Using an Ontology for Guiding Natural Language Interaction with Knowledge Based Systems

Memòria presentada al Departament de Llenguatges i Sistemes Informàtics de la Universitat Politècnica de Catalunya per a optar al grau de Doctora en Informàtica

Marta Gatius i Vila

sota la direcció del Doctor Horacio Rodríguez Hontoria

Barcelona, setembre de 2000

AGRAÏMENTS

Voldria fer arribar el meu sincer agraïment a totes les persones que m'han ajudat durant aquest anys de recerca, que queden reflectits en aquest treball.

D'una manera especial, agraeixo al meu director de tesi, l'Horaci Rodríguez, tot el que m'ha ensenyat, la sàvia manera de guiar (i ordenar) la meva recerca, la inquietud intelectual que contagia.

Voldria donar les gràcies també a tots els companys del grup de Recerca en Processament del Llenguatge Natural i d'una manera especial a la Núria Castell, que em va introduir en aquest mon. Apendre i "digerir" amb ells els nous conceptes relacionats amb el llenguatge ha estat molt interessant.

En general, a tots als companys del Departament de LSI i especialment al grup de professors de Terrassa i als meus companys de despatx, amb els que ha estat un plaer compartir les hores de feina.

Vull agrair-li al Jordi la seva serena manera d'estar al meu costat en tot moment, la seva vitalitat i coratge que fa que qualsevol cosa sigui possible.

També vull donar les gràcies a la meva mare, als meus germans i als meus oncles, pel seu interès en el meu treball durant aquests tots aquests anys.

ABSTRACT

Since the 1980's, knowledge based systems (KBSs), programs that use knowledge to model situations and solve problems, have spread throughout industry, finance and science. Human communication with these systems deals with complex concepts and relationships that are not present in other software applications. Allthough the natural language (NL) is especially appropriate for expressing these concepts, there are not many KBSs incorporating NL interfaces. The main reasons for this are problems of efficiency in NLI performance, lack of adequacy to the communication needs of the applications and the high cost of developing and maintaining them.

The aim of this thesis is to study how the communication process and engineering features can be improved in NL interaction with KBSs. This study has been focused on the efficient and reusable representation of the knowledge involved in NL communication with KBSs. GISE (*Generador de Interfaces a Sistemas Expertos*), a system supporting NL communication with KBSs has been developed. This system adapts the general linguistic resources to application requirements in order to automatically obtain application-restricted grammars.

The main issue of the system design is a separate and reusable representation of all types of knowledge involved in communication with KBSs. This knowledge consists of the application knowledge appearing in the communication, the tasks of communication, the linguistic knowledge supporting their expression and the general relationships between conceptual knowledge and its linguistic realization. Three general bases were designed to represent all this knowledge: the Conceptual Ontology (CO), the Linguistic Ontology (LO) and a set of control rules.

Conceptual knowledge is represented in the CO. This conceptual knowledge includes domain and functionality issues. All knowledge required to model the applications as well as the description of all possible communication acts is provided in the CO. The CO is the skeleton for anchoring the domain and the functionality of the applications. The complexity of KBS performance makes a formal and explicit representation of their domain and functionality necessary.

The general linguistic knowledge needed to cover the expression in NL of the tasks the system performs is represented by means of the LO and a set containing all possible realizations of the application terms. The LO is domain and application independent.

The control information to relate the general linguistic knowledge to conceptual application knowledge in order to generate the application-restricted grammars is represented by a set of production rules.

The modular organization of the relevant knowledge into separate data structures provides great flexibility for adapting the system to different types of applications and users.

The grammars generated by GISE use expressive and precise language tuned to the application and adapted to the evolution of the communicative process. A menu-system to guide the user in introducing the NL is integrated into the GISE interface.

GISE has been applied to a couple of applications: SIREDOJ, an ES in law and a railway communication system.

CONTENTS

LEGENDS OF THE FIGURES

CHAPTER 1

INTRODUCTION

Since the 1980's, knowledge based systems (KBSs), programs that use knowledge to model situations and solve problems, have spread throughout industry, finance and science. Human communication with these systems deals with complex concepts and relationships that are not present in other software applications. The natural language (NL) mode seems to be an appropriate mode, especially for communication with KBSs. It is easier to use NL to express complex information than to use other modes of communication. However, there are not many KBSs incorporating a NL interface (NLI). The main reasons for this are problems of efficiency in NLI performance, lack of adequacy to the communication needs of the applications and the high cost of developing and maintaining them.

In this thesis, entitled "Using an ontology for guiding natural language interaction with knowledge based systems", NL communication between users and KBSs is analyzed. Some solutions to the most important problems in this communication are proposed.

1.1 BACKGROUND

1.1.1 Communication with KBSs

Significant efforts in commercial and research software have focused on the development of human-computer interfaces. Person-machine technology has evolved towards improvements both in the quality of the functionality of the interface and in the engineering features of the development process. Several lines have been followed in this evolution. The most significant of these are the separation of the application and communication information modules, and the adaptation of the communication process to applications and users.

Systems and tools to deal with the construction and maintenance of transportable communicative modules have been developed. The conceptual coverage of dialogues has been expanded to deal with more complex domains and applications. Important improvements in the friendliness of the interaction have been achieved by integrating different modes and media. Help tools that guide the user with respect to the interface possibilities have been incorporated.

Following this evolution, significant improvements in communication with simple applications have been achieved. There are, however, important problems still to be solved in interfaces to KBSs. Interfaces to KBSs deal with complex concepts not present in other software applications. Besides, in these applications, extremely complex communicative relationships arise, producing a closely coupled communicative/functional modular structure. Standard tools for building communicative modules widely used in conventional software engineering are not well suited to knowledge-intensive applications.

Therefore, an important quantity of research work developing interface systems for KBSs is being undertaken. Most of the systems developed in these research projects on communication with KBSs include the NL mode. The NL is a friendly mode used to express the complex concepts appearing in communication with KBSs.

1.1.2 NL communication

NLI technology started in the 1970's with NL access to databases, and has subsequently taken several directions, following the main trends of the evolution of human-computer interfaces and natural language processing (NLP). NLIs have evolved from simple interfaces recognizing a restricted form of NL towards more complex systems supporting rich enough linguistic coverage to deal with the increasing complexity of applications.

Although the basic techniques and systems were developed years ago and many attempts have been made to adapt NLIs to the application requirements, the introduction of NLI technology in software applications has not reached the level that was once expected.

The main reason for the poor presence of NLIs is the performance problems they still present. The run-time requirements necessary to process user interventions are large. Besides, because of the ambiguity of NL, mistakes are not infrequent. Another important problem in the functionality of the NLIs is lack of user knowledge with reference to the limits of the system capabilities.

The high cost of developing and maintaining NLIs is another important reason for their poor industrial presence. Linguistic sources needed in these systems are expensive to develop and neither the tools nor any standard methodologies facilitating this task currently exist.

NLIs adapted to the communication needs of the application reduce large run-time requirements. Linguistic resources adapted to the application have proven efficient especially when the sentences introduced by the user are limited to those supported by the grammar and lexicon. For this reason, many NLIs using an application-restricted grammar include a menu-system to guide the user in introducing the NL.

However, application-dependent linguistic resources are expensive to develop and difficult to reuse. Several ways have been attempted to reduce the cost of creating these resources. Most approaches adapt general linguistic resources to specific applications. The cost of such adaptation can be reduced when carried out automatically.

There are other relevant approaches for adapting linguistic resources to applications, such as incorporating dynamic mechanisms to the general grammar as a means of restricting the

grammatical options at run-time, or obtaining a general grammar adapted to a specific type of application.

These approaches have been followed successfully for simple applications, such as database consulting systems. Systems automatically generating application-restricted NLIs to relational databases have been developed. These systems reduce the cost of obtaining application-restricted grammar by adapting general grammar rules supporting consulting and updating to the concepts and relations of a specific database. However, these systems cannot be easily adapted to KBSs. The complexity of the knowledge involved in KBSs appears to be a major obstacle to obtaining application-restricted NLIs at a reasonable cost.

1.1.3 Reusing the knowledge involved in the NL communication with KBSs

The knowledge involved in the communication becomes larger as the complexity of the application grows. The reuse of the knowledge involved in communication reduces the cost of developing NLIs for different applications. A step can be made towards the transportability of NLIs and the reuse of their constituents by isolating and representing, in a declarative form, the conceptual and linguistic knowledge involved in a specific application. Increasingly, such information is represented in ontologies because this formal representation assures the consistency and reusability of the knowledge represented.

In order to adapt general linguistic resources to specific applications, general relationships between conceptual and linguistic knowledge must be defined. The optimal representation of the relationships between conceptual knowledge of general and application kind, and the specific language details needed for its linguistic realization, is a problem that has not yet been completely solved.

This thesis will propose GISE, a system supporting NL communication with KBSs. This system outlines a new approach in improving the development and performance of NLIs for complex systems. The approach consists of using a conceptual ontology for guiding NL interaction. General linguistic resources are adapted automatically to cover the application communication tasks represented in a conceptual ontology. The main issue of the system design is a separate and reusable representation of all types of knowledge involved in communication with KBSs.

1.2 AIM OF THE STUDY

The aim of this thesis is to study how the communication process and engineering features can be improved in NL interaction with KBSs. This study has been focused on the efficient and reusable representation of the knowledge involved in NL communication with KBSs. GISE (*Generador de Interfaces a Sistemas Expertos*), a system supporting NL communication with KBSs has been developed. This system adapts the general linguistic resources to application requirements in order to automatically obtain application-restricted NL-guided interfaces. The interfaces generated by GISE use expressive and precise language tuned to the application and adapted to the evolution of the communicative process.

In the system proposed in this thesis, the different types of knowledge relevant to communication with KBSs are represented in independent and general bases. These knowledge bases are easily adapted to specific applications. Such bases represent the application knowledge appearing in the communication, the tasks of communication, the linguistic knowledge supporting their expression and the general relationships between conceptual knowledge and its linguistic realization. The general bases representing all this knowledge are: Conceptual Ontology (CO), Linguistic Ontology (LO) and the set of control rules.

Conceptual knowledge is represented in the CO. This conceptual knowledge includes domain and functionality issues. All knowledge required to model the applications as well as the description of all possible communication acts is provided in the CO. The CO is the skeleton for anchoring the domain and the functionality of the applications. The complexity of KBS performance makes a formal and explicit representation of their domain and functionality necessary.

The general linguistic knowledge needed to cover the expression in NL of the tasks the system performs is represented by means of the LO. The LO is domain and application independent.

The control information to relate the general linguistic knowledge to conceptual

application knowledge in order to generate the application-restricted NLIs is represented by a set of production rules.

The modular organization of the relevant knowledge into separate data structures provides great flexibility for adapting the system to different types of applications and users.

The general linguistic bases were designed for communicating in Spanish and Catalan. Although transporting them to other languages has not been considered, it is not expected to pose major problems given that the LO was organized following the basic principles of a general grammar, the Nigel grammar, which is a large systemic functional grammar. The examples presented in this thesis are translated into English.

A menu-system to guide the user in introducing the NL is integrated into the interfaces generated by GISE.

GISE has been applied to a couple of applications: SIREDOJ, an ES in law domain and a consulting system in the railway domain.

1.3 OVERVIEW OF THE THESIS

This section describes the content and the organization of the thesis.

The thesis has been organized into nine chapters. This chapter introduces the work developed in the thesis. In the second chapter, an overview of NLI evolution is given, together with a survey of the use of ontologies in NLP systems. The third chapter describes the functionality and architecture of GISE, the system designed for improving NL communication with KBSs. The GISE CO representing all conceptual information required in the communication is described in the fourth chapter. The LO of the system is detailed in Chapter 5. Chapter 6 describes the production rules controlling the process of adapting the general linguistic knowledge to the CO in order to obtain the most appropriate application-restricted grammar and lexicon. The characteristics of the resulting NLIs supporting the NL communication with KBSs are given in Chapter 7. The applications of the system are described in Chapter 8. Finally, the general conclusions are provided in the last chapter.

All examples appearing in this thesis are obtained from the two applications quoted above.

The content of each chapter is summarized bellow.

Chapter 2: State of the Art

In this chapter a survey of the state-of-the-art of NL communication as well as of the use of ontologies in different types of applications is given. It is divided into two sections. In the first section, an overview on the most relevant work in the area of NLIs is given. Current trends in the related areas of NLP resources, dialogue modeling, multi-modal interfaces and speech mode are presented. In the second section, the use of ontologies to represent knowledge and its incorporation in current applications is discussed. The chapter concludes with an overview on the use of ontologies in most relevant NLP systems.

Chapter 3: The functionality and architecture of GISE

The main issues of the functionality and architecture of GISE, a system designed for communication with KBSs, are described in this chapter. The functionality and architecture of the system design is detailed in the two first sections. The process of adapting general knowledge to the specific knowledge needed for an application is described in the last section. The separate knowledge bases representing the different types of knowledge involved in this communication are detailed in the following chapters.

Chapter 4: Conceptual Ontology

CO, representing all conceptual knowledge necessary for supporting communication with KBSs, is described in this chapter. The general commitments followed in the CO design as well as examples of application concepts and relations represented in the CO are discussed.

