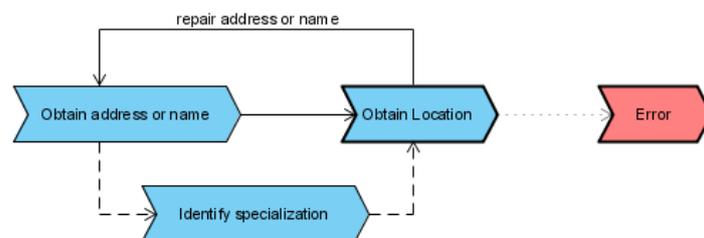


Figure B.7: Obtain\_Location flowchart

**Options in task Obtain\_Location**Specialize the requested data:

The regular work out of this type of task is to display all the information about the requested item. The specialization consists of displaying just the specific information, as described in the requestedData attribute. Turns (U2-S2) in dialogue B.10 are an example of the specialization of the requested data. The user indicates that the address is the only needed information (U2) and the system answers just this specific data (S1).

Repair the specialization:

In some cases can happen that the user asks for non-available data. In such case, other mechanism in the system decide which information to give to the user. If there is no mechanism able to repair the interaction, the system considers it a failure. Turns (U3-S3) in dialogue B.10 show how the system uses a general message to inform that the requested information is not available. Then, the system automatically updates the requestedData and it gives all the known information (which is the default behavior of the task). The use of this strategy allows the system to inform the user that the location was found, although not the specific data. This type of answer avoids the user to believe that the system did not find the location.

**Errors in task Obtain\_Location**Obtaining the location:

Dialogue B.12 is an example of the task Obtain\_Location where an error occur when accessing the service. In consequence, the system cannot find the location. The DM redirects the flow to a general state. First, the system informs the user about the error. Then, it continues with a generic question.

## Dialogue B.12

---

U1: Where is the Museum of History?

S1: OK. I am sorry. There has been an error while accessing the service. Do you want to return to the main menu?

---

### B.2.2 The task “Obtain\_Event”

The task Obtain\_Event gives information about a specific event. Figure B.8 shows the flow of this task. It is a simple flow where the user gives some data that identifies the event. Then, the system searches the event in the knowledge base and gives all the information retrieved. The flow contains a loop where the user can change or repair the data given and then the system searches for an event again.

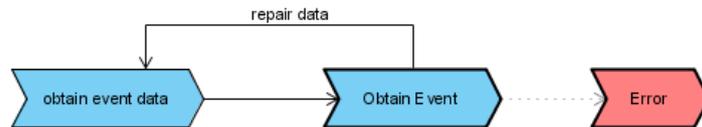


Figure B.8: Obtain\_Event flowchart

An object EVENT contains many data and, as in the case of the task Obtain\_Location, the user can specialize the requestedData by indicating the specific data that he/she would like to obtain. The title is the main attribute that identifies an event. However, it is possible that in some cases the title is not enough to identify a unique event. For example, a concert can be played in different cities. In this case, two different events exist in the knowledge base. In order to correctly clarify which of the two events the user is looking for, the system needs more information: either the date or the location. Our dialogue manager takes into account the clarification dialogue moves. Besides, our implementation considers to redirect the dialogue flow to the task FindList\_CA. Figure B.9 shows the flow of the task adapted to the dialogue system.

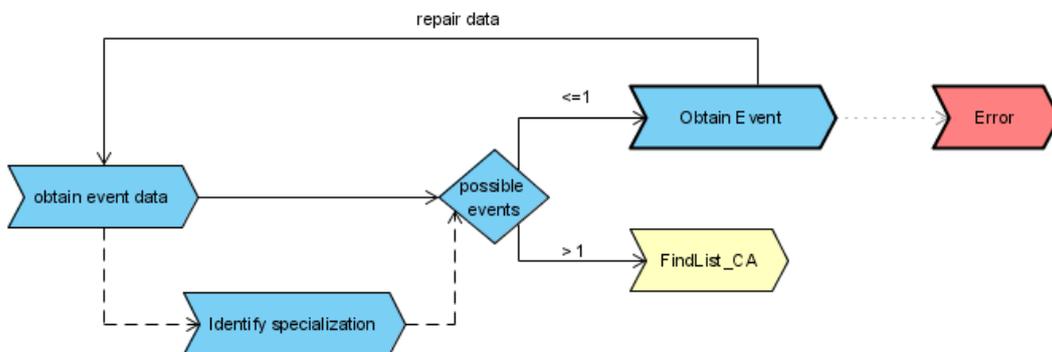


Figure B.9: Obtain\_Event flowchart adapted to the DS

**Options in task Obtain\_Event**Specialize the requested data:

An event has many information and the user may be interested only in some data. The user can indicate any of the fields in the object EVENT as the specific information that he/she is looking for. In consequence, the NLU has to be able to identify the specialization, and the NLG has to provide answers containing just the requested information. In turns (U3-S3) in dialogue B.11 the user gives the title of the event and indicates that he/she wants to know the starting time. The system searches for the event and gives just the requested data.

Flexible Constraints:

This task gives information about specific events. So, the queryConstraints must provide the exact attributes and values that identify a unique event. The combination of attributes that identify an event are: {title}, {title, date}, {title, location}, {location}, {location, date}. For instance, in dialogue B.13 the user asks for the venue of an event and gives the title and the date of the event. In dialogue B.14 the user gives the title and the location and asks for the date. These events corresponds to a series of films that the municipality shows outside during the summer. In this festival every film is shown in a different place. In other type of festivals there are events in many places, and at a single place the event played is different every day. In contrast, some places play the same event during several days. This is the case of art exhibitions and theater plays. This is the reason why a location can also identify an event. In these cases the user gives the name of the location and the system answer with the data about the event, including the title.

Note that, whatever is the request, the system gives the information used to perform the search. This is a mechanism to implicitly confirm that the system understood the values given by the user, but also makes the system to seem more helpful. After giving any result, the user can change the constraints and execute the task again.

Dialogue B.13

---

U1: Where is the outside cinema tomorrow?

S1: OK. I am searching. The Outside Cinema is at Plaça Reial, tomorrow.

---

Dialogue B.14

---

U1: When is the outside cinema at Praça Reial?

S1: OK. I am searching. The Outside Cinema is tomorrow, at Praça Reial.

---

Communicate the result(s):

The user is often unaware of how to use the task, the number of available events and how to identify them. Hence, when the user under-specify the queryConstraints, several events may match the input. However, this task can only give information about a single event. We can consider several solutions for this situation: i) the system informs the user which is the missing data (clarification), ii) the system redirects the flow to the task FindList\_CA (a list of items is delivered), iii) the system considers the situation an error.

The last solution seems poorly helpful to the user, whereas the other two seem more natural answer. Dialogue B.15 shows an example of solution i). The “Outside Cinema” is at different places every day. In order to get an answer, the user must clarify which date is interested in. Note that the task manager determines which is the missing data. In our system we have adopted solution ii). This solution requires the dialogue and task manager to be able to shift between tasks. In dialogue B.16 the system gives a list of places. This answer does not correspond exactly to the user request, but it seems more useful than the answer in dialogue B.15. Eventually, after displaying the list of results, the user may select one of the options and will be redirected again back to the task Obtain\_Event.

Dialogue B.15

---

U1: Where is the outside cinema?

S1: OK. I am searching. There are several possible results for Outside Cinema. Which date are you interested in?

---

Dialogue B.16

---

U1: Where is the outside cinema?