Chapter 5: Linguistic Ontology

The LO, representing all general linguistic knowledge needed in the communication process is described in this chapter. Main decisions in its design are discussed. Examples of the linguistic structures required for expressing the communications tasks are given.

Chapter 6: Control rules

In this chapter, the production rules controlling the process of adapting the general linguistic resources to the application communication tasks are described. These rules generate the sublanguage required for communication with a specific application. The first section introduces the functionality of the control rules. The three steps to this adapting process are described in the second section. The formalism of the control rules is defined in next section. The basic set of control rules designed and implemented for generating the

interfaces for a broad type of applications is detailed in the third section. Finally, the last section provides an example following the performance of this basic set of ontrol rules.

Chapter 7: The Natural Language Interface

In this chapter, the characteristics of the application-restricted NLI used by GISE are given. The first section of this chapter gives a general gives a general description of the NLI. The different components of this interface are described in the following sections. The several possible forms of integrating the interfaces generated into the whole system are discussed in last section.

Chapter 8: Applications of GISE

Several applications of GISE are discussed in this chapter. Particularly, the application of the system to SIREDOJ, an advising ES in the domain of law is described in first section. Section 2 describes the application of GISE to a consulting system in the railways domain.

Chapter 9: Conclusions

The last chapter compares the main features of GISE with other systems following similar approaches and provides general conclusions. The contributions to this thesis as well as a number of suggestions for further research are outlined.

CHAPTER 2

STATE OF THE ART

This chapter presents a general survey of the state of the art in the two main areas related with the content of this thesis: NL communication and the ontologies.

2.1 THE EVOLUTION OF NL COMMUNICATION

Starting at the onset of the seventies, NLI systems have evolved towards improvement both in the functionality and the engineering features of the development process. The most significant lines followed in this evolution are the portability of the systems to different applications and domains, the expansion of the conceptual and linguistic coverage to deal with more complex applications and the integration of the NL mode with other modes of interaction.

An overview of the trends of this evolution and the most relevant NLI systems is given below.

2.1.1 The first NLI systems

The first NLI systems were developed during the seventies. Most of those first systems were designed for consulting DBMS. The greater part were built as monolithic systems with communicative and functional tasks fully integrated. They used simplistic pattern-matching and special purpose grammars developed on an ad-hoc experimental basis. In most of these systems, the NL process consisted of matching NL sentences to a

predetermined set of keywords. Besides, they were simply question/answer systems; they did not incorporate a rigorous treatment of the dialogue structure.

Some of the systems developed during the seventies however, made significant contributions to the NLI technology. Examples of those systems are LUNAR ([Woods72]), RENDEZVOUS ([Codd74]) and LADDER ([Hendrix77]).

LUNAR was the first system using Augmented Transition Networks (ATN) for NL processing.

RENDEZVOUS used a dialogue system for knowledge acquisition and disambiguation during the user consult.

LADDER was the first complete system supporting NL access to different relational databases. It used a semantic grammar.

The first commercial NLI systems were developed from these early systems. The most successful of the first commercial interfaces was NLI Intellect. It was developed from a previous research system, ROBOT. NLI Intellect supported interaction with different types of systems. It was developed by A.I.C. and commercialized by IBM. It was introduced in the market in the 70's but expanded in the 80's.

Despite several attempts being made, the commercial success of NLIs did not reach the level expected. The first NLIs had several important problems. Although NL communication has improved since then, some of its drawbacks have not been completely solved.

One of the most important problems inherent in the use of NL is the lack of user knowledge about the limits of the language recognized by the system. Even when the user is able to introduce any sentence, the systems can only correctly process a limited set of them.

The strategy of matching NL sentences to a predetermined set of keywords, although easy to develop, had very severe limitations:

- It was not possible to be precise enough in the match process.

- The amount of synonymy made exact keyword matching nearly useless.

Besides, as pointed out by Odgen in his study on existing NLIs ([Odgen88]), a principal problem in those NLIs was that they were adapted neither to the application functionality nor to the users.

Major improvements in interface design and development have been achieved by isolating communication and functional design (as proposed by Took in [Took90]).

The functionality of the interfaces has been improved with the use of a sublanguage adapted to the application. The friendliness of the communication improved when extending the coverage of the NL to support specific linguistic phenomena of this communication, such as abbreviations, acronyms, pronouns, ellipsis and coordination. The efficiency and friendliness of the communication process also improved with the incorporation of dialogue modeling to cooperate with users in expressing their needs.

However, NLIs adapted to the application and supporting a wide NL coverage are expensive to develop and maintain. For this reason, the design of transportable interfaces was a step forward in the evolution of the engineering development process. Since the middle of the eighties, relevant NL systems adaptable to different applications have been developed.

2.1.2 Improvements in the NLI development process: Transportable systems

The first transportable systems were consulting database systems. The transportable part of those systems was restricted to the conceptual domain. Adapting the linguistic resources of those systems to a relational database schema allows the portability of the interface to all relational databases. As the engineering features of the process of developing NLIs improve, transportable systems for other types of applications and languages were also built.

Relevant database consulting systems specially designed for transportation are TEAM ([Grosz87]), DATALOG ([Hafner85]), HAM_ANS ([HAM_ANS_85]) and ASK ([Thomson85]).

TEAM(Transportable English Access Data Manager) is a database consulting system. In TEAM, the transportable part is restricted to the conceptual domain and database organization. The language model uses an Extended Sintagmatic Grammar (AnaGram) to

parse user interventions. The parser information is represented in an internal logic. An independent module translates this information to the database consulting language.

DATALOG is a NLI system for DBMSs allowing the portability of the syntax and semantic process to different DBMS.

ASK is an NLI system designed to be adaptable to different DBMS, different database consulting languages, different linguistic coverage and even different software environments.

The system HAM-ANS is the first system supporting a rigorous treatment of dialogue. This system can be adapted to different applications, types of users and the types of dialogues.

Relevant work developed during the 80s for applications other than DBMSs are the XCALIBUR ([Carbonell83]), LDC-1 ([Ballard83]), UC ([Wilensky84]), KLAUS ([Grosz83]) and KING KONG ([Kalish87]) systems. The communication supported by the three first systems does not present more complexity than communication supported by consulting database systems. The actions and objects expressed in this communication are not very complex. The two lasts mentioned systems support communication with KBSs. The NLIs for KBSs require more linguistic and conceptual resources to deal with the complex phenomena that arise in this type of communication.

XCALIBUR is a system designed for accessing a DBMS that was adapted for accessing ESs and the Operating Systems. The performance of the final interface, however, resembles that of the NLIs to databases: it has been designed to retrieve information rather than to solve a problem. For processing using interventions, the system uses a semantic grammar close to the application.

The LDC-1 is an American system designed for developing NLIs for office environments.

The UC (Unix Consultant) is an NLI to the Operating System UNIX. It uses a semantic grammar close to the system. This system has made an interesting contribution in the area of NL generation. Its NL generation component, KING ([Jacobs87]) builds NL responses from semantic descriptions represented in frames.

KLAUS (Knowledge Learning And Using Systems) is a project for developing acquisition knowledge systems. The resulting systems would be the NLIs to different types of applications. The main contributions of the system are the treatment of the conceptual and

lexical knowledge acquisition as well as that of the logical inferences supported by the system. The syntactic representation is PATR_II. The semantic representation is an internal logic similar to the first order logic.

KING KONG is a system developed using the TEAM initial model. It was designed for ESs oriented to task execution. It was applied to different types of ESs, such as air mission planning and an automatic software generator.

Following the main trends of the evolution of human computer interfaces and NLP, systems and tools to deal with the construction and maintenance of transportable communicative modules and their integration into the whole system have been developed since the late eighties. These systems incorporated different types of modules: lexicon, grammar, knowledge base graphic modules, menu modules, etc. Examples of relevant work in this area are the FRED ([Jackobson86]), INKA ([Freiling84]), NAT ([Coch91]) and NL_MENU ([Thomson86]) systems.

FRED is an Intelligent Database Assistant. It incorporates different mechanisms to cooperate with the user. It allows consultation with DBMS implemented in different database languages (such as Focus, Oracle and SQL).

NAT contains a set of tools for building interfaces for different languages and different types of applications.

INKA is a tool for knowledge acquisition during the building of an ES. It uses an Interface Structured Language. The syntax of the Interface Structured Languages is similar to English, and the semantics is close to the tasks to be performed. The most relevant example of these languages is INGLISH (INterface enGLISH), a language developed by Phillips to create task model-based English interfaces. Using a language similar to English and semantically restricted to the tasks is an efficient approach, because the language options available to the user are completely application-restricted. However, it requires a formal application model because all the information that is needed to determine whether a particular statement can be executed must be encoded in the language.

This approach has been applied successfully for building NLIs to consult databases, as proved in the NL-Menu, a commercial system developed by Texas Instruments. The NL-Menu is a system generating NLIs to relational databases. The NLIs generated by the system include a menu-system to help the user when introducing the sentences. The menu-system controls the displaying in the screen of the NL options acceptable to the system at

each step of the communication process. The cost of the construction of the linguistic sources required in these interfaces is low, because a limited number of statements are needed in database consulting interfaces, and the semantic restrictions encoded in the grammar are few. Nevertheless, the construction cost grows in interfaces to complex applications, such as KBSs, in which the number of statements the users may need to say increases. Furthermore, in most KBSs, no schema or description is available. The propositions in those systems may have arbitrary meanings. Relations between propositions are not clearly defined.

2.1.3 Dealing with more complex applications: The NLIs for KBSs

The great complexity and the size of the accepted language in NLIs for KBSs is a problem when developing, tailoring and maintaining the sources needed for NLIs. The absence of tools to lend assistance and the low level of reusability of language modules make these processes expensive.

Another major difficulty comes from the KBSs functionality. The processing of a broad language is time consuming. Besides, while DBMS are only expected to supply the information the user requests, KBSs are designed to be problem solvers. Users consult these systems about an issue, and the systems must then gather information in order to advise the users. NLIs to KBSs handle a number of questions that could not be previously handled by the application, such as questions about general properties. The NLI must be responsible for managing all the new information to translate user inputs into facts and goals of an underlying KBS. A structure providing a foundation for the translation is necessary. It is also desirable that such a structure be general and transportable from one system to another. The NLIs may also answer specific questions without invoking the inference process.

An additional problem inherent in the use of NL for KBSs is that it allows the user to enter information in no particular order, whereas other modes, such as menus, predefine this order.

Despite all these problems, NL seems an appropriate mode for interfaces to KBSs because NLIs prove to be concise and efficient when the universe of possible messages is large.

The semantic complexity of NL becomes more useful as the problem-solving and reasoning capabilities of target programs grow. Discussions on the advantages and disadvantages of NLIs for KBSs can be found in different articles such as those of [Pollack82], [Rich84] and [Carbonell88].

NLIs for KBSs are used not only during the exploitation phase but also in the building phase, in which the application knowledge acquisition is done through NLIs. Examples of NLIs used in knowledge acquisition are the system KLAUS, described above and those described in [Freiling84] and [Datskovsky87]. Examples of NLIs used in the exploitation phase are the KING KONG system described above, the system described in [Ryan88] and that described in [Moerdler87].

The improvement of the friendliness and the efficiency in NLIs for complex applications has been achieved by enriching language coverage, incorporating knowledge about the system, following dialogue modeling methodologies, and the application and integration of different modes and media of interaction. All these trends in NLI evolution are discussed bellow.

2.1.4 Enriching the linguistic resources

The need for a rich enough language to deal with the increasing complexity of applications as well as the evolution of NL resources has resulted in interfaces incorporating complex NLP. NL communication has a strong need for different types of linguistic resources, especially for grammars and parsers for processing user interventions. New trends and improvements in this area are rapidly incorporated in NLIs.

There have been interesting works on NLIs implementing modern theories of grammars. An interesting antecedent of those works is CHAT-80 ([Warren82]). This system is the most well-known of the Logic Program based Interfaces. It was the first system in using a Definite Clause Grammar (DCG). The system knowledge as well as the semantic interpretation is represented in first order logic. Although unlike those described above, it is not a complete system, its treatment of specific linguistic problems, such as plurals and quantification is especially interesting.

One of the first successful attempts to implement a grammatical theory in a rigorous and efficient way was LOQUI ([BIM87]). This system was developed from the HAM-ANS system. LOQUI, a multilingual database interface system. LOQUI was developed as the NL component of the European project LOKI ([Binot88]), a Logic Oriented Approach to Data and Knowledge Bases supporting NL Interaction. The aim of the LOKI project was to provide a set of advanced tools based on Prolog for improving the support of, andaccess to, data and knowledge bases. The portability of LOKI is provided by a general domain-independent logical representation of the meaning of the input sentences. This logical representation supports reasoning and inferences on the knowledge of the application domain. LOQUI is a highly modular multilingual portable system, which can be adapted to new domain and database management systems. The system supports English and French, although the modular organization of the system also allows the incorporation of other languages. The English module uses a rigorous and efficient implementation of the Generalized Phrase Structure Grammar (GSPG). The German module uses a LFG formalism.

There are many other interesting systems using a well-developed implementation of a modern grammar. Most of these works use unification grammar, integrating syntax and semantic analysis, such as those described in [Moreno92], [Moreno93] and [Rodríguez89]. An overview of the unification grammar formalisms and their treatment is given in [Rodríguez95]. A more general survey of the state of the art on the rule-based grammar resources in current NLP systems is given in [Cole96].

There are also NLI systems supporting multilingual access, such as the European project SESAME ([Sabbagh90]), a French and English database interface system, NAT ([Coch91]) for French, English and Spanish, DABINAL ([Solak91]) for Polish and English and HSQL([Ljungberg91]) for Nordic languages.

Linguistic resources for NLP are expensive to develop and maintain. As pointed out in the recent survey of the state of the art in the multilingual information management given in [Hovy99], special interest has been paid in recent years to defining standards in developing and evaluating linguistic resources in order to facilitate their reuse and improvement. There are many projects in this field developed by the American government programs (DARPA) and the European Community programs (CEC).

As discussed in the first chapter of the above-mentioned report, the most relevant point in the evolution of the development of grammars is the gradual recognition of the impossibility of generating a complete grammar for any language, and so, the development of grammars for broad-coverage applications. The recognition of the amount of domain-specific essential lexical and syntactic information has led to the development of domain and application adapted grammars.

Linguistic resources for language processing and for generation have not yet come together, as pointed out by Bateman in [Bateman97b]. Since early works in generation (such as that of [Appelt85] and [McKeown85]), the functional and pragmatic issues have been considered as the central areas when organizing linguistic resources. In contrast, most relevant works on analysis consider structure and syntax as the central areas. Different approaches in generation focus the role of function to a greater or less degree. Some subordinate it entirely to structure, some attempt to combine structure and function, others consider communicative functions as occupying first place. Many of the approaches belonging to the third group use grammars based on systemic-functional linguistics (SFL), as the KMPL, described in [Bateman97a]. These approaches emphasize the role of the paradigmatic organization of resources in contrast to their syntagmatic organization.

As Bateman states in [Bateman97b], a paradigmatic organization has proved appropriate for the design of grammar development environments, especially for working multilingually. Paradigmatic functional organizations are more likely to show substantial similarities across languages than are syntagmatic structural descriptions.

Most existing NL generation systems do not use, however, reusable, general resources. There are many generation systems using simple, task-oriented template-based techniques. There are also intermediate approaches using restricted linguistic resources adaptable to different domains or applications. An example of theses systems is that described in [Caldwell94], a system for generating job descriptions in English and French. These descriptions are generated from a domain conceptual representation related to its linguistic expression. The language generator is based on a Montague type grammar. The system described in [Busemann98], which uses a domain-motivated linguistic ontology supporting rapid adaptation to new tasks and domains, is also a work of some interest.

Another useful distinction when comparing generation systems refers to the unit of language they generate: single phrases, single sentence or connected text. A detailed current state of the art on NL generation systems is given in [Zock96].

2.1.5 The use of ontologies in NLI systems

The complexity of conceptual and linguistic knowledge involved in NLIs for KBSs makes its representation more important in separate and declarative sources. A step can be taken towards the transportability of NLIs and the reuse of their constituents by isolating and representing the conceptual and linguistic knowledge involved in a specific domain or application in a declarative form. For this reason, many of the recently developed NLIs for KBSs incorporate an efficient and transportable representation of the linguistic components. Additionally, some of them incorporate ontologies to represent the conceptual knowledge needed in the processing of language.

Recent research work on knowledge representation and NLP emphasizes the advantages of organizing conceptual knowledge according to an ontology in which objects, concepts, relations and other entities appearing in a domain are explicitly defined. The representation of this information in an explicit and formal organization improves clarity, consistency, accessibility, extendibility and reusability. An overview of the state of the art of the use of ontologies in NLP and in other computational applications is given in Section 2 of this chapter.

2.1.6 The linguistic coverage in current commercial systems

The evolution towards the building of transportable systems incorporating complex NLP has achieved good results in the functionality and development of NLIs. The language coverage has been extended to support complex linguistic phenomena, such as quantification, negation, simple cases of references, ellipsis, incorrect words, etc. There are, however, other linguistic phenomena still to work on, such as subordination, different types of anaphora, nominal groups supporting various complements, the use of non-grammatical expressions.

The evolution described above has obtained especially good results in the functionality and development of NLI to DBMSs. A more in-depth description of NLIs to DBMSs is given in [Androutsopoulos95]. Also interesting are the works of [DeJong93] and [Sijtsma93] comparing the language supported by commercial NLI to databases. As discussed in these works, the treatment of complex linguistic phenomenon is also incorporated into current commercial interfaces. There are commercial interfaces supporting coordination of nominal groups and sentences, the use of pronouns, ellipsis, reference and negation. Many of these systems also support synonyms and different ways of expressing a request. Besides, most of them incorporate some kind of process for mistake recovery.

The most successful commercial systems during the middle of the decade were Q&A and HAL. Q&A was developed by Symantec. It runs over PC and supports many languages. HAL is an interface to Lotus 1-2-3 and was developed by Lotus Inc.

In the late eighties, two products from the BBN Laboratories, PARLANCE and NLI DATATALKER, attracted significant financial support. There were also other interesting systems such as the NL-Menu, described above, and the NLI, developed by NLI Inc. Some of the systems resulting from the research projects outlined in this discussion were also commercialized, such as LOQUI, commercialized by BIM, NAT and SESAME.

Nevertheless, these systems did not achieve the commercial success expected. As described in [Brunner90], the functionality of the NLIs not only depends on the sophistication and breadth of their linguistic coverage but also on how well this is integrated with direct manipulation methods to help usability. Direct manipulation methods, such as graphics and gestures are most appropriate for expressing simple queries, while complex questions require a language with richer semantics.

NL seems an appropriate mode for complex queries, such as those supported in intelligent consulting systems, as, for example, the TAMIC system ([Bagnasco96]), an Italian system for public administration consulting supporting deduction.

Furthermore, the integration of the NL with other modes in order to achieve friendly and efficient communication is currently an active area of research. The increasing complexity of the communication supported in the resulting systems makes the incorporation of dialogue modeling methodologies more necessary.

2.1.7 Dialogue modeling

The incorporation of dialogue modeling to adapt communication to the different tasks, modes and users of the applications improves efficiency and effectiveness in communication.

As described in the survey of the state of the art of discourse and dialogue given in [Cole96], current approaches are based on the five predominant theories given below.

Hobbs theory ([Hobbs85]) proposes a limited set of coherence relations to be applied to discourse segments.

Grosz and Sidner theory ([Grosz86]) describes the structure of the discourse according to the speaker's focus of attention (the attentional state), the structure of the speaker's purpose (the intentional structure) and the structure of sequences of utterances (the linguistic structure). This theory, as that of Hobbs is appropriate for NL processing.

The Mann and Thompson theory ([Mann87]) is also known as the Rhetorical Structure Theory (RST). This theory proposes a hierarchical organization of text spans, spans being either the nucleus or satellite of one of a set of discourse relations. This theory is appropriate for generation.

Mckeown theory ([McKeown85]) proposes a hierarchical organization of discourse around fixed schemata. This schemata drives content selection in generation.

Kamp theory ([Kamp81]) is known as Discourse Representation Theory (DRT). This theory was developed for representing and computing trans-sentential anaphora and others forms of text cohesion. Previously mentioned theories are sentenced based and therefore did not deal with the cross-references appearing in dialogues. This theory has already been used in the design of sophisticated question-answering systems.

The procedures developed for modeling dialogue follow three different approaches: dialogue grammars, plan-based models and joint action theories.

Dialogue grammars try to model the sequencing regularities in dialogue ([Sachs78]), such as questions being followed by answers, proposals followed by acceptances, etc. These grammars are a useful computational tool in expressing simple regularities of dialogue, but they need to function with plan-based approaches for complex systems.

Plan-based models of dialogue consider utterances as communication actions or speech acts ([Searle90]), such as requesting, informing and confirming. These theories assume that the speaker's speech acts are part of a plan, and this has to be uncovered. The main drawback of this theory is that the processes of plan-recognition and planning can be combinatorially intractable.

The joint action theories are theories regarding dialogue as a joint activity, something that agents undertake together ([Clark86]). These theories propose a new strategy to deal with reference and confirmations.

One problem of the theories described above is that most of them are based on dialogues between humans. Human computer interactions have their own sublanguages whose characteristics often allow a much simpler dialogue model than models capturing human interaction.

Many existing NLI systems incorporate a dialogue model much simpler than those theories described above. NLI systems achieve a friendly and efficient communication when users are able to express the commands and queries that the background system can deal with and that the system can react to both quickly and accurately.

A recent study on dialogue management in NL systems is that of Luperfoy in [Luperfoy96].

2.1.8 Multimodality

Important improvements in the friendliness and efficiency of NL communication has been achieved during the 90s thanks to the development of multimodal interfaces incorporating NL mode.

Multi-modal interfaces, integrating human perceptual processes such as vision, audition and tactility, have focused a great amount of computational effort. The integration of NL with other modes of interaction such as menus, speech, graphics and gesture provide the most efficient and natural communication for a broad range of applications.

The distinction between multimodal and multimedia systems is not the same for all authors. For example, Maybury in [Maybury98] defines multimedia as the physical means

through which information is input, output and/or stored, defining multimodal as the human perceptual processes. Other researchers, such as those in project MMI2 ([Binot90]) consider that the multimodal systems are only those including several input and output media and committed to a single internal representation language for the information. They consider multimedia systems to be those using individual-specialized representation for each mode.

Using individual-specialized representation is more efficient for storage and processing. However, using a single internal representation language permits the same representation to be presented in any mode and, as a consequence, the selection of the most appropriate mode at each state of the communication. The representation of the information expressed in such different modes, such as NL and graphics or gesture in a single language deals with important difficulties. Restricting this information to a specific domain solves most of these difficulties.

Initial work in this area is that of Hayes and his Carnegie-Mellon group ([Hayes87]). The system developed by this group integrates the NL and graphic mode. This system deals with the problems that arise when using more than one mode of interaction in a cooperative way. The two basic problems are deciding the most appropriate mode at each step of the communication, and treating the anaphoric reference between the two modes.

Some of the most important studies carried out on NLIs for KBSs are in the area of multi-modal interfaces. An overview of NLI systems for KBSs during the late eighties and early nineties is given in [Gatius92]. As pointed out in this work, the use of a common meaning representation for the information introduced through the different modes is a major problem in these systems. This representation must efficiently support NL semantic complexity as well as the different characteristics of other modes, such as graphics. A difficult problem to solve is reference between the modes.

XTRA ([Allgayer89]), SAUCI ([Tyler88]), ACCORD ([Chappel89]) and MMI2 ([Binot90]), are examples of some relevant multimodal systems for KBSs integrating NL.

XTRA is an Interface system for System Experts. The input modes supported by the system are NL and gesture and the output mode uses graphics and tables.

The ACCORD system (Construction and Interrogation of knowledge bases using natural language text and graphics) support English, French and German access. The processing of English and French is carried out using a Categorial Unification Grammar (UCG). The

processing of German uses an LFG. The semantic interpretation uses the Indexed Language, a typed first order logic based on the Discourse Representation Theory of Kamp.

SAUCI (A Self Adaptative User Computer Interface) is a multimodal system applied to the automatic manufacture domain. The input modes are English and graphics. The meaning representation language used is first order logic. This language supports anaphoric references between the two modes. It uses a complex dialog control.

MMI2 (A Multi-Modal Interface for Man Machine Interaction with Knowledge Based Systems) is a multimodal interface system for building KBSs in different domains. The input modes of the system are NL English, French and Spanish, gesture, direct manipulation of graphics and command language. The output modes are English, graphics and non-verbal audio. The meaning representation language used for all information is a typed first order logic with relativised quantification and second order relation symbols.

The dialogue control in multimodal systems is complex. An interesting work in this area is the project CFID ([Harper87]). The goal of the project CFID (Communication Failure in Discourse: Techniques for detection and repair) was to study mistakes in human-machine communication. The project developed a Database Consulting System using NL and gestures. The NLs supported are English and French. The grammar formalism is LFG. The common meaning representation is situation semantics. The developers determined that, although situation semantics was interesting at the sentential level, the theory did not cover discourse very well.

Mutimodal systems have also been developed for applications providing access to different types of knowledge (text, maps, pictures, etc.), such as the CUBRICON system ([Neal90]), and the Italian ALFresco system ([Stock91]), for an information system on art. The two systems interact with the users using NL (spoken and typed) and graphics. They also provide multimedia access in the form of various physical devices to interact with the user.

Recently, multimodal/multimedia systems have been developed for a broad variety of applications. Many systems in the area have been developed during the last decade, such as the HITS system, described in [Holland91] and the DETENTE system ([Wroblewski91]). Other relevant multimodal/multimedia systems are described in [Maybury93]: the COMET and the WIP systems, generating coordinated explanation in NL and graphics, the AIMI system, on the domain of military maps and the JETA system, for repairing jet engines.