S1: OK. I am searching. There are 8 results for Outside Cinema. Results from 1 to 4 follow:

1. Praça Reial on July 5th.
- 

Dialogue B.16 - Continues on next page

Dialogue B.16 - Continued from previous page

---

2. ...

---

### Errors in task Obtain\_Event

#### Obtaining the event:

As shown in figure B.9, an error can occur when accessing the service. In this case the system informs the user about the error and asks the user what to do next.

#### Searching the event:

When the queryConstraints used for accessing the service does not match any event, the user can repair the set of constraints and execute the task again with the new values. In dialogue B.17 the user asks for the date of an event (U1) and the system answers that the event is unknown (S1). The content of this answer indicates that the system has not identified the title of the event, but that has identified the intentions of the user. More intelligent systems will also produce suggestions for similar titles. Once the user has repaired the misunderstanding (U2), the DM executes the task again with the new values (S2). In sentence (U2) the user gives an alternative name for the event.

Dialogue B.17

---

U1: Where is the outside cinema?

S1: OK. I am searching. The outside cinema event does not exists.

U2: cinema a la fresca

S2: OK. I am searching. Cinema a la Fresca is at Plaça Reial.

---

### B.2.3 The task “FindList\_CA”

Figure B.10 shows the flow for the task FindList\_CA. This task allows the user to obtain a list of items that match a list of constraints. The information related to an event is: the title, the location, the dates, the participants, the genre, the channel, the minimum age and the description. Other information up to the service could be included as any combination of attribute-value pairs is allowed. The user is able to modify the content of the constraints right after each task execution. The titles (what), the locations (where), the dates (when) and

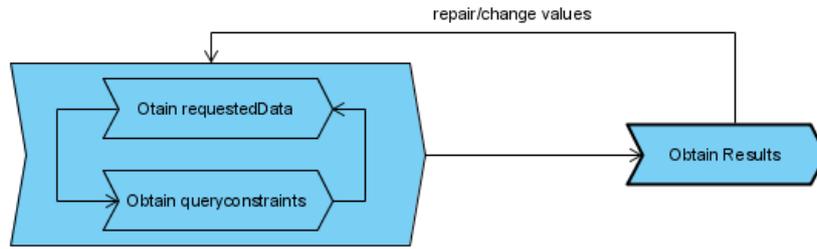


Figure B.10: FindList\_CA flowchart

the participants (who) are the attributes allowed for the requestedData. So, the system can give lists of titles, locations, dates or participants. Indeed, to give a list of descriptions or a list of genres does not make sense.

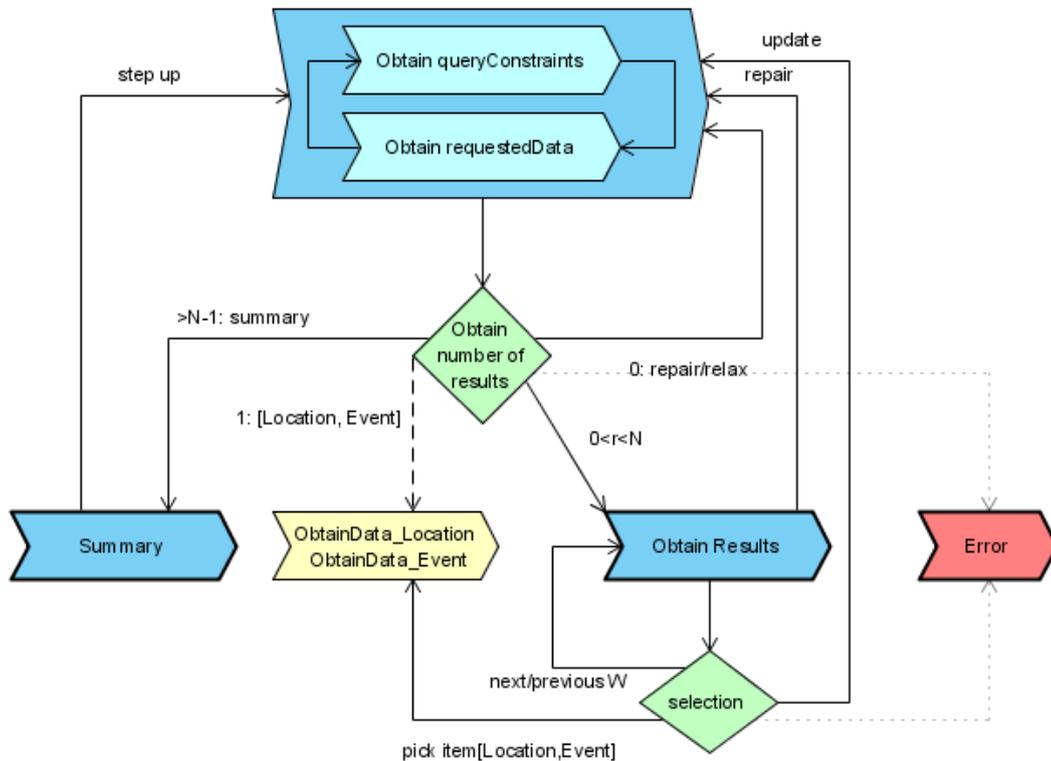


Figure B.11: FindList\_CA flowchart adapted to the DS

Figure B.11 shows the flow of the task adapted to the dialogue system. This flow illustrates the management of the results of the task execution. The system executes the task when the task has enough input parameters. The answer of the system depends on the number of results obtained from the execution. If there are no results, that means that the constraints were too strong. It is also possible that some values were incorrect or misunderstood. The system could

relax the constraints and execute the task again without requiring the intervention of the user. Similarly, if there are too many results, it is not suitable to enumerate them all (specially in voice-based systems). The system can give a summary of results or suggest the user how to step up the constraints efficiently. In case that the system obtains only one item, the system could redirect the flow to one of the other two tasks: Obtain\_Location or Obtain\_Event. This strategy allows the user to obtain detailed information using less interactions and making the communication more dynamic. The regular procedure consists of displaying a list of results that match the constraints.

#### TITLES:

In this task, the requestedData field not only indicates what the user is looking for, but it also indicates how to display the list. Turns (S5-S7) in dialogue B.11 show an example. The user asks for the title of the events given a specific location and date. This is the default configuration work out of this task.

#### PLACES:

In turn (U1) in dialogue B.11 the user asks for a list of locations. The system presents a list of locations (S1) and the user selects one of them (U2). At this point the interaction indicates that the user is looking for titles, instead of the specific information of the location. In fact, in this kind of services through the regular web browsers, the application displays the list of events for the location jointly with the information of the location.

Nevertheless, the system can use other strategies, depending on the degree of initiative of the system. If the system gives more initiative to the user, then it asks what information the user prefers. This solution requires more interactions. If the initiative is shared, the system needs a mechanism to analyze other aspects of the dialogue and determine the next actions. This is the case of our system design. When the user selects a place from the list, this new constraint is automatically added to the list of queryConstraints. Then, the system executes the task again and obtains another list of results. This behaviour does not require complex computation and neither extra interactions with the user. If this last execution returns only one result, then the system shifts to the task Obtain\_Event and presents to the user the information of the selected event.

### DATES:

Sometimes the user is looking for a list of dates, i.e. the user knows the event and wants to know when it is played. Theater plays and Circus performances are examples. When using this kind of service through the web, the user selects the date (and sometimes the timetable) for obtaining detailed information or buying tickets. Our service does not offer this kind of interactions and, in consequence, to select a date is not allowed. Additionally, some events are available for a long time. In such cases the list of dates is very long. Sophisticated NLG modules produce more elaborated messages for giving that information efficiently.