The complexity of the new applications, as well as the development of the different modes and media of communication, increases the need to incorporate knowledge information into the systems. This knowledge allows the adaptation of the system at each state and mode of the communication, and to assist the users. Because these systems incorporate knowledge, they are also called intelligent multimodal systems. Most of the multimodal systems mentioned adapt the theories of discourse developed for NL mode to other modes of interaction. A central point in the design of current multimodal systems is the automatic generation of coordinated and coherent multimodal presentations during communication with the user.

The integration of different modes and languages offers new research opportunities, such as summaries in different languages using different media. An example of such current works is that described in [Merlino99], on the optimal presentation of multimedia summaries of Broadcast News.

As pointed out by Maybury and Stock in the overview on multimedia communication given in [Hovy99], once the problem of integrating different modes and media in increasing development is solved, the multimodal and multimedia systems will be those best suited to many applications. Interaction in these systems will be more efficient, enabling more rapid task completion with less work, more effective, adapting the interaction to the context (user, task and dialogue) and more natural, supporting different modes as interacting with a human interlocutor.

2.1.9 Speech

One of the most relevant facts in the NLI technology evolution is the increasing commercial presence of spoken language applications (i.e. Dragon Systems, IBM, Apple, Kurzweil) during this decade.

Even though the use of the speech mode has been a topic of interest since the beginnings of the development of NLIs, its importance increased throughout the eighties and nineties. The main causes of this increasing importance were the gradual commercialization of the interfaces using speech and its integration into the multimodal systems.

The increasing interest in spoken system is reflected in the high participation is the main congress in the field, such as EUROSPEECH, ICSLP, IEEE and ICASSP.

There is work still to be done on spoken interfaces in order to integrate them into multimodel systems, where the user and the systems select the most appropriate mode of communication.

Spoken interfaces are necessary in applications where no other mode of communication is possible, such as applications for handicapped, telephone applications, applications from vehicles, such as those giving information about the traffic or a route. They may also render more efficient current services with important performance shortcomings.

The most relevant projects in spoken systems are the European CEC SUNDIAL ([Peckham93], [Gerbino93], [Giachin97]) and the ATIS, founded by ARPA ([Seneff91], [Ward94], [Pieraccini97]).

The Dialogos system for accessing the Italian Railways timetables ([Popovici97]) and the RailTeL system for accessing the French Railways timetables ([Lamel97]) are examples of telephone consulting systems in the domain of trains. Examples of spoken consulting systems in other domains are the Danish dialogue system for consulting and booking flights ([Dybkjaer96]) and the Dutch SCHISMA system for consulting theatre information and booking ([Hoven94]).

There are also relevant projects currently being developed, such as ACCeSS (Automated Call Center trough Speech Systems), REWARD (Real World Applications of Robust Dialogue) and ARISE (Automatic Railway Information Systems for Europe), a train consulting system supporting Dutch, French and Italian.

The most important recent work in the area has studied solutions to solving the problems of recognizing spontaneous speech. A substantial body of this work proposes the use of spoken dialogue models. Examples of these are: the works of Bilange ([Bilange91]) describing a task independent oral dialogue model, Bobrow ([Bobrow77]) describing a frame driven dialog system, Popovici ([Popovici97]) on language modeling for task-oriented domains, and Huguenard ([Huguenard97]) describing a model of phone-based interaction to generate predictions about possible failures for an application. There are also many other interesting proposals for improving the robustness of conversational systems, such as the work of Goerz ([Goerz99]) on interactivity in all levels of processing and the

work of Nakana ([Nakana99]) on understanding unsegmented user utterances in real-time systems.

An increasingly important issue for spoken interfaces is their evaluation. Interesting works in this area are Johnston's study on the advantages and disadvantages of multi-modal interaction over speech-only interaction ([Johnston97]) and that of Walker [Walker97], describing a general framework for evaluating spoken dialogue agents where user satisfaction, task success and dialogue cost performance measures are combined.

2.2 THE USE OF ONTOLOGIES

The word *ontology* is controversial in the context of AI. Guarino discusses the possible interpretations of ontology in his terminological clarification in [Guarino95]. A common general interpretation of this word is Grüber's definition: *an ontology is an explicit specification of a conceptualization*. In Grüber's definition, the meaning of conceptualization is an abstraction of the entities assumed to exist in an area, and the relationships between these entities.

From the theoretical point of view, some of the most important contributions in the modern evolution of knowledge representation are the works of Brachman and Newell, supporting distinguished representation levels. *The epistemological level* proposed by Brachman ([Brachman79]), in which primitives defining concept types and structured inheritance relations are established, is present in all current languages used in implementing ontologies. The stringent separation between conceptual and linguistic knowledge and its representation, proposed by Newell in his theory describing *The Knowledge Level* ([Newell82]), is assumed in ontology design. A new level, *the ontological level*, has recently been distinguished by Guarino ([Guarino93]) to constrain knowledge primitives and thus build more understandable and consistent ontologies.

The need for agreement on the basic primitives and ontological commitments in ontologies has been stressed in most works on the subject, as in Gómez-Pérez ([Gómez-Pérez98]), in which most well-known ontologies are summarized. The Frame Ontology, described in

[Gruber93], plays an important role as an example of metaontology. It captures the representation of the primitives most commonly used in frame-based representation languages. The Frame Ontology is used in Ontolingua, a language for building shared ontologies.

2.2.1 The purpose of most relevant ontologies

Although the high cost of the development of large knowledge bases highlights the need for methodologies to develop ontologies and to enable knowledge sharing and reuse, most systems are developed from scratch (the design of ten of the most representative ontologies was compared by Fridman and Hafner in [Fridman97]. There is no consensus regarding a sound methodology for building ontologies, nor are there any standard tools for assistance. Important recent works defining theoretical principles in ontology design include Guarino (mentioned above) and Grüinger (described in [Grüinger95]). The need to define the purpose of the ontology in order to organize it appropriately has been widely recognized. The main issues in current ontology design are the expression of knowledge in NLP, knowledge sharing and integration, and the formal representation of applications and domains where complex organizations are required. All these issues are described below.

2.2.2 Ontologies for NLP

Increasingly, NLP systems use an ontology to represent the conceptual knowledge needed in the processing of knowledge. As pointed out in the survey on multilingual resources in [Hovy99], the ontologies of NL systems are usually simple, the concepts and the relations between them are not formally defined, and the number of concepts represented is not very large. Most of these ontologies are linked to lexicons to provide the words for expressing the concepts and relations represented. The Generalized Upper Model (GUM) ([Bateman90]), a general knowledge representation for use in different NLP systems, is one of the best-known examples of a general ontology organized for NLP. GUM contains approximately 300 high-level abstractions of English syntax and was designed as the top level of the SENSUS system, described in [Hovy88]. The SENSUS system integrates

large-scale linguistic ontologies for the machine translation of several languages, text summarization and generation. The concepts in the SENSUS ontology are linked to lexicons of different languages: Japanese, Spanish, Arabic and English.

There are also general world representations designed for general purposes that are organized, either wholly or in part, according to the expression of knowledge in natural language. A well-known example of these ontologies is CYC ([Lenat90]), a massive effort to formalize common-sense knowledge, containing approximately 40,000 concepts. A more detailed description of the use of ontologies in NLP is given below.

2.2.3 Ontologies for knowledge sharing and integration

Knowledge sharing between applications has been the objective of many ecently developed ontologies. A clear example of the importance of knowledge sharing is the Knowledge Sharing Effort, a consortium for developing conventions and supporting technology for knowledge representation ([Neches98]). This project involves participants from over a dozen different research centres. The main subjects developed within this project are the translation between different representation languages (Interlingua), the knowledge representation system specifications and the consensus on contents of sharable general and domain knowledge bases. The most important results of this project are the Knowledge Interchange Format (KIF), the Process Interchange Format (PIF) ([Lee96]), the Frame Ontology and Ontolingua. KIF is a format for representing and interchanging knowledge. PIF is a format for sharing heterogeneous software process descriptions. The Frame Ontology is a metaontology implemented in KIF. Ontolingua is a language for writing portable ontologies based on KIF. Translators from Ontolingua into some of the main representation languages (such as LOOM, Epikit and KIF) have been built. Ontolingua is also the language used in the Ontology Server, a set of tools and services to build shared ontologies between geographically distributed groups developed by the ARPA project.

There are other important works for knowledge sharing and translation, like the METHAONTOLOGY framework ([Fernández97]) and the Ontology Design Environment (ODE) ([Blázquez98]). The METHAONTOLOGY framework describes the different tasks implied when building an ontology. ODE is a software environment providing the user a

set of knowledge representations independent of the target language that can be translated to different languages (SQL, SFK and Ontolingua).

The integration of ontologies has also been the subject of major works. Two examples are the ONIONS project (Ontological Integration on Naïve Sources) to integrate medical ontologies and the SENSUS project to integrate large-scale linguistic ontologies.

The ONIONS project, described in [Gangemi96], proposes a methodology for integrating taxonomic knowledge that has been applied to task-oriented expert medical knowledge bases. The basic problem described in building a general ontology to integrate several different knowledge bases in a medical domain is the lack of a formal description of explicitly intended meanings. Specific ontologies for different taxonomic sources have been defined and integrated in a general ontology.

The SENSUS ontology ([Hovy88]), referred to above, is a combination of some well-known linguistically motivated ontologies: the GUM ([Bateman90]), the top level of the ONTOS ontology ([Nirenburg92]), the LDOCE (Longman Dictionary of Contemporary English) ([LDOCE78]) and WORDNET ([WODNET98]).

A major problem in most of these works on knowledge sharing and integration is the necessity to agree upon a shared terminology for ontologies. As there are many ontologies that are not formally defined, some projects in the area focus on formalizing informal ontologies. A methodology for formalizing already developed ontologies is described by Grüninger in [Grüninger95]. Experiences converting an informal ontology to Ontolingua formalism are described in [Uschol96]. Ontology libraries are also described as an important aid for ontology construction. New ontologies can be built from library ontologies, as a specialization of a generic ontology or as an aggregation of more than one ontology (each generic ontology being a lattice in the generated ontology).

2.2.4 Ontologies for modeling KBSs

The most complex task of building KBSs is building its knowledge base. Reasoning engines are available off-the-shelf in many software tools. But there are no off-the-shelf knowledge bases. Ontologies are also used for modeling knowledge-based applications (such as expert systems and distributed multi-agent applications) and business processes.

Some important work has been done on defining methodologies for the construction of ontologies for representing expert systems (ESs). ESs are good examples of systems that require detailed real knowledge in given domains. The organization of this knowledge is tailored to support the particular inferences the system needs to draw. ESs usually lack formal functional specifications. The increasing complexity in domain knowledge and functionality in ESs favors the use of ontologies in representing both the application specifications and the domain knowledge. Examples of methodologies for ES modeling are: Task-Based Specification Methodology (TBSM) ([Yen93]) and the METHONTOLOGY methodology, mention above. In TBSM, conceptual information is composed of a model specification, including a domain model and a state model, and a process specification, including functional and behavioural issues. Similar distinctions are proposed by the METHONTOLOGY methodology, in which two phases are distinguished in the conceptualization design: analysis to build the domain model, and synthesis to build the static and the dynamic model. Examples of projects using ontologies for modeling KBSs are the KACTUS project (described in [Schreiber95]), Comet and Cosmos projects (both described in [Mark95]). KACTUS is an ESPRIT project on modeling the knowledge of complex technical systems for multiple use. It has been applied to various domains: electrical networks, off-shore oil production and ship design and assessment. The Comet project supports the design of software systems and the Cosmos project supports engineering negotiation.

2.2.5 Ontologies for formalizing domain knowledge

The main purpose of many ontologies is the organization of knowledge in specific domains. The EngMath ontology ([Gruber94]), a mathematical ontology, the PhysSys ([Borst96]), an ontology modeling, simulating and Designing Physical Systems and the CHEMICALS ontology ([Fernández96]), an ontology for chemical substances, are examples of ontologies designed to model domain knowledge. The CHEMICALS ontology was designed following the METHONTOLOGY methodology mentioned above.

2.2.6 Ontologies to formalize the enterprise environment

In the business environment, proper knowledge representation becomes more important as complexity increases. Business process re-engineering requires an integrated model of the enterprise and its processes, organizations and objectives. Examples of this importance are the PIF, mentioned above, for sharing ontologies in business applications, mentioned above, the Enterprise Project ([Stader96]) and the Toronto Virtual Enterprise (TOVE) project. The PIF ontology was designed to exchange process descriptions between business process modeling and a tool repository. The basic criterion followed in PIF is that generality is preferred over efficiency; only the minimal set of classes is used and is expanded as needed. The Enterprise Project is a major project using an ontology to model business environment. The Enterprise Ontology was developed to archive integration for a variety of enterprise tools used in the project. The ontology is a set of terms frequently used in enterprises, and it focuses on the following areas: organization, strategy, activities, processes and marketing. All terms used in the project are committed to this ontology. The ontology is encoded in KIF (Knowledge Interchange Format). The TOVE project was developed by Grüninger and Fox. Its basic goal is to create enterprise models capable of answering queries by using what is explicitly represented and what can be deduced. It is a formal approach to ontology engineering.