### PARTICIPANTS:

The user is also allowed to ask for the list of participants. The content of this list depends on the specific application. For example, the participant of a concert could be the name of the group or the name of the people that belongs to the group. Our system is unaware of this kind of information and the NLG uses more general messages suitable for all types of contents. This simplification makes the system also more adaptable to other services.

In fact, there are events that can include a participant that, at the same time, participates in other events, e.g. music festivals. In those cases the user can select a specific participant in order to browse another list of items.

It would be preferable that the system indicates the actions that can perform at each turn. For instance, in turn (U3) in dialogue B.11, when the user selects the item, the new constraint replaces the old one. Unlike this, in turn (U2), the new constraint is added to the old one. The message displayed after the enumeration of options (S1,S2) indicates the actions that the system performs.

Moreover, the results could display additional data for each item in the list, as in turn (S5). This organization of the data simulates regular web browsers. The difficulty is that different modules have to collaborate in order to generate these type of messages: the task manager has to include this data in the information state, the dialogue manager has to be able to interpret it and correctly pass it to the natural language generator, and this last has to be able to include the data in the message.

**Options in task FindList\_CA**List the results:

After listing the results, the user is able to perform several actions: to see more results (browse), to pick up an item from the list, to step up the constraints or to relax them. In order to browse the results the user uses well-known commands already used in other environments, such as “next” and “back”. To pick up an item the user indicates the number (U2), or any data given (U3). The interpreter must be able to interpret any of the possible answers.

Relax the constraints:

When the execution of the task does not return any result it is because i) there are no data in the knowledge base related to the data given, or ii) the user gave too many constraints and the system cannot find items that satisfy them all. The relaxation of the constraints only can occur in the second case. It is not possible to relax non existing or unknown data. To relax the constraints consists of deleting a specific attribute from the list of constraints or changing one of the values of the attributes. The former is more appropriate for constraints containing too many fields, whereas the latter is used when there are few. The resources related to the task (namely the task specification) indicates which field or value to relax in each case. For example, when the user gives the genre and the channel, it is more suitable to update the channel than the genre, as shown in turn (U5) in dialogue B.11. It suggest that the genre has more priority than the channel. Such type of information must be specified in the conceptual knowledge of the service.

Besides, after the relaxation, the system executes the task again. Then, it displays the results to the user jointly with the real used values. Consequently, the system implicitly informs the user about the constraint update.

Summarize the results:

In our dialogue system, summarize is the opposite to relaxation. When the user under-constraints the search, the system retrieves too many items. To list them all in a dialogue-based system is not suitable. In such situations, the system can use default values for specific attributes, but usually the user must give more information about what is looking for. The most simple solution is to let the user to give any other data. However, it can be useful

to guide the user or to suggest useful extra data. For example, in turn (U6) in dialogue B.11 the system asks the user for more data, however the data given is not enough for constraining the search. A more natural and friendly answer is to give a summary of the data (U7).

#### Use default values:

The informative services offer a wide flexibility for query constraints. Hence, the application must tackle any combination of attributes and values. Default values for the attributes are useful for both, experts and novel users. Experts know the behaviour of the system and so they omit those attributes they know the system will assign automatically. Novel users do not know all the needed attributes and the system expedites the conversation using the more common values for omitted attributes. Turn (U7) shows an example where an experienced user does not indicate the requested data.

### **Errors in task FindList\_CA**

#### Retrieving the Results:

It occurs when the system cannot obtain the results. The system informs the user about the error and redirects the conversation to another state in the flow.

#### Managing the Results:

Our system can shift between tasks, but this mechanism can fail. In such cases the system has to recover itself from the failure and suggest the user to return to the previous correct state.

# Resources

---

The modules that comprise the DIGUI system use a set of files. These files are the system's resources. They contain information about the tasks (namely structures and content) and the application's domain. The resources are designed for a specific module in particular, but they are available for all the modules within DIGUI.

The content of the resource files has different degrees of persistence. Some of the files were written at the time that we developed the system and they rarely need changes, e.g. the grammars for the parser. Other resources have to be written every time that a new application is integrated into the system, and they usually change when the application is updated, e.g. the plans. Finally, some resources continuously change, such as the content of the events database, particularly if the system uses online resources.

We chose XML as the formalism for writing DIGUI resources. XML documents can be written easily from other raw data and give structure and semantics to the data. Moreover, they can be easily read from different modules and can be shared on the Internet.

First, this appendix gives the XML schemas for the different resources that were developed for DIGUI. Then, it describes how we acquired the specific data for each application and how to produce the resources for a new application.

We have defined different XML schemas for each type of resource used in DIGUI, namely the task schemas and the domain data. These schemas have been designed taking into account the needs of the main module that uses them. The task schemas are mainly used by the task manager, while the domain data are linked to the natural language modules.

## C.1 The task schemas

We organized the web services as a top-level task that can be decomposed into smaller tasks. A top-level task is a main operation (functionality) that the service offers. The representation of a top-level task follows the structure below: the name of the operation, the conditions of the operation and the tasks that make up the operation. The order of the tasks is important because they are executed sequentially.

```
<operation name="...">
  <task></task>
  <task></task>
  [...]
  <condition></condition>
  [...]
</operation>
```

We defined three types of tasks in our services: SubmitForm, ObtainData and FindList. Each task is represented using the specific task schema. The tasks contain information about the input and output parameters, the constraints among parameters and the default values. The schema does not show how to solve conflicts between constraints. The specific procedure that processes the schema is responsible for resolving such issues.

### SubmitForm

The specification of a SubmitForm task consists of: the mandatory parameters, the optional parameters, the type of result that it returns and the conditions that must be fulfilled to execute the task. Each parameter has a name, type and cardinality. The default values include the parameter, the value to assign and the condition needed to use it.

```
<submitform name="...">
  <condition></condition>
  [...]
  <parameters>
    <parameter name="..." type="..." cardinality="..."></parameter>
    <parameter></parameter>
    [...]
  </parameters>
  <optionals>
    <parameter></parameter>
```

```

        <parameter></parameter>
        [...]
    </optionals>
    <defaults>
        <default>
            <literal object=".." slot=".."><value></value></literal>
            <condition></condition>
        </default>
        [...]
    </defaults>
    <result type="..."></result>
</submitform>

```

### ObtainData

The specification of an ObtainData has a set of parameter lists, rather than just one. Each list of parameters is a queryConstraints: a set of parameters that is needed to execute the task. Apart from the queryConstraints, an ObtainData task also consists of the conditions needed to execute the task and the type of result returned.

```

<ObtainData name="...">
    <condition></condition>
    [...]
    <queryConstraints>
        <parameter name="..." type="..." cardinality="..."></parameter>
        <parameter></parameter>
        [...]
    </queryConstraints>
    <queryConstraints>
        <parameter></parameter>
    </queryConstraints>
    <result type="..."></result>
</ObtainData>

```

### FindList

The most complex schema corresponds to the FindList task. The specification of a FindList task consists of: the set of queryConstraints, the allowed requestedData, the default values and their conditions, and some hints for generating summaries. The conditions are also required to know which mix of requestedData can be combined with the queryConstraints. For example, the user cannot ask for the title of an event if a title value has already been given in the queryConstraints parameters. Default values are particularly useful in this type

of tasks, in which a minimum set of constraints can be used to query the service. Each default value is evaluated separately in an order that depends on the specific implementation. The specification can also include the parameters that are allowed in the summary. This is a complex process, so it is an optional feature that not all of the applications implement.

```

<FindList name="...">
  <condition></condition>
  [...]
  <queryConstraints> [...] </queryConstraints>
  <queryConstraints> [...] </queryConstraints>
  <requestedData>
    <parameter name="..." type="..."></parameter>
    <parameter></parameter>
    [...]
  </requestedData>
  <defaults>
    <default>
      <literal><value></value></literal>
      <condition></condition>
    </default>
  </defaults>
  <summaries>
    <summary>
      <literal> <value></value></literal>
      <condition></condition>
    </summary>
    <summary> [...] </summary>
  </summaries>
</FindList>

```

### C.1.1 Schemas for the LOC service

#### Task Collection

The collection task involves obtaining the date and submitting the collection. Only one parameter is needed to obtain the collection date: the address. The collection task then returns a date. In the SubmitForm task Collection, there is a different list of parameters for private and company collections. This task returns an identifier of the transaction and requires the execution of two more tasks: confirmation that the objects are not pollutant in private collections and calculation of the price of the collection in company ones. The input parameter for the former task is the list of objects, and the list of pollutants is returned. The

input in the latter task is the volume of the objects, and the price of the collection is returned.