2.3 THE USE OF ONTOLOGIES IN NATURAL LANGUAGE SYSTEMS

Some systems that currently deal with NLP already adopt some type of ontology for their more abstract levels of information. Complex NLP systems need to represent knowledge of the world, both general common-sense (in general-purpose systems) and domain specialized (in specialized systems). Most of these systems incorporate linguistic knowledge into their world knowledge representations. Bateman studies the use of ontologies in the main NLP systems in [Bateman91]. He distinguishes three different types

of ontologies representing three different approaches to organizing world knowledge in NLP systems: conceptual ontologies, mixed ontologies and interface ontologies.

Conceptual ontologies

Conceptual ontologies are abstract organizations of essentially non-linguistic world knowledge. Some NLP systems use language-neutral ontologies to organize concepts in the domain world. Most AI designed ontologies (e.g., CYC) belong to this group.

Mixed ontologies

Mixed ontologies are abstract semantic-conceptual representations of world knowledge used as semantics for grammar and lexicon. Lexical entries directly contain categories of the ontology, which are categories of real-world knowledge. Important examples of systems using this type of ontology are: the LILOG natural language understanding project, the ACE system ([Jacobs87]), a framework for language generation in interfaces for KBSs, and Mikrokosmos ([Malesh95]), a machine translation system.

In the ACE system, the main principle followed in knowledge organization is the encoding of metaphorical relationships and other associations among concepts that capture certain generalizations about language use. In this system, linguistic structures are directly associated with conceptual categories in which world knowledge is represented.

In the Mikrokosmos system, the ontology is used in language interpretation and generation. The meaning of natural text is represented in a language-neutral interlingual format as instantiated elements of the ontology. The lexical items are defined in terms of their mappings onto ontological concepts.

Interface ontologies

Interface ontologies are abstract organizations that act as interfaces between world knowledge and grammar and lexicon. The purpose of this approach is to establish a linguistically motivated organization of objects and relations in which application specifications could be represented. Relations between categories and linguistic distinctions can be more or less direct. The GUM and Accord are examples of systems using this approach. Systems using an interface ontology may also need a conceptual ontology when applied to complex domains and/or applications. This is the case of the system ONTOGENERATION, described in [Aguado98]. ONTOGENERATION is a Spanish information retrieval system pertaining to the chemical domain. It uses an

ontology for chemicals (called CHEMICALS) to represent this domain and the GUM ontology to interface domain knowledge and linguistic resources.

The approach used in mixed ontologies, distinguishing only the syntax level and the conceptual level, is supported by many linguistic works. One relevant example of this work is that of Jackendoff, defining the grammatical constraint ([Jackendoff83]) and the X-theory ([Jackendoff77]), in which the linguistic categories proposed are closely related to semantic-conceptual categories.

The approach used in interface ontologies, stratification between lexico-grammatical information, semantic information and conceptual information, is also supported by some linguistic theories, such as the systemic-functional theory, defined by Halliday ([Halliday85]) and used in the GUM.

According to Bateman, language is the best criterion to follow in ontology design, although since many linguistic details are domain-dependent it is not easy to achieve a domain-independent knowledge organization where specific linguistic distinctions are considered. Different solutions to this problem are presented in mixed and interface ontologies. In mixed ontologies, of the general conceptual classes proposed, few maintain contact with details of linguistic realization (e.g., action, state, object, place, etc.). There is no such methodology for expanding these general classes in a particular domain representing specific linguistic distinctions. Conceptually and linguistically motivated sorts are usually mixed, and the resulting ontologies are loose. In interface ontologies, classes are usually not so abstract and are directly related to linguistic distinctions. The dilemma is that, while abstract classes are necessary for improving the functionality of the system, classes closely related to linguistic distinctions are the best suited for NLP systems dealing with a wide variety of linguistic phenomena. The dilemma can be summarized as general versus domain specific and, particularizing it to conceptual knowledge representation, conceptual versus lexical ontologies.

Those ontologies that are particularly close to language are used to represent the lexicon directly. The most relevant projects working conceptual-based versus lexical-based representations are WORDNET ([WORDNET98]) and EDR ([Matsukawa91]). WORDNET is an English lexical/conceptual ontology, in which lexical objects are organized semantically. In WORDNET, the lexical unit is the synset, a set of words

assumed to be quasi-synonyms[1] related by means of conceptual-semantic and lexical relations, such as relations of hyponym (superclass) and metonymy (part-whole). EUROWORDNET (EUROWORDNET98]) can be seen as a multilingual extension of WORDNET. The aim of the project is to build wordnets for several European languages linked by means of semantic relations. EDR is a Japanese project to construct a conceptual dictionary for English and Japanese. A promising methodology in the development of these ontologies closed to language is that based on automatically extracting syntactic and semantic information from large parsed corpora. Classifications could be built automatically or semi-automatically by using this methodology.

An important work in the area of ontologies for natural language processing is that of the ANSI group on Ontology Standards and that of the EAGLES Lexicon/Semantics Working Group for creating a consensus on the design of ontologies for NLP.

[1]Used as synonyms in some context

CHAPTER 3

THE FUNCTIONALITY AND ARCHITECTURE OF GISE

As mentioned in the introduction, the aim of this thesis is to study how the NL communication between users and KBSs can be improved. This thesis proposes a new approach to improving the development and functionality of NLIs for KBSs. The approach is based on the study of the most appropriate representation of the different types of knowledge involved in communication with KBSs. GISE (*Generador de Interfaces a Sistemas Expertos*), a system using an ontology to automatically generate the most appropriate NLI for each application is proposed. The central issues of the system design are discussed in this chapter.

This chapter has been organized into three sections. The first section describes the functionality of the system proposed. Its architecture is detailed in the second section. The third section describes the process of generating an NLI for a specific application.

3.1 THE FUNCTIONALITY OF GISE

3.1.1 Main goals in the functionality of NLIs to KBSs

As pointed out in the previous chapter, most improvements in human-computer technology have been achieved by adapting the interfaces to the performance of the applications and users. Interfaces have been adapted to the different types of users by incorporating techniques guiding the user with respect to system capabilities, such as menu-systems, helping tools and explanations. These techniques help in preventing mistakes and misunderstandings during the communication process, improving its friendliness and effectiveness.

Several approaches have been followed when adapting NLIs to applications. Most of these approaches consist of adapting the linguistic resources to the communication tasks required for each application. These approaches have been successfully followed for developing interfaces to systems using restricted linguistic resources, such as database consulting systems.

Application-restricted resources for KBSs are more difficult to obtain because they require larger linguistic resources to express all the tasks involved in communication. Besides, no schema or description is available for KBSs. This means that there is no easy way to obtain information about propositions. Propositions may have arbitrary meanings. Another major difference in the functions of KBSs and DBSM is in that DBMS is only expected to supply user requests. A KBS is a problem solver. Users consult it about an issue and it must gather information in order to advise them. For these reasons, the cost of developing, maintaining and tailoring the linguistic resources adapted to the functionality of the specific KBS is high. Different profiles of expertise are needed (linguistic, expert on the domain, expert on the application, etc.).

One of the main goals of this work is to study how general resources can be adapted to each specific KBS automatically or semi-automatically. For this purpose, a system is

proposed that provides a framework to represent the specification of the domain and the functionality of the application in a conceptual ontology supporting NL interaction.

3.1.2 Main issues of the functionality of GISE

GISE was designed to automatically generate application-restricted NLIs from application specifications. The central issue in the system design is the study of the appropriate representation of the different types of knowledge involved in the communication. These different types of knowledge have been represented in separate reusable bases.

The general knowledge needed to model the KBSs as well as the knowledge representing the tasks involved in their communication with the users is defined in the CO. It is to be noted that only entities and tasks involved in the communicative process need to be modeled. It is not necessary to build a model of the whole KBS. In the case of the whole KBS being organized around an ontology, only those concepts appearing in the communication would have to be adapted to the CO.

The tasks of communication mainly consist of operations consulting and describing particular knowledge on the application. These tasks are represented as operations on the concepts modeling the application in the CO. Examples of basic operations are those creating or removing instances of concepts modeling the application, filling their slots, connecting instances, and querying their properties.

The linguistic knowledge needed to express all possible communication tasks is represented in a linguistic ontology (LO).

Finally, there is a set of general rules controlling the process of adapting the general resources to those required for a specific application. This process consists of adapting the general communication tasks to the application knowledge represented in the CO and then the linguistic knowledge represented in the LO, in order to express these application tasks. This process is described in Figure 3.1.

As shown in the figure, obtaining the specific grammar and lexicon for an application is performed by a set of control rules using conceptual (both general and domain/application specific) and linguistic information as knowledge sources. The cost of developing NLIs is reduced because the knowledge sources necessary for the communication are reused.

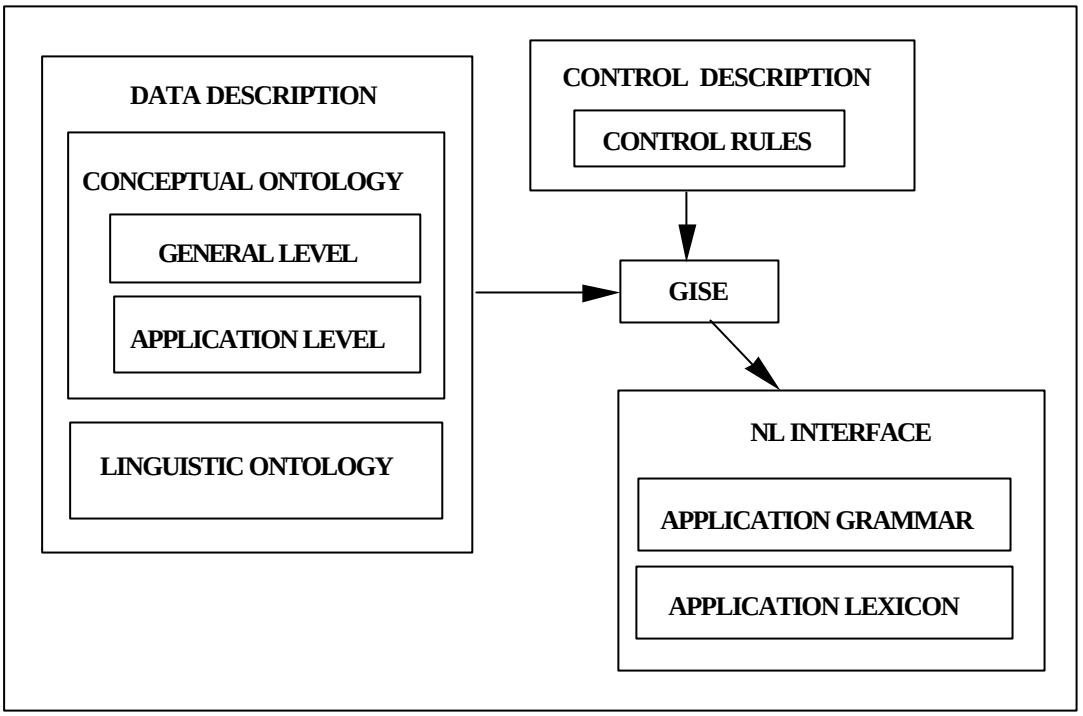

Figure 3.1: The process of generating an application-restricted NLI

Building an interface is an incremental process of acquiring the knowledge sources involved. First, the domain and functionality of the application must be represented in the CO. The possible lexical realizations of the concepts involved in the communication must be represented in the set of application terms. Once all these knowledge sources have been updated, the control rules adapt the general linguistic knowledge in the LO to cover the specific communication tasks for an application.

A basic set of control rules was defined to generate interfaces supporting the linguistic coverage needed in the communication with different types of KBSs. However, this set of rules can be extended and modified to enrich the language coverage.

The resulting interface and the CO are integrated into the KBSs. Several forms of integration are allowed. The NLIs generated could also be integrated together with other modes of communication. Chapter 7 describes more precisely the interfaces generatedby GISE, and how they can be integrated into the application.

3.1.3 The functionality of the interfaces generated by GISE

The aim of this thesis is to improve the NL communication process with any KBS. However, in order to reuse the different types of knowledge involved in the development of an interface, the system designed was restricted to a class of KBSs: the ESs performing heuristic classification. Most currently existing industrial ESs belong to this category. This limitation involves a reduction of the type of communicative tasks to be covered by the interface, but does not affect the architecture and functionality of the system. The system design can easily be extended to cover other communicative tasks following the same methodology.

As the basic task in the ESs considered here is classification, the knowledge in these ESs and the information provided by the user (directly or as a result of a query) is used to classify a case into one of a set of defined classes. In these systems, communication with users usually relies on a taxonomy of objects having different properties and relationships. System interventions consist of describing parts of this taxonomy and querying the user about unknown or poorly established facts. User interventions are contributions enriching the taxonomy and queries for a partial description of it.

The functionality of GISE is to obtain user-interfaces, that is, interfaces supporting the communication between the users and the KBS during the exploitation phase, when the final users execute the application. In this phase, the KBS acts as a consultant for solving problems in its domain, such as asking questions about a problem at hand, answering queries and displaying results. However, the functionality of the interfaces generated could easily be extended to the acquisition phase. The design proposed could be adapted for generating expert-interfaces, supporting the communication with the expert when the knowledge acquisition is undertaken, in the building phase.

The interfaces generated by GISE use expressive and precise language adapted dynamically to the evolution of the communicative process and the application tasks. These interfaces support the language required to express, in a natural way, operations consulting and describing the application knowledge represented in the CO.

Users can introduce the sentences either by typing them in or by using a menu-system guiding the users in introducing the NL options that are acceptable to the system at each stage of the communication. The incorporation of a menu-system guiding the user in

building NL sentences solves the problem of lack of user knowledge of the sublanguage acceptable to the system, thereby improving user satisfaction. By restricting the language to the application and by incorporating a menu mode guiding the users, mistakes and misunderstandings in the communication are avoided. The task is entirely successful because only the expressions of the tasks that the application can perform are supported by the grammar. The cost of dialogue is low because users are guided in introducing each word of the sentence.

The friendliness of the generated interfaces has been improved with the incorporation of helping tools, such as an option giving information about the interface lexicon and another option that allows the user to correct previously introduced NL options.

The system design could also be adapted to performing other tasks requiring the definition of general relations between global linguistic knowledge and specific application knowledge, such as generating explanations, descriptions and summaries.

3.1.4 Applications of GISE

GISE was initially applied to an ES in law, SIREDOJ (Intelligent System for Legal Information Retrieval). SIREDOJ simulates a specialist in duty inquiries in the field of building contracts. It simulates different tasks: it establishes the type of contract, the subjects involved, knowledge of content, etc. The user introduces all the information about a case and the application then displays the juridical conclusions with the legal justifications, just as a specialist would do.

The SIREDOJ knowledge base is restricted to a set of types of contracts. The representation of the application on the GISE CO includes the description of different concepts involved in building contracts. The most relevant concepts represented are: **CONTRACT**, **BUILDING_CONTRACT, CONTRACT_PARTS**, **CONTRACT INFORMATION**, **REQUIREMENT** and the three different types of duties in the building contracts: **BUILDING_REQUIREMENT**, **DELIVERY_REQUIREMENT** and **PAYMENT_REQUIREMENT**. The description of these concepts is provided by a set of attributes. These attributes are also represented as CO entities and are described by a set of descriptors or facets.

In order to enrich the different knowledge sources involved in the process of generating NLIs, GISE was applied to a different type of application: consulting system in the railways domain. The concepts involved in this type of application are simpler than those in SIREDOJ. The number of concepts and attributes representing this application in the CO is, however, larger. The main concepts represented are: **TRAIN** (and its subclasses representing the different types of trains), **TRAIN_STATION**, **LINE**, **SEAT_FARE**, **PRICE** and **DISCOUNT**. These concepts were described by a set of attributes. For example, the attributes describing the concept **TRAIN** give information about departure and destination stations, schedule, seats, etc.

The grammars generated by GISE for these two applications are described in Chapter 8.

All the examples illustrating particular decisions on the system design in the following chapters are obtained from these two applications. The concepts identifiers are presented in bold upper case letters. The identifiers of the attributes describing these concepts are presented in bold lower case letters. Although the concepts and attribute identifiers are in English, its superficial realization is in Spanish.

3.2 THE ARCHITECTURE OF GISE

The appropriate organization of the different types of knowledge involved in the NL communication with KBSs is crucial in order to favor its efficient processing and reuse. As Bateman discusses in his survey on the ontologies for NLP in [Bateman91], representing conceptual and linguistic knowledge separately is especially necessary to capture generalizations between conceptual information and its expression. Bateman proposes a methodology for constructing ontologies for NLP. This proposal consists of representing world knowledge in a CO, and semantic-syntactic knowledge in a separate ontology, acting as an interface between the lexico-grammatical resources and the CO.

Organizing the information in multiple strata or levels is justified for theoretical and practical reasons. This organization supports theoretical positions that assume a higher degree of stratification of the linguistic system between lexico-grammatical information,

semantic information and conceptual information. It provides a more flexible and efficient representation of the relationships between conceptual categories and the linguistic categories supporting their expression. Besides, this organization also captures general relations between knowledge represented in different levels. The relationships between the levels are strong, the design of the interface ontology is guided by the lexico-grammar constructions and the classes (or sorts) in the CO.

A similar organization has been adopted in GISE to represent all knowledge involved in NL communication with KBSs. This knowledge consists of the application knowledge appearing in communication, communication tasks, the linguistic knowledge supporting the expression of such tasks, and the general relations between the different types of knowledge involved. This knowledge has been organized in separate data structures: the CO, a semantic-syntactic taxonomy of the attributes describing the concepts, the LO and the control rules. These knowledge bases are represented in Figure 3.2.

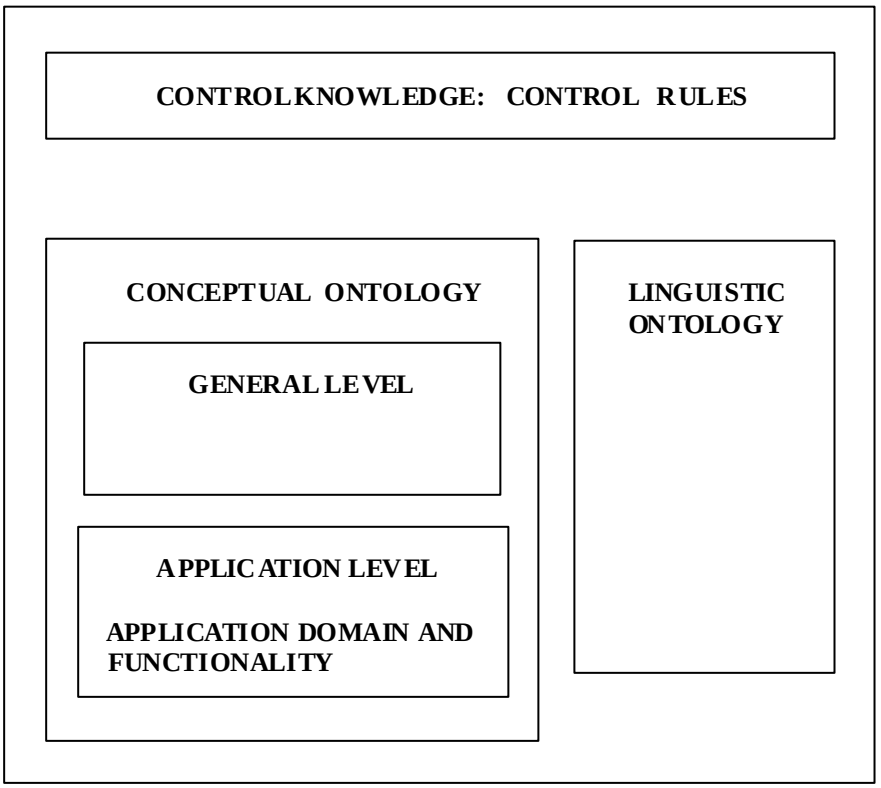

Figure 3.2: The knowledge bases involved in the communication process

3.2.1 Conceptual Ontology

The CO provides a conceptual framework allowing the automatic construction of NLIs that, once included in the overall application, could be responsible for all the communicative tasks between user and application.

All the knowledge in the CO is organized at two levels: the **general level** and the **application level**. The general level describes the conceptual knowledge common to all applications. The application level describes a specific application as well as its communication tasks.

The basic entities distinguished in the CO are concepts, attributes and operations. Concepts are described by a set of descriptors or attributes. Attributes describing concepts and operations concerning these concepts are described by other attributes called *facets*[2]. There is a set of obligatory structural properties that must be used when describing application concepts and attributes. These properties are also represented as facets in the objects description. This information is used by the system when generating the most appropriate interface for a specific application. An example of these structural properties is the facet **interface_entity**, which must be present in all application concepts and attributes. Its value, **yes** or **no**, indicates whether or not the concept or attribute is used during communication.

Concepts, attributes and operations are organized in three separate but related taxonomies. The taxonomy classifying the attributes is semantic-syntactic motivated. The purpose of the taxonomy of attributes proposed is to organize the meanings that need to be expressed. These meanings basically consist of operations regarding the attributes of the concepts in the CO. This taxonomy acts as an interface between the conceptual knowledge

[2] Although in literature the term *facet* is used for attributes describing conceptual attributes, in this thesis the use of this term is not restricted to that type of attributes. The term *attribute* is used for referring to conceptual attributes giving conceptual information. The term *facet* is used for all other attributes: the attributes describing structural properties of concepts as well as those describing conceptual attributes, operations and LO objects.

representing the application in the CO and the linguistic structures required for its expression in the LO.

3.2.2 Linguistic Ontology

Linguistic knowledge is represented in the LO. The LO represents the linguistic structures required to express all possible communication tasks. The LO was organized following the basic principles of the Nigel grammar (described in [Nigel88]). The linguistic structures constituting the general grammar are described as classes in the LO. Representing the general grammatical structures in an LO facilitates the process of obtaining the particular grammatical structures adapted to each application. To improve this process, a domain level with the linguistic structures required to express the communication tasks was also included.

The words common to all applications, such as auxiliary verbs, articles and prepositions are incorporated into the LO. For each application, the realization of the application terms used in the communication must be described. The syntactic information associated with each possible lexical realization must also be provided. This information consists of the category and the syntactic features (gender, number, tense, etc).

3.2.3 The control rules

The relationships between conceptual and linguistic knowledge determining the linguistic resources most appropriate to a specific application are represented by a set of general control rules. These rules control the process of relating the general linguistic structures in the LO to the application requirements. This process is carried out in two steps. First, the CO operations representing the general communication tasks are adapted to the application concepts. Then, the application-specific communication tasks are related to the linguistic structures in the LO supporting their expression.

The description of all these knowledge bases is detailed in the following chapters. Application knowledge, communication tasks in addition to the semantic-syntactic taxonomy of the conceptual attributes represented in the CO are described in Chapter 4. The LO is detailed in Chapter 5. The formalism of the control rules as well as the basic set of rules defined for different types of KBSs are described in Chapter 6.

3.3 THE PROCESS OF GENERATING THE INTERFACE

The use of large coverage linguistic resources for specific applications has proved unsatisfactory. The space and run-time requirements are too large. Application-restricted grammars improve efficiency in language processing but are expensive to develop and difficult to reuse, especially for complex systems like KBSs. Several strategies have been attempted that are aimed at reducing the cost of creating application-dependent grammars. This adaptation process can be performed by generating a specific application subgrammar or by providing the grammar of a dynamic mechanism to restrict the grammatical options at run-time. The cost of generating application-tuned subgrammars is reduced when undertaken automatically, as is the case in some recent work, such as that described in [Henschel97]. In most systems, application-restricted grammars are obtained from linguistic resources restricted to supporting communication with one type of application (or more). Examples of such work are described in [Caldwell94] and in [Busemann98].

Using dynamic mechanisms is also an efficient way of restricting grammar, as established by the dynamic rule pruning mechanism described in [Dowing88]. This pruning mechanism is based on information available at run-time and is used to reduce the grammatical options that must be considered.

In this proposal, the LO is adapted automatically to the communication tasks required for an application in order to obtain application-restricted dynamic grammars and lexicons. Although the LO describes general linguistic structures, it also includes a domain-motivated level restricting general structures to those required in communication with different types of applications. This level improves the efficiency of the adapting process.

The current section describes general aspects involved in the generation of the most appropriate grammar and lexicon for each application. The general process of adapting general linguistic knowledge to the communication tasks required for a KBS consists of three steps. These three steps are:

Step 1. Providing the application with domain-specific knowledge.

Step 2. Adapting the general communication tasks to cover application knowledge.

Step 3. Adapting general linguistic knowledge to express the communication tasks required for an application.

The system designer must perform the first step. A set of general control rules is in charge of the two other steps. These three steps are described below.

3.3.1 Providing domain-specific knowledge

The first step in generating an application-restricted interface consists of providing the system of the domain and functional knowledge of a specific KBS. In order to generalize this process to different applications, a general organization of the knowledge needed to represent the application is defined in the general level of the CO, as described in the previous section. The application's domain-specific knowledge must be represented in the CO application level. Additionally, the application's functionality and domain must be described following the basic commitments in the CO design.

Because the general knowledge needed in communication with KBSs has been already represented in the CO, the process of building the domain ontology for a specific application is reduced to organizing application knowledge that appears in communication as an instantiation of the CO general level. This process can be described as follows:

Step 1.1. Providing the description of the concepts of the domain. Each concept must be described by an identifier, a primitive relation (**isa** or **instance**) relating it to the taxonomy of concepts in the CO, a set of attribute-values and the obligatory structural properties. Optionally, preconditions on the concepts can be incorporated into its description.

Step1.2. Providing the description of the attributes describing the concepts of the domain. The identifier, the domain, the range, the class they belong to and the structural properties must describe all attributes appearing in the definition of concepts. If the range of the attribute is a closed set of values, this set must be included in the attribute definition. Closed values can be represented either as a simple set of individual terms or as **menus**, that is, lists of values that are displayed in the screen during the communication.