	<b>queryConstraints/Parameters</b>	<b>Result</b>
<b>Obtain_Date</b>	1. address	date
<b>Obtain_Pollutants</b>	1. object_name[]	object_name[]
<b>Obtain_Price</b>	1. volume	price
<b>Submit_Collection</b>	1. (private): address, date, object_name[] 2. (company): address, date, volume, price	id

Table C.1: The parameters in the task Collection

### Task Cancellation

Cancellation of a collection consists of retrieving the collection and ordering the cancellation. The system looks for the collection in the database using either the id or the address. If the collection is found, the task returns it. Ordering the cancellation is a transaction. The only parameter required in this task is the collection. Then, the result of the transaction is returned.

	<b>queryConstraints/Parameters</b>	<b>Result</b>
<b>Obtain_Collection</b>	1. identifier 2. address	collection
<b>Submit_CancelCollection</b>	1. collection	boolean

Table C.2: The parameters in the task Cancel Collection

### Task Obtain\_GP

This is a single task that involves looking for a green point near a given address. In our implementation, the only parameter required is the district. Nevertheless, this task also accepts an optional input parameter: a list of objects. Several green points do not accept specific pollutant objects. When the search includes the optional parameter, then the task only looks for green points that accept all objects. The task returns a GP object, which contains the data related to a green point.

	<b>queryConstraints/Parameters</b>	<b>Result</b>
<b>Submit_CancelCollection</b>	1. district 2. district, typeOfObject[]	GP

Table C.3: The parameters in the task Obtain Green Point

### C.1.2 Schemas for the CA service

The Cultural Agenda has three main tasks: to obtain a location, to obtain an event and to obtain a list of items. The CA service involves a top-level task to obtain the query constraints. Then, the system decides which task to execute depending on the constraints. The tasks to obtain a location and to obtain an event are very similar. Both of them return a single result. In contrast, several results have to be handled to obtain a list of items. Therefore, the task needs additional specification sections.

#### Tasks Obtain\_Location and Obtain\_Event

The task needs either a name or an address to find a location. The task returns a Location object, which contains the data related to a location.

In contrast, a combination of the title, the date and the location is required to find an Event. Table C.4 shows the exact combination of these data that are allowed. The task returns the event that matches the combination of query constraints. When more than one event matches the constraint, then the system shifts to the FindList\_CA task.

	<b>queryConstraints</b>	<b>Result</b>
<b>Obtain_Location</b>	1. name 2. address	location
<b>Obtain_Event</b>	1. title 2. title, date 3. title, location 4. location 5. location, date	event

Table C.4: The parameters in the tasks Obtain\_Location and Obtain\_Event

#### Task FindList\_CA

The FindList\_CA task described in table C.5 corresponds to our specific implementation of the Cultural Agenda. This task has a general condition that must be fulfilled: any attribute that is in the requestedData cannot also be in the queryConstraints. The specification of this task consists of: the queryConstraints, the requestedData, the summary, the relaxation and the default values.

The set of queryConstraints has more elements than in other tasks. Moreover, the parameters in the queryConstraints have a specific feature: they can accept more than one

value. By default, the system interprets the list of values as an or-operation of constraints. However, some parameters have other semantics. In particular, two values for the date attribute mean a period of time, the value for the age attribute means a minimum age, the time represents a period during the day (morning, afternoon, evening, night), and the value for the price attribute means the maximum cost.

In addition, this task has the `requestedData` attribute, which indicates the information that the user wants to obtain. The user is allowed to ask for one or more attributes. By default, if the user does not specify any specific attribute, the system returns all of them.

The summary section has a set of parameters. For each parameter there is a set of conditions that indicates when the parameter can be used to generate a summary of the results. If a single parameter does not allow a good summary to be created, then several parameters can be combined. Moreover, if more than one parameter can be used to generate the summary, our implementation of the service has a predefined parameter priority: first the location, then the date, the genre and the channel.

The relaxation section also contains a set of parameters and combinations of them. Each item in the relaxation section indicates how to relax the specific set of attributes. In some cases, there is more than one way to relax them. Table C.5 shows the set of predefined relaxation rules. Each row indicates the set of attributes involved in the constraint, and the new relaxed set. The number indicates a possible order of execution. For instance, relaxation number 3 indicates a constraint that contains the title and another attribute *x*. There are two possible new constraints: a) only the title and b) only the other attribute.

In addition, we defined a default way to relax the constraints: to delete the following attributes progressively in this order: channel, district, date, genre and title. We studied the advantages of using ontologies in the two web services considered in this thesis. For instance, specific constraints can be relaxed by following the relation “belongs-to”, e.g. the relaxed form of an address is the address district. We also considered how to relax the dates and found a strong dependence on the domain context. For example, when the user gives a day of the week, the system can relax the constraint and consider a period, e.g. from Monday to Friday or Saturday to Sunday. Another example of how ontologies can improve the performance of the system can be found in the taxonomy of objects and their materials. The system could infer whether an object is pollutant by searching the materials in its parts.

Finally, there is the default values section. Default values are used when a specific input parameter does not have a value and the condition is fulfilled.

<b>queryConstraints</b>	<b>RequestedData</b>	<b>Default Values</b>
1. location 2. district/city 3. facility 4. event 5. minimum age 6. participant 7. genre 8. date 9. time 10. language 11. price	1. event (what) 2. date (when) 3. location (where) 4. participant (who)	1. requestedData = location 1a. title $\in$ queryConstraints 1b. date $\in$ queryConstraints 2. requestedData = channel 2a. date $\in$ queryConstraints 3. requestedData = title 3a. location $\in$ queryConstraints 4. requestedData = location,title 4a. {date,channel} $\in$ queryConstraints
<b>Summary</b>		<b>Relaxation of the constraints</b>
1. channel 1a. channel $\notin$ queryConstraints 2. genre 2a. channel $\in$ queryConstraints 3. district 3a. location $\in$ requestedData 3b. {location, district} $\notin$ requestedData {location, district} $\notin$ queryConstraint		1. date: a) weekend, b) week 2. location: a) district 3. {title, x}: a) title, b) x 4. {genre, x}: a) genre, b) x 5. {date, location}: a) location, b) date 6. {date, channel}: a) channel, b) date 7. {date, district}: a) district, b) date 8. {channel, location}: a) channel, b) location 9. {channel, district}: a) district, b) channel 10. {location, district}: a) district

Table C.5: The parameters in the task FindList\_CA

## C.2 The domain knowledge

Most of the modules in DIGUI handle a semantic representation of the user's requests. Basically, the NLU generates a semantic representation of the user's turn and the other modules process it to generate new data. The way these data are processed depends mainly on the relations among the concepts and attributes of the domain representation, i.e. the task parameters. Each application has its own domain representation. In our implementation, the domain knowledge consists of objects, attributes and their relations.

For instance, as shown in the previous section, the system can relax the query constraints. Consider a query in which a user is looking for the timetable of a specific film at a specific cinema. If the film is not playing in the given cinema, then we can relax the constraint and

look for the timetables of the same film in cinemas in the same street or the same district. To perform this relaxation, the system needs to know the relation among the addresses, the districts and the attributes of the cinemas.

DIGUI has a specific module, named Conceptual Knowledge (CK), that stores the domain knowledge and implements the domain-related operations. The main operations of the CK are: to check whether a specific attribute is relevant to another attribute or a system action, to check whether an attribute achieves part of a goal, to combine the content of different attributes or actions, and to check whether two attributes are compatible.

In our specific implementation, an attribute  $\alpha$  is relevant to another attribute  $\beta$  if both have the same name or  $\alpha$  subsumes  $\beta$  and  $\alpha$  has a value.

An attribute  $\alpha$  solves a system action  $\sigma$  that contains the attribute  $\beta$  if  $\alpha$  is relevant to  $\beta$  and  $\sigma$  is a question.

An attribute  $\alpha$  is relevant to a system action  $\sigma$  that contains the attribute  $\beta$  if *alpha* solves  $\sigma(\beta)$ ;  $\alpha$  is a negative answer and  $\sigma$  is a wh-question; or  $\alpha$  is relevant to  $\beta$  and  $\sigma$  is an alt-question<sup>1</sup> Note that negative answers to wh-questions and alt-questions are relevant but do not resolve.

The result of combining attribute  $\alpha$  with  $\beta$  is attribute  $\beta$  with the values of attribute  $\alpha$ .

Two attributes are incompatible when they have the same name but different values and the attribute only accepts one value.

The resource data that are related to domain knowledge are stored in XML files. Every object has an XML specification that consists of its attributes or slots (name, cardinality and type) and conditions related to the object or the slots. A condition consists of the list of attributes (literal) and, optionally, the set of constraints (if\_then). It must be read as follows: if the constraints in the if\_then are satisfied, then the attributes in the list must also have the indicated values. If there is no "if\_then" section in the condition, then the list of attributes must have the indicated values. The value of an attribute can be a simple type (String, Integer, Double or Boolean) or another object (literal).

Apart from objects, the XML files also contain the relations among the objects. In our

---

<sup>1</sup>An alt-question is defined in [Larsson, 2002] as a clarification question in which the system forces the user to choose among different alternatives.

implementation, we specify that the relation of the type: `is_a`, `type_of` and/or `part_of`.

The general XML schema is as follows:

```

<object [entity] name="...">
  <slot name="..." type="{int , string , boolean (0,1) , double , objects}"
    cardinality="{1,*}">
    <condition></condition>
  </slot>
  <condition></condition>
</object>

<condition>
  <if_then><literal></literal><literal></literal></if_then>
  <literal object="object_name" slot="slot_name">
    <value (not)></value>
  </literal>
  <literal>[...]</literal>
</condition>

<condition>
  <literal object="object_name" slot="slot_name">
    <value (not)>*</value> </slot_name>
  </literal>
</condition>

<value>( String , int , double , boolean )</value>
<value>
  <literal object="object_name" slot="slot_name">
    <value></value>
  </literal>
</value>

<object name="...">
  <is_a>[...]</is_a>
  <is_a>[...]</is_a>
  [...]
  <part_of>[...]</part_of>
  <part_of>[...]</part_of>
  [...]
  <type_of>[...]</type_of>
  [...]
</object>

```

The following XML data example shows part of the knowledge related to the object address.

The first part gives the structure of the object address. Then, there are the relations between an address and other objects. There are two types of address, rural and metropolitan, and the address is part of a location. Finally, the example shows the use of the object address as an attribute of the object that represents a green point.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!--
 * ADDRESSES: street@number@city@postal@district@meropolitan@other
 * mandatory: street@number@city
 * example: C. collserola@2@BCN@@@1@sortida 6 ronda de dalt. -> well processed

    public String street = "";
    public int number = -1;
    public String city_name = "";
    public int postal_code = -1;
    public String district = "";
    public boolean metropolitan = false;
    public String other = "";

-->

<object entity name="domainaddress">
  <slot name="street" type="string" cardinality="1"></slot>
  <slot name="number" type="int" cardinality="1"></slot>
  <slot name="city" type="string" cardinality="1"></slot>
  <slot name="postal_code" type="int" cardinality="1"></slot>
  <slot name="district" type="string" cardinality="1"></slot>
  <slot name="metropolitan" type="boolean" cardinality="1"></slot>
  <slot name="other" type="string" cardinality="*"></slot>
  <condition>
    <literal object="domainaddress" slot="street"><value>*</value></literal>
    <literal object="domainaddress" slot="number"><value>*</value></literal>
  </condition>
</object>

<!------->

<object name="metropolitan">
<type_of>address</type_of>
</object>

<object name="rural">
```

```

<type_of>address</type_of>
</object>

<object name=" address">
<part_of>location</part_of>
</object>

<!------->

<object entity name=" greenpoint">

    <slot name=" address" type=" domainaddress" cardinality="1"></slot>
    <slot name=" timetable" type=" string" cardinality="1"></slot>
    <slot name=" name" type=" string" cardinality="1"></slot>

    <condition>
        <literal object=" greenpoint" slot=" address"><value>*</value></literal>
        <literal object=" greenpoint" slot=" name"><value>*</value></literal>
    </condition>
</object>

```

### C.3 The language patterns

We have organized the information involved in natural language generation by language. For each language that the system uses, there is an XML file that contains the data involved in each service, and the common data. The XML schema is organized into sections. Each section contains the information related to a specific service or related to the common data. Basically, the information consists of: the rules, the patterns, the concepts, the attributes, the lexical entries and the actions with no pattern. Figure C.1 shows an example of the data generated for the natural language generator resources. The operators that are used to describe the content are the following:

- `answ` and `ask` are the actions.
- `Directed`, `Mid` and `Open` indicate the category of the pattern.

`name` is a lexical entry.

- `<name>` is a pattern.
- `[$name]` is a slot for a concept.
- `[#name]` is a slot for an attribute.

- [*@name*] is a slot for a value.
- (*)* indicates an optional constituent.
- | separates different items.

```

answ<val_place> -->
  (the [\$event]) [@event] [verb_related_place] in [@place]
ask<search> -->
  Directed [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 C.1: A fragment of a language generation resource

## C.4 Acquisition of the resources

DIGUI is a dialogue system that is specially designed to access web services. Hence, most of the data that is related to the services are already available online. The system must acquire information about the applications and their domains, to feed the DS modules.

The LOC service allows the user to search the places in which pollutant objects are collected. Information about these places already exists on the web. The CA service also exists on several web sites. All the information about the events and places is online and can be used to create the language resources and the data for the specific task schema.

The domain knowledge is basically the information that the user and the system exchange. In our system, these data have been obtained manually (the tasks) or semi-automatically (the taxonomy of objects, the city streets).

### The large objects collection service

The following is the specific domain knowledge for the LOC: the green points, the objects and the city address. The green points are real places and their location does not change often. Information about the green points was manually extracted and stored in files. The objects were automatically extracted from the IKEA website and then they were manually

extended and classified as pollutant or non-pollutant. The names of the streets in the city were automatically extracted from the city's web site. On this web site, the citizens can search for a specific address and other services on a city map. In addition to the complete name of the streets, the city provided us with other short names that the citizens use.