All the attributes describing the concepts have to be classified according to the taxonomy of attributes containing semantic-syntactic classes. New subclasses of attributes adding new semantic-syntactic information can be incorporated into the taxonomy of attributes. In order to add a new class to the basic ones already established, it is only necessary to provide the name, its upper classes and other attribute facets (if necessary) in the CO. If new linguistic classes supporting the realization of the attributes in this new class are required, they have to be incorporated into the LO.

Step1.3. Providing the lexical realization of the application concepts, attributes and values and their linguistic description. A link to its lexical realization represented in the set of terms is provided only for objects defined as interface entities. Each application term can be linked to one or more lexical realizations. For each term realization, the syntactic category and its the syntactic features (in current design, gender, number and tense) must be provided. The realization of a term is not reduced to single word; a nominal group and a multi-word phrase can also represent it. Optionally, synonyms and abbreviations (such as acronyms) can also be incorporated to provide a friendlier expression. More than one term can be provided to represent different realizations of the same concept or attribute.

As the approach proposed is independent of the implementation language, existing tools and environments helping to build ontologies in different languages can also be used to implement or translate the domain-restricted ontology.

3.3.2 Tuning to task-specific communication

As mentioned before, the functionality of the NLIs has been improved by restricting the linguistic resources to those needed in the communication between the user and the KBS. For this reason, not only must the application domain knowledge be specified, but so too must the communication tasks over this domain. The communication tasks basically depend on the type of application.

The linguistic structures required to express these operations have to be represented in the LO. For this reason, the communication tasks considered in GISE were restricted to those appearing in a specific type of KBSs, the ESs relying on heuristic classification. The communication tasks in these ESs basically consist of providing and consulting information about particular cases in a specific domain. These communication tasks have been represented in the CO as operations that create, describe and consult CO concepts. However, the incorporation of new operations to cover the communication tasks appearing in other types of application will not represent any change in the process of obtaining the most appropriate interface for an application.

For each application, once the domain-specific knowledge is provided, the general communication tasks are adapted to that which is necessary for a specific application. This process consists of creating instances of the CO operations that are applied to the concepts modeling the application.

This process is performed automatically. It only requires the intervention of a system specialist when new operation needs to be incorporated into the general taxonomy of operations in the CO. The specific details of this automatic process-tuning to task-specific communication are given in Chapter 6.

3.3.3 Tuning to specific NL coverage

The process of adapting general linguistic knowledge to a specific application consists of adapting the knowledge in the LO to cover the specific communication tasks required for an application. In order to generalize this process for different KBSs, general relations between conceptual and linguistic knowledge must be established. The study of the general relations between the conceptual knowledge needed in the communication tasks and its linguistic expression has been one of the most important objectives to this work. In order to capture the specific syntactic details with regard to the realization of specific concepts, an

appropriate representation of the conceptual and linguistic knowledge involved is required. The CO and the LO were designed to support these general, application-independent relations. The syntactic-semantic taxonomy of attributes, together with the control rules relating the communication tasks for an application to its expression, provides the knowledge necessary to automatically obtain the application-restricted grammar and lexicon.

CHAPTER 4

CONCEPTUAL ONTOLOGY

4.1 INTRODUCTION

CO provides a conceptual framework for representing the conceptual knowledge involved in the communication with KBSs. CO allows the automatic construction of application-restricted NLIs.

Not all the concepts involved in the performance of the KBSs need to be modeled and represented in the CO; this is required only for the concepts relevant to any communicative task. This knowledge could be obtained from any internal explicit representation of domain and behavior, but most KBSs lack this explicit representation. If the KBSs already use an

ontology for its internal representation, this ontology could easily be adapted to the CO proposed in this design.

Representing the application knowledge involved in the communication in a CO allows the reusability of the interface-generating process in different domains/applications. It also allows a separate development and a robust integration of communicative and application-specific tasks. In order to generalize the process of obtaining application-restricted NLIs to KBSs, not only the basic concepts and relations common to all KBSs need to be represented but also the communication tasks needed in the exploitation phase, when the user informs and asks about particular knowledge.

All conceptual knowledge needed in the communication process (general and application-dependent knowledge) is described and structured in the CO. This knowledge is organized following the Task-Based Specification Methodology (TBSM) for system modeling described in [Yen93]. In such a methodology, the conceptual information is organized in a Model Specification, including a Domain Model and a State Model, and a Process Specification that includes functional and behavioral issues. All these aspects are covered in the CO.

The CO represents the general knowledge common to all applications. This general knowledge is instantiated for each application. The general knowledge is the skeleton for anchoring the application knowledge for each KBS.

4.2 THE GENERAL COMMITMENTS FOLLOWED IN THE CO DESIGN

The main goals in this design are to build a reusable, extendible, comprehensible ontology that is easy to integrate. The general commitments followed when representing the conceptual knowledge in the CO are described below.

4.2.1 Basic relations

The ontology is designed in an object-like fashion, where basic objects are concepts, attributes describing concepts and operations on these concepts. There are two basic relationships classifying these objects: subsumption (**isa**) and instantiation (**instance**). The **isa** relation classifies a general object (a **class**) as a member of a more general object. The **instance** relation classifies a concrete object (an **instance)** as a particular instantiation of a general object.

4.2.2 Inheritance

The system allows orthogonal multiple inheritance; more than one dimension can be considered when classifying objects. Information acquisition is carried out monotonically. All attributes describing a class are inherited in the subclass. In the subclass, the range of these attributes can be restricted to add new information. The information added in a class (the attributes and their range) must be consistent with the information inherited. Exceptions are allowed for specific cases, for example, the range of an attribute may not belong to the range of the attribute in the upper class.

4.2.3 The organization

The basic entities distinguished in the CO are concepts, attributes and operations.

Concepts

Concepts include both physical and abstract entities. Their conceptual properties are described by a set of descriptors called attributes. These attributes are also described as CO entities. The structural properties of the concepts are described by a set of predefined

descriptors called *facets* (as has been mentioned in previous chapter). For each application, the concepts and their attributes appearing in the communication are represented in the CO. There are, however, concepts and attributes not expressed during the communication that also need to be represented in the CO for reasons of consistency. The facet **interface_entity** attached to all CO entities indicates whether they are expressed (the value of this facet is **yes**) or not (its value is **no**). Concepts and attributes appearing in the communication must be linked to their linguistic realization by the facet **lex**.

Concepts in the application level either represent an individual or a collective. In this design, collections are represented by sets, although other types of collections could also be considered. A conceptual set is a particular type of concept represented by a set of concepts belonging to the same class. The facets **collective** and **has_member** must be attached to those concepts description. The members of the set must be also described in the application level. The facet **member_of** relates the members to their set. If all the members in a set have the same conceptual and structural properties, then there is only one concept representing the prototype of the members. Their description must include the facet **prototype**. If the members are not described exactly by the same attributes and facets, then each member in the set is represented by a different concept.

Optionally, preconditions on the concepts can be incorporated into their description.

Attributes

Attributes are conceptual properties attached to concepts. Attributes are also represented as objects described by other attributes, called facets. The obligatory facets for all attributes are: **interface_entity**, **domain**, **range** and **value**.

The facet **domain** indicates the concept or concepts described by the attribute. The range of the attributes can be open or closed. A closed range consists of a finite number of possible values. The description of attributes having a close range must include the set of all possible values. There are three different types of closed sets of values: **yes/no**, **lists** and **menus**. The difference between these two types is that menus are displayed in the screen at run-time to show the user all the possible values of an attribute. Menus are especially useful for guiding the user on specific domain information. For example, in the CO describing the ES in law SIREDOJ, several menus were defined: one for all types of

contracts considered, one for all possible types of payment, one for all possible reasons for not building, etc.

There are also attributes which values cannot be restricted to a predefined set, their ranges is an open set. These values can be instances of concepts, quantities, proper names and text. If the value of the concept must be a conceptual instance, all existing instances of that concept are displayed on-screen and the user must select one (or more) of them. In the case of the range of the attribute being a quantity or a proper name, a **dynamic function** asks the user to introduce the value during the communication. Attribute values can also be text, representing descriptions or comments.

An attribute describing a concept can have more than one value. The structural property **multiple** must be included in the description of those attributes having more than one value.

Operations

All allowed operations over the concepts are also represented as objects in the CO. These operations represent all required communication tasks. The information describing an operation is represented by a set of specific facets. These facets represent information about the operation parameters and the operation preconditions.

Concepts, attributes and operations are organized in three separate but related taxonomies. A similar conceptual organization in three separated taxonomies is followed in other well-known conceptual ontologies for different purposes, including NLP. The organization of these three basic entities in separate taxonomies is described in the following section.

All the knowledge in the CO is organized into two levels: the **general level** and the **application level**. The general level describes the conceptual knowledge common to all applications. This knowledge consists of the definition of the basic entities needed in all applications.

A fragment of the general level describing the most general objects is shown in Figure 4.1.

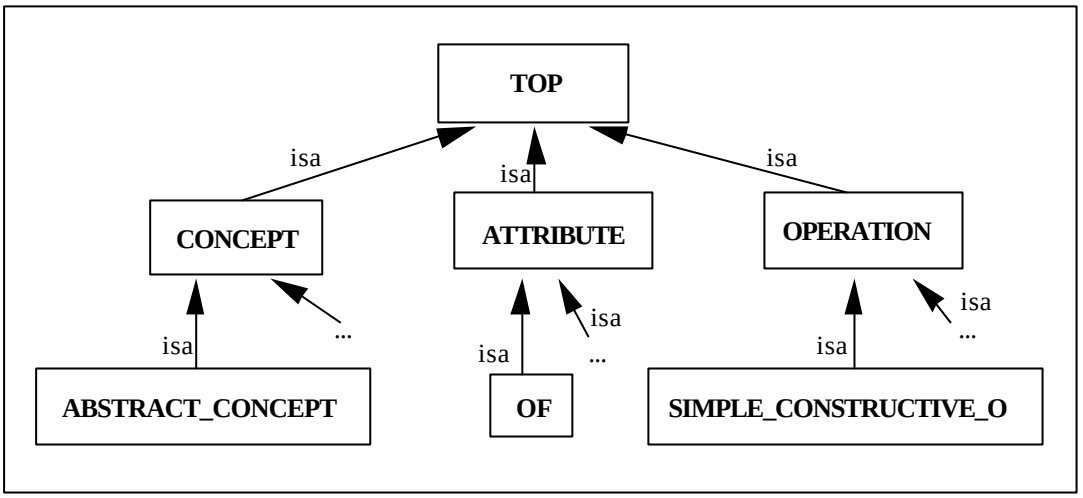

Figure 4.1: A fragment of the CO general level

The application level represents the application concepts and attributes involved in the communication as well as all possible operations on those concepts. The domain and functional knowledge for a specific application is represented on the basis of the knowledge described in the general level. Two sublevels were distinguished on the application level: the **description level** and the **case level**.

The description level represents all concepts, attributes and operations describing the application knowledge. The concepts and attributes must be defined manually following the commitments described above. All communication tasks needed for a specific application are obtained automatically by adapting the communication tasks in the general level to the application concepts and attributes. This level cannot be modified during the communication.

The case level describes all information about the specific cases examined by the KBS during the exploitation phase. All knowledge represented at the case level is an instantiation of the description level. While the description level is static, the case level is dynamic. The case level is built incrementally during the communication process.

4.3 THE TAXONOMY OF CONCEPTS

The basic concepts common to all applications are described in the general level. GISE does not provide an exhaustive classification of concepts. The system design could include a general conceptual ontology, such as the CYC Top Ontology, available in [CYC98] and the EuroWordNet Top Ontology, described in [EuroWordnet98]. A different possible strategy would be to include a complete description of the domain and functional issues common to all KBSs in the CO. However, because GISE was designed for communication with different applications and domains, the criteria followed in the CO design was to represent the minimum knowledge common to applications.

Each concept is described by an identifier, a primitive relation (**isa** or **instance**) relating it to the taxonomy of concepts, a set of attributes and a set of facets. As described in the previous section, attributes represent conceptual properties and their description must be incorporated into the CO taxonomy of attributes. Facets represent structural properties. The only obligatory facet for all concepts is **interface_entity**, indicating if they are expressed during communication or not. If the value of this facet is yes, the concept description must include a pointer, **lex**, to all its corresponding realizations.

Figure 4.2 shows an example of a class of concept, the class **TRAIN_STATION**. This concept, as any other concept in the ontology, is described by a set of attributes, facets and their values. The primitive relation **isa** indicates that **TRAIN_STATION** is a subclass of the conceptual class **PHYSICAL_CONCEPT**. The value **yes** of the facet **interface_entity** indicates that this concept appears in communication. **lex** is a pointer to its linguistic realizations, described in the set of application terms. The rest of attributes describing the class give the conceptual information necessary to describe particular instances of this class during interaction with the user. These attributes must be defined in the general taxonomy of attributes.