### **The cultural agenda service**

The following is the specific domain knowledge for the CA service: the events, the participants, the locations and their facilities. The events and the participants in the CA service continuously change. The data become obsolete as the events are no longer available. New events appear every day. Our current implementation uses static files, that need a daily update so that the rest of the modules use the correct resources (namely, the natural language understanding and generator). A better solution is to use wrappers to access the web services periodically to update the resources. In contrast, the locations and the facilities are less volatile. The locations were acquired automatically from the municipality web service, whereas their facilities were written manually.

#### **C.4.1 Creating and adapting the resources to a new service**

DIGUI is a flexible system designed to ease the integration of other services. The system uses explicit models to represent the tasks and the dialogue, which decouples these models from the management processes. The NL modules also separate the parser and content generator processes from the specific domain and the language dependent data: the grammars, lexicons, and the surface generator.

The first step in integrating a new service into DIGUI is to obtain the service tasks and their structures. Basically, we need to identify the data that the user provides and the data that the system gives to the user. These data can be recognized easily from the other interfaces that are used to access the service. A rapid development is to identify the minimum set of parameters needed to access the services and to use them to configure a system baseline. The modularity of the system allows the resources to be refined iteratively.

The second step consists of generating the dialogue plans. Our system considers one plan for each task. Then, we break the task down into smaller ones. The plan can be written down

according to the schemas described in Chapter 5. Subsequently, the plans for the dialogue manager allow us to identify the set of system actions and attributes of the service.

The third step consists of generating the NL resources. Basic domain knowledge consists of the concepts and attributes that appear in the plans. Then, the domain knowledge can be extended with additional concepts, attributes and relations. The NLU needs grammars and lexicons from the domain knowledge to be able to interpret the user's interventions. These grammars have to generate the user's actions, as expected from the system (and identified in the plans). The NLG uses patterns and lexicons to generate the system's phrases. We have to classify the possible system actions and enumerate all the possible attributes of each action. These data can also be obtained from the plans. Finally, a table relates the system actions to the patterns that produced the sentences for the specific action.

Further refinements are possible once the basic resources have been generated. For instance, we can enrich the grammars, the lexicons and the domain knowledge, we can add further parameters and constraints to the tasks, or we can extend the plans with alternative ways to attain a user goal.



## Related publications

---

### **Combining User Intention and Error Modeling for Statistical Dialog Simulators**

Silvia Quarteroni, Meritxell González, Giuseppe Riccardi, Sebastian Vargas.

In the Proceedings of INTERSPEECH 2010.

Tokyo, September 2010.

### **Cooperative User Models in Statistical Dialog Simulators**

Meritxell González, Silvia Quarteroni, Giuseppe Riccardi, Sebastian Vargas.

In the Proceedings of 11th SIGDIAL Meeting on Discourse and Dialogue.

Tokyo, September, 2010.

In these two papers, we studied several characteristics of the user behaviour. In particular, we built a user simulator based on a corpus of dialogues and composed of several models.

### **Estructura y Gestión de Tareas en un Sistema de Diálogo para Acceder a Servicios Web**

Meritxell González, Marta Gatius.

In the Proceedings of XXVI Congreso de la Sociedad Española para Procesamiento de Lenguaje Natural.

SEPLN2010, Valencia, September, 2010.

This work presents our task model and management, and the results of the test done using the data collected in the previous evaluation.

### **Guiding the User when Searching Information on the Web**

Marta Gatius, Meritxell González.

In the Proceedings of 14th Workshop on the Semantics and Pragmatics of Dialogue, pg. 147--148.

In Pawel Lupkowski AND Matthew Purver (ed.)

Polish Society for Cognitive Science.

PozDial (SemDial), June, 2010.

In this work we present the latest system architecture, that includes the task manager. It also shows how we integrate the task manager with the response generator.

### **A Flexible Dialogue System for Enhancing Web Usability**

Marta Gatus, Meritxell González.

In the Proceedings of the 18th International World Wide Web Conference.

Publisher ACM Press 2009, ISBN:978-1-60558-487-4, pp. 1167-1168.

WWW2009, Madrid, April 2009.

In this paper we describe the adaptive module and the experiments we carried out to evaluate and compare the user satisfaction and other dialogue features.

### **Discourse Management in Voice Systems for Accessing Web Services.**

Marta Gatus, Meritxell González.

In the Proceedings of the 2007 Workshop on the Semantics and Pragmatics of Dialogue, pp. 165-166.

DECALOG 2007, Rovereto, May 2007.

In this paper we presented, in the context of the dialogue systems community, the design of the dialogue plans and dialogue strategies we developed in our ISU-based approach to dialogue management.

### **Natural Language Processing Technologies**

Marta Gatus, Meritxell González, Leonardo Lesmo, Pietro Torasso

Deliverable D3.2. February 2005.

### **HOPS Architecture Specifications**

Pablo Hernández, Jordi Sánchez, Ángel López, Sheyla Militello, Marta Gatus, Meritxell González, Eli Comelles, Leonardo Lesmo, Xavier Noria, Robert Salla, Carlos de la Morena,

Jose Antonio Fernández, Alberto Deiro.

Deliverable D4.1. May 2006.

### **Natural Language Processing and Dialogue Management Development**

Marta Gatius, Meritxell González, Eli Comelles and Leonardo Lesmo.

Deliverable D4.3. April 2007.

These deliverables correspond to the work we developed in the European Project HOPS (IST-2002-507967). The first describes the state of the art of natural language technologies and, in particular, the technologies we used. The second gives the details of the HOPS architecture: the design, the implementation and how to develop new applications within the platform. In the latter the reader can find a detailed description of the natural language processing and dialogue manager modules that we developed for the system.

### **An Information State-Based Dialogue Manager for Making Voice Web Smarter**

Marta Gatius, Meritxell González, Elisabet Comelles.

In the Proceedings of the 16th International World Wide Web Conference.

Publisher ACM Press 2007, ISBN: 978-1-59593-654-7, pp. 1315-1316.

WWW2007, Banff, May 2007.

Since a main directive of the system was to access on-line public services, we participated several times in the W3C Conference. Concretely, in this paper we presented the system architecture and how we integrated different components in order to improve the portability of the system.

### **A multilingual Dialogue System for Accessing the WEB**

Marta Gatius, Meritxell González.

In the Proceeding of 3rd International Conference on Web Information Systems and Technologies.

Copyright © INSTICC - Institute for Systems and Technologies of Information, Control and Communication ISBN: 978-9 72-8865-79-5, pp 184-189.

WEBIST 2007, Barcelona, March 2007.

This work describes how the dialogue systems can be used to improve the accessibility and usability of the existing web services and applications.

### **Integrating Semantic Web and Language Technologies to Improve the Online Public Administrations Services**

Marta Gatius, Meritxell González.

In the Proceedings of the 15th International World Wide Web Conference.

Publisher ACM Press 2006, ISBN: 1-59593-323-9, pp. 847 - 848.

WWW2006, Edinburgh, Scotland, May 23-26, 2006.

In this work, we discussed how ontologies can be used to represent the domain application and to improve the performance, development and integration of different modules in the dialogue system, as for instance, the natural language modules and the dialogue manager.

### **Using Application-Specific Ontologies to Improve Performance in a Bottom-up Parser**

Marta Gatius, Meritxell González.

In the Proceeding of the 11th Conference of the European Chapter of the Association for the Computational Linguistics. Workshop on Knowledge and Reasoning for Language Processing.

Publisher Association for the Computational Linguistics, ISBN: 1-932432-59-0, pp. 12-19.

KRAQ'06, Trento, Italy, April 2006.

The development of the ontology that represents separately the different types of knowledge involved in communication process favored the development and the improvement of several aspects of the natural language modules. This specific paper presents how the ontology can be used to improve the performance of the parser processing the user input.

### **Obtaining Linguistic Resources for Dialogue Systems from Application Specifications and Domain Ontologies**

Marta Gatius, Meritxell González.

In the Proceedings of the 10th International Conference on Speech and Computer.

ISBN 5-7452-0110-x, pp 737-740.

SPECOM 2005, Patras, Greece, October 2005.

This work describes how the application ontology facilitates the development of the linguistic resources for two different applications in four different languages and two channels.

These resources are used by several dialogue components.

**Un sistema de diálogo multicanal para acceder a la información y servicios de las administraciones públicas**

Meritxell González, Marta Gatius.

In the Proceedings of I Congreso Español de Informática (CEDI), XXI Congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural.

Revista núm. 35, pp. 285-292. ISSN: 1989-7553.

SEPLN 2005, Granada , Septiembre 2005.

In this work we described how the ontology describing the domain application can be used for rapid development and efficient interaction management of multi-channel and multi-lingual dialogue systems.

**The project HOPS: Enabling an Intelligent Natural Language Based Hub for the Deployment of Advanced Semantically Enriched Multi-channel Mass-scale Online Public Services**

Marta Gatius, Meritxell González.

In the Proceedings of I Congreso Español de Informática (CEDI), XXI Congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural.

Revista núm. 35, pp. 437-438. ISSN: 1989-7553.

SEPLN 2005, Granada , Septiembre 2005.

In this work we presented, in the Spanish Congress of Computer Science, the HOPS project and the work progress.

**Un sistema de diálogo multilingüe dirigido por la semántica**

Marta Gatius, Meritxell González.

Revista de la Sociedad Española para el Procesamiento del Lenguaje Natural

Revista núm. 34. Abril 2005.

This article, published in the journal of the Spanish natural language processing community, gives our initial dialogue system overview based on the development of an ontology for representing the application and domain knowledge and favoring different aspects related to the

different components involved in the dialogue system. The ontology serves for establishing the structure of the dialogue for the specific application, which in turn facilitates the development of the system resources.

### **Using Ontologies for Improving the Communication Process in a Dialogue System**

M. Gatius, M. González

In the Proceedings of the 6th International Workshop on Computational Semantics.

IWCS-6, Tilburg, Holland, 2005.

This paper presents how the ontology that represents the domain application can also improve the communication process at different levels, such as to process the user's interventions and to generate the system's responses.

### **Utilización de ontologías en el desarrollo de sistemas de diálogo**

Marta Gatius, Meritxell González.

In the Proceedings of III Jornadas en Tecnología del Habla.

3JTH, Valencia, Noviembre 2004.

This work describes the development of the ontology that models the application and promotes the development of the dialogue system components by explicitly identifying in the ontology the different types of knowledge that are involved in communication.

### **Ontology-driven VoiceXML Dialogues Generation**

Marta Gatius, Meritxell González.

In the Proceedings of the Berliner XML-Tage 2004.

Humboldt University, Berlin, October 2004.

In this paper we proposed a reusable representation of the knowledge involved in the application for rapid development of VoiceXML dialogues, including grammars and system prompts.

# Bibliography

---

- J. Allen, D.K. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. Toward conversational human-computer interaction. *AI Magazine*, 22(4):27--38, 2001a.
- J. Allen, G. Ferguson, and A. Stent. An architecture for more realistic conversational systems, 2001b.
- J. Allen, N. Chambers, G. Ferguson, L. Galescu, H. Jung, and W. Taysom. Plow: A collaborative task learning agent. In *Conference on Artificial Intelligence (AAAI)*, pages 22--26. Springer-Verlag, 2007.
- P. Andrews. *Persuasive Computer Dialogue Improving Human-Computer Communication*. PhD thesis, The University of York, United Kingdom, 2008.
- J.A. Bateman. Kpml: The komet-penman (multilingual) development environment. Technical Report, GMD/IPSI, 1994.
- J.A. Bateman and M. Zoch. *Natural Language Generation*. The Oxford Handbook of Computational Linguistics, Ruslan Mitkov, Oxford University Press, 2005.
- J.A. Bateman, R. T. Kasper, J. D. Moore, and R. A. Whitney. A general organization of knowledge for natural language processing: the penman upper model. Technical Report, USC/ISI, 1990.
- J.A. Bateman, B. Magnini, and F. Rinaldi. The generalized Italian, German, English upper model. In *ECAI-94 Workshop on Implemented Ontologies*, 1994.
- N. Blaylock and J. Allen. A collaborative problem-solving model of dialogue. In *In Proceedings of the SIGdial Workshop on Discourse and Dialog*, pages 200--211, 2005.

- B. De Carolis, G. Cozzolongo, S. Pizzutilo, and V. Silvestri. Mymap: Generating personalized tourist descriptions. *Applied Intelligence*, (26):111--124, 2007.
- J. Chu-Carroll and M.K. Brown. An evidential model for tracking initiative in collaborative dialogue interactions. *User Modeling and User-Adapted Interaction*, 8(3-4):215--254, 1998. ISSN 0924-1868. doi: <http://dx.doi.org/10.1023/A:1008205526147>.
- J. Chu-Carroll and H.S. Nickerson. Evaluating automatic dialogue strategy adaptation for a spoken dialogue system. In *1st Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 202--209, 2000.
- L.F. D'Haro. *Speed Up Strategies for the Creation of Multimodal and Multilingual Dialogue Systems*. PhD thesis, Universidad Politécnica de Madrid, Spain, 2009.
- L.F. D'Haro, R. Cordoba, J.M. Lucas, R. Barra-Chicote, and R. San-Segundo. Speeding up the design of dialogue applications by using database contents and structure information. In *SIGDial Workshop on Discourse and Dialogue*, pages 160--169, London, UK, 2009.
- J. Feng, S. Reddy, and M. Saraçlar. Webtalk: Mining websites for interactively answering questions, 2005.
- 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. Gatus. *Using an ontology for guiding NL interaction with knowledge based systems*. PhD thesis, Universitat Politècnica de Catalunya, 2001.
- M. Gatus 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. Gatus and M. González. Ontology-driven voicexml dialogues generation. In *Proceedings of the Workshop at the Berliner XML-Tage*, pages --, 2004b.
- M. Gatus and M. González. The project hops: Enabling an intelligent natural language based hub for the deployment of advanced semantically enriched multi-channel mass-scale online public services. In *XXI Congreso de la SEPLN, I Congreso Español de Informática*, pages --, 2005a.

- M. Gatus 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 --, 2005b.
- M. Gatus and M. González. Un sistema de diálogo multilingüe dirigido por la semántica. *SEPLN*, (34):--, 2005c.
- M. Gatus 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 --, 2005d.
- M. Gatus 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. Gatus and M. González. Integrating semantic web and language technologies to improve the online public administrations services. In *Proceedings of the 15th International World Wide Web Conference*, pages --, 2006b.
- M. Gatus 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. Gatus 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. Gatus and M. González. A flexible dialogue system for enhancing web usability. In *Proceedings of the 18th International World Wide Web Conference*, pages --, 2009.
- M. Gatus and M. González. Guiding the user when searching information on the web. In *Proceedings of the 14th Workshop on the Semantics and Pragmatics of Dialogue*, pages 147--148, 2010.
- M. Gatus and H. Rodríguez. Natural language guided dialogues for accessing the web. In *5th International Conference on Text, Speech and Dialogue*, volume 2448. Springer-Verlag in Lecture Notes in Artificial Intelligence subseries of LNCS series, 2002.

- M. Gatus, M. González, and E. Comelles. An information state-based dialogue manager for making voice web smarter. In *Proceedings of the 16th International World Wide Web Conference*, pages --, 2007a.
- M. Gatus, M. González, E. Comelles, and L. Lesmo. Natural language processing and dialogue management development. In *European Project HOPS (IST-2002-507967). Deliverable D4.3*, April 2007b.
- M. González and M. Gatus. 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.
- M. González and M. Gatus. Estructura y gestión de tareas en un sistema de diálogo para acceder a servicios web. In *XXVI Congreso de la SEPLN, Congreso Español de Informática CEDI 2010*, 2010.
- M. González, S. Quarteroni, Giuseppe Riccardi, and Sebastian Vargas. Cooperative user models in statistical dialog simulators. In *In the Proceedings of 11th SIGDIAL Meeting on Discourse and Dialogue*, 2010.
- 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.
- D. Griol, Z. Callejas, and R. López-Cózar. A comparison between dialog corpora acquired with real and simulated users. In *SIGDIAL '09: Proceedings of the SIGDIAL 2009 Conference*, pages 326--332, Morristown, NJ, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-64-0.
- C.I. Guinn. An analysis of initiative selection in collaborativetask-oriented discourse. *User Modeling and User-Adapted Interaction*, 8(3-4):255--314, 1998. ISSN 0924-1868. doi: <http://dx.doi.org/10.1023/A:1008359330641>.

- D. Guzzoni, C. Baur, and A. Cheyer. Active: A unified platform for building intelligent web interaction assistants. *Web Intelligence and Intelligent Agent Technology, International Conference on*, 0:417--420, 2006.
- M. Hadjadinjak and F. Mihelic. The paradise evaluation framework: Issues and findings. *Computational Linguistics*, 2(32):265--272, 2006.
- M. Hartmann, D. Schreiber, and M. Kaiser. Task models for proactive web applications. In *WebIST*, 2007.
- D. Jurafsky and J.H. Martin. *Speech and Language Processing*, chapter 24. Prentice Hall, 2008.
- 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, 1997.
- 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, 1998.
- 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.
- D. Louloudis, N. Fakotakis, and G. Kokkinakis. Mathematical modeling of dialogue control strategies for interactive dialogue systems. In *SPECOM*, pages 741--744, 2005.
- K. McKeown. Discourse strategies for generating natural-language text. *Artificial Intelligence*, (27):1--41, 1985.
- D. Milward and M. Beveridge. Ontologies and the structure of dialogue. In *CATALOG, 8th Workshop on the Semantics and Pragmatics of Dialogue*, 2004.

- D. Milward, G. Amores, T. Becker, N. Blaylock, M. Gabsdil, S. Larsson, O. Lemon, P. Manchón, G. Pérez, and J. Schehl. Integration of ontological knowledge with the isu approach. European Project TALK (IST-507802) Deliverable D2.1, 2005.
- I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 4:93--114, 2001.
- T. Paksima, K. Georgila, and J. Moore. Evaluating the effectiveness of information presentation in a full end-to-end dialogue system. In *Proceedings of the SIGDIAL 2009 Conference*, pages 1--10, London, UK, September 2009. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/W09-3901>.
- S. Quarteroni, M. González, G. Riccardi, and S. Vargas. Combining user intention and error modeling for statistical dialog simulators. In *INTERSPEECH*, Makuhari, Tokyo, Japan, September 2010.
- J.F. Quesada. Modelado de diálogo basado en conocimiento, acciones y expectativas. In *SEPLN*, number 29, 2002.
- C. Rich, C.L. Sidner, and N. Lesh. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15--25, 2001.
- V. Rieser and O. Lemon. Does this list contain what you were searching for? learning adaptive dialogue strategies for interactive question answering. *Natural Language Engineering*, 15(1):57--375, 2009. ISSN 1351-3249.
- S. Rosset, O. Galibert, G. Illouz, and A. Max. Integrating spoken dialog and question answering: The Ritel project. In *Proceedings of Interspeech 2006---ICSLP: 9th International Conference on Spoken Language Processing, Pittsburgh, PA, USA*, pages 1914--1917, 2006. URL [http://www.isca-speech.org/archive/interspeech\\_2006/i06\\_1529.html](http://www.isca-speech.org/archive/interspeech_2006/i06_1529.html).
- A. Sahuguet and F. Azavant. Building intelligent web applications using lightweight wrappers. *Data Knowledge Engineering*, 36(3):283--316, 2001. ISSN 0169-023X. doi: [http://dx.doi.org/10.1016/S0169-023X\(00\)00051-3](http://dx.doi.org/10.1016/S0169-023X(00)00051-3).

- J. Schatzmann, K. Weilhammer, M. Stuttle, and S. Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowl. Eng. Rev.*, 21(2):97--126, 2006. ISSN 0269-8889.
- 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.
- 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.
- D. Sonntag, R. Engel, G. Herzog, A. Pfalzgraf, N. Plfeger, M. Romanelli, and N. Reithinger. *SmartWeb Handheld: Multimodal Interaction with Ontological Knowledge Bases and Semantic Web Services*. Sringer, Heidelberg, 2007.
- M. Steedman and R. Petrick. Planning dialog actions. In *8th SIGdial Workshop on Discourse and Dialogue (SIGdial 2007)*, pages 265--272, 2007.
- 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, Psychology, 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.
- S. Vargas, S. Quarteroni, G. Riccardi, A.V. Ivanov, and P. Roberti. Leveraging pomdps trained with user simulations and rule-based dialogue management in a spoken dialogue system. In *SIGDIAL '09: Proceedings of the SIGDIAL 2009 Conference*, pages 156--159, Morristown, NJ, USA, 2009a. Association for Computational Linguistics. ISBN 978-1-932432-64-0.

- S. Vargas, F. Weng, and H. Pon-barry. Interactive question answering and constraint relaxation in spoken dialogue systems. *Natural Language Engineering*, 15(1):9--30, 2009b. ISSN 1351-3249.
- L. Villarejo, N. Castell, and J. Hernando. Dialogue management in an automatic meteorological 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 meteorological information system. In *Proceedings of the Berlin XML Days*, 2003b.
- W. Wahlster. Smartweb: multimodal web services on the road. In *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*, pages 16--16, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-702-5. doi: <http://doi.acm.org/10.1145/1291233.1291243>.
- 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. Wang. SALT: a spoken language interface for web-based multimodal dialog systems. In *Proceedings of ICSLP---Interspeech 2002: 7th International Conference on Spoken Language Processing, Denver, CO, USA*, pages 2241--2244, 2002. URL [http://isca-speech.org/archive/icslp\\_2002/i02\\_2241.html](http://isca-speech.org/archive/icslp_2002/i02_2241.html).
- J. D. Williams and S. Young. Partially observable markov decision processes for spoken dialog systems. *Comput. Speech Lang.*, 21(2):393--422, 2007. ISSN 0885-2308.
- J.D. Williams. Partially observable markov decision processes with continuous observations for dialogue management. In *Computer Speech and Language*, pages 393--422, 2005.
- 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.

- S. Young, M. Gasic, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu. The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Computer Speech & Language*, 24(2):150 -- 174, 2010. ISSN 0885-2308.