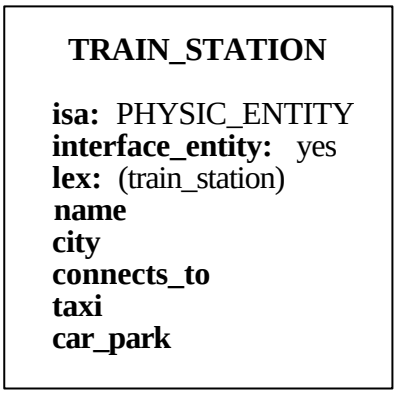

*Figure 4.2: The representation of the class **TRAIN_STATION***

The attribute **name** represents the station's name. Its value must be a proper name. The range of this attribute is open; a dynamic function called **name** will request the attribute value from the user during communication. The attribute **city** represents the name of the city in which the train station is located. Its value must also be a proper name. The function **name** will ask the user to introduce the city name at run-time.

The attribute **connects_to** represents the names of all the train-stations connected. Its value can be zero, one or more names representing instances of the class **TRAIN_STATION**. Its value is defined as an instance value. All the existing instances of the concept **TRAIN_STATION** will appear on-screen at run-time to guide the user to introduce the correct value. By showing all acceptable options on-screen at each state of the communication, the user is compelled to appropriately relate the new instance to the existing instances in the case ontology, thereby assuring its consistence.

The attribute **taxi** indicates if there is a taxi station close to the train station. The attribute **car_park** indicates if there is a station car park. The range of the two attributes is the closed set **yes/no**.

4.4 THE SYNTACTIC-SEMANTIC TAXONOMY OF ATTRIBUTES

As mentioned in the introduction, one of the most important problems to solve in the conceptual representations used in NLP is how application knowledge can be related to its linguistic realization in a general way.

In terms of Bateman's classification, there are three different types of ontologies in NLP: conceptual ontologies, representing only conceptual knowledge, mixed ontologies, where conceptual and linguistic knowledge is mixed, and interface ontologies, where the organization of knowledge is basically linguistic. According to this classification, the CO proposed is conceptually motivated because only conceptual knowledge is considered when defining the taxonomy of concepts. Linguistic knowledge is not considered. However, the basic classes of concepts in the CO can be related to general linguistic categories because there is a strong relationship between the organization of language and knowledge, as supported by many current linguistic theories.

The taxonomy of attributes acts as an interface between the conceptual knowledge representing the application, and the linguistic knowledge needed for its expression. It facilitates the generalizing of the process obtaining the linguistic structures required for the expression of the application specific conceptual knowledge. The linguistic realization of the attributes is very important in this design because the communication tasks supported basically consist of the expression of operations on the attributes of concepts representing a specific application. All conceptual attributes are classified in the taxonomy according to their linguistic behavior. The syntactic-semantic classification of attributes allows a variety of different linguistic coverage for each attribute class.

This approach has proved satisfactory because it avoids the problem of mixing ontologies, where conceptually motivated and linguistically motivated classes are mixed, which results in these ontologies being loose. In this proposal, the conceptual and linguistic knowledge is represented as completely separate and the syntactic-semantic taxonomy acts an intermediate level relating both. Although this classification was incorporated into the general level of the CO, where conceptual knowledge common to all applications is represented, it is completely independent of the conceptual knowledge.

The problem of relating attributes to the different grammatical structures for obtaining interfaces to software applications has been studied in several works. The most important of these is Perkins ([Perkins89]), categorizing the attributes into 16 types and assigning to each attribute type a question template, a declaration template and an uncertain template.

These types of attributes were obtained by analyzing the linguistic behavior of existing interfaces to KBSs. Only a restricted type of simple sentences was considered when establishing this classification.

The attribute taxonomy proposed in this thesis is the result of considering existing classifications of attributes of concepts, such as that by Perkins and the GUM ([Bateman90]), and by means of an empirical evaluation of the sentences used in different KBSs. In this taxonomy, linguistic behavior is considered when defining the basic attribute classes. All the attribute classes distinguished are necessary to reflect different surface realizations.

The linguistic information associated with each attribute class is represented in its facets. This information relates the attributes in the class to the LO structures representing the different forms of expressing the operations of consulting and filling the attributes. The facet **decsent**, links the attribute class to the to the linguistic structures required to express the filling operation and the facet **intsent** links it to those linguistic structures expressing the consulting operation. The information represented in these facets includes, as well, specific constituents of the LO structures associated with the attribute class, such as prepositions, interrogation pronouns and adverbs.

The basic attribute classes are:

- **WHO_DOES**
- **WHO_OBJECT**
- **WHAT_OBJECT**
- **IS**
- **HAS**
- **OF**
- **DOES**

These seven basic attribute classes are associated with grammatical roles: participants (**WHO_DOES**, **WHO_OBJECT**, **WHAT_OBJECT**), being (**IS**), possession (**HAS**), descriptions and relationships between two or more objects (**OF**) and related processes (**DOES**).

Subclasses are obtained from basic classes considering other information relevant for the linguistic realization of attributes. The **OF** class was subdivided into three classes:

- **OF_PERSON**

- **OF_OBJECT**
- **OF_DESCRIPTION**

The class **OF_PERSON** describes relations between the concept representing a person and one or more persons. The class **OF_OBJECT** represents relations between the concept and one or more objects. The class **OF_DESCRIPTION** represents qualities and circumstances related to the concept.

The class **OF_PERSON_SIM** was distinguished in the class **OF_PERSON**. This subclass represents relations between persons that are expressed by two different words: one for each direction of the relation. Examples of attributes belonging to this class are **father-son**, **teacher-student** and **boss-employee**.

The class **OF_DESCRIPTION** was subdivided into the classes:

- **OF_TIME**
- **OF_PLACE**
- **OF_MANNER**
- **OF_CAUSE**
- **OF_QUANTITY**
- **OF_NAME**
- **OF_TYPE**

These subclasses represent attributes describing time, place, manner, cause, quantity, name and type respectively. They have been further subclassified considering specific linguistic details in the expression of the attributes in the class, such as having an associated verb or preposition.

Each attribute class is associated with one or more general forms to express the basic operations of filling and consulting the attributes represented by the class. These forms correspond to linguistic structures represented in the LO, described in next chapter. For example, the filling of attributes in the class **OF** in Spanish is expressed using an attributive clause of the form:

<attribute_name> de <concept_name> es <attribute_value>
(*<attribute_name>of <concept_name> is <attribute_value>*)

where the concept name, the attribute name and the attribute value are represented by nominal groups.

The general form for consulting the value of the attributes in this class is:

¿<interrogative pronoun> es <attribute_name> de <concept_name>?
(*<interrogative pronoun> is <concept_name> <attribute_name>?*)

Depending on the subclass of the attribute, the interrogative pronoun would be *cuál* (*which*), *quién* (*who*) or *qué* (*what*).

Although the consulting and filling of attributes belonging to the subclasses of the class **OF** can be expressed following the general forms described above; there are patterns associated with each subclass to express these operations more naturally.

For example, the class **OF_QUANTITY** describes attributes referring to quantities. Attributes in this class always involve the use of a unit of measure. The interrogative adverb *cuánto/cuántos* (*how much/how many*) appearing in the interrogation clauses expressing consult operations on these attributes is also included in the description of the class. In Spanish, attributes expressing a quantity have an associated verb (which corresponds to an associated adjective in English).

Examples of attributes describing concepts and their classification in the taxonomy of attributes are given next. These attributes belong to the classes **OF_QUANTITY**, **OF_NAME**, **OF_PLACE** and **DOES**. The patterns associated with these classes are described below.

The general form to express the filling of the attributes in the class **OF_QUANTITY** in Spanish corresponds to simple transitive clauses of the form:

<concept_name> <associated_verb> <attribute_value>
(*<concept_name> is <attribute_value> <associated_adjective>*)

There are two different consulting forms for attributes in this class:

¿Cuántos <associated_unit><associated_verb> <concept_name>?
¿Cuánto <associated_verb> <concept_name>?
(*How <associated_adjective> is <concept_name>?*)

The pattern to express the filling of the attributes in the class **OF_NAME** is:

<concept_name> se llama <attribute_value>

(<concept>'s name is <value_name>)

The pattern to consult the attributes in this class is:

¿Cómo se llama <concept_name>?

(*What's <concept-name> name?*)

The expression of the operation to fill attributes in the class **OF_PLACE** is a clause of the form:

<concept_name> está en <attribute_name> <value>

(*<concept_name> is in <attribute_name> <value>*)

The operation to consult these attributes is a sentence of the form:

¿Dónde está <concept_name>?

(*Where is <concept_name>?*)

The more natural form of expressing the operation of filling the value of the attributes in the class **DOES** is an intransitive clause of the form:

<concept_name> <attribute_name> <attribute_value>

(The same form would be used in English)

The consulting of attributes in this class follow the form:

¿<attribute_name> <concept_name> <attribute_value>?

¿<concept_name> <attribute_name> <attribute_value>?

(*Does <concept_name> <attribute_name> <attribute_value>?)*

An example adapted to show how highly language-dependent information is represented in this taxonomy is given in Figure 4.3 and Figure 4.4. Only the relevant classes and instances for a particular example are represented in these figures. The plain arrows represent a link between a class (or instance) and its direct upper class. The dash arrows represent a link between a class and an upper class indicating that the direct upper class is not shown in the figure.

Figure 4.3 shows the representation of the conceptual class **PERSON** and its instance **JUAN**. Several of the attributes describing these concepts are also shown in the figure. The class **PERSON** possesses, among others, the attributes **age**, **father** and **height**. All of these attributes are classified in the attribute taxonomy on the basis of their linguistic realization. For example, let us consider the attribute **age** describing the concept **JUAN**. It is realized

as the definite nominal group *la edad* (*the age*). This attribute belongs to the class **OF_QUANTITY**. For this reason, the value of the attribute can be consulted following the general form of attributes in class **OF**. The resulting clause will be:

¿Cuál es la edad de Juan? (*What is Juan's age?*)

However, the most natural form of consulting this attribute is following the general form associated with the class **OF_QUANTITY**. That is using the clause:

¿Cuántos años tiene Juan? (*How old is Juan?*)

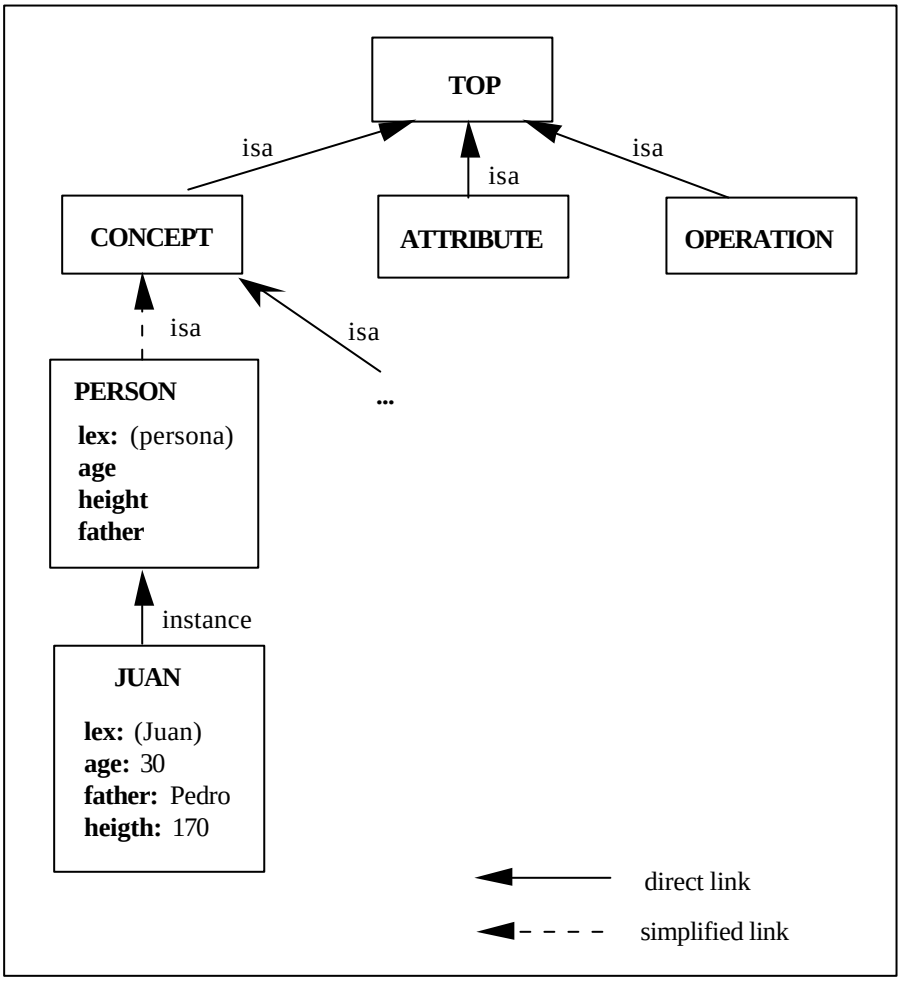

*Figure 4.3: The class **PERSON** and its instance **JUAN***

Figure 4.4 shows the representation of the attributes describing the concept **PERSON** in the taxonomy of attributes. The attributes **age** and **height** belong to the subclass **OF_QUANTITY**. The attribute **father** belongs to the class **OF_PERSON_SIM**. The

links to the linguistic realizations associated with the attribute are represented in the facet **lex**.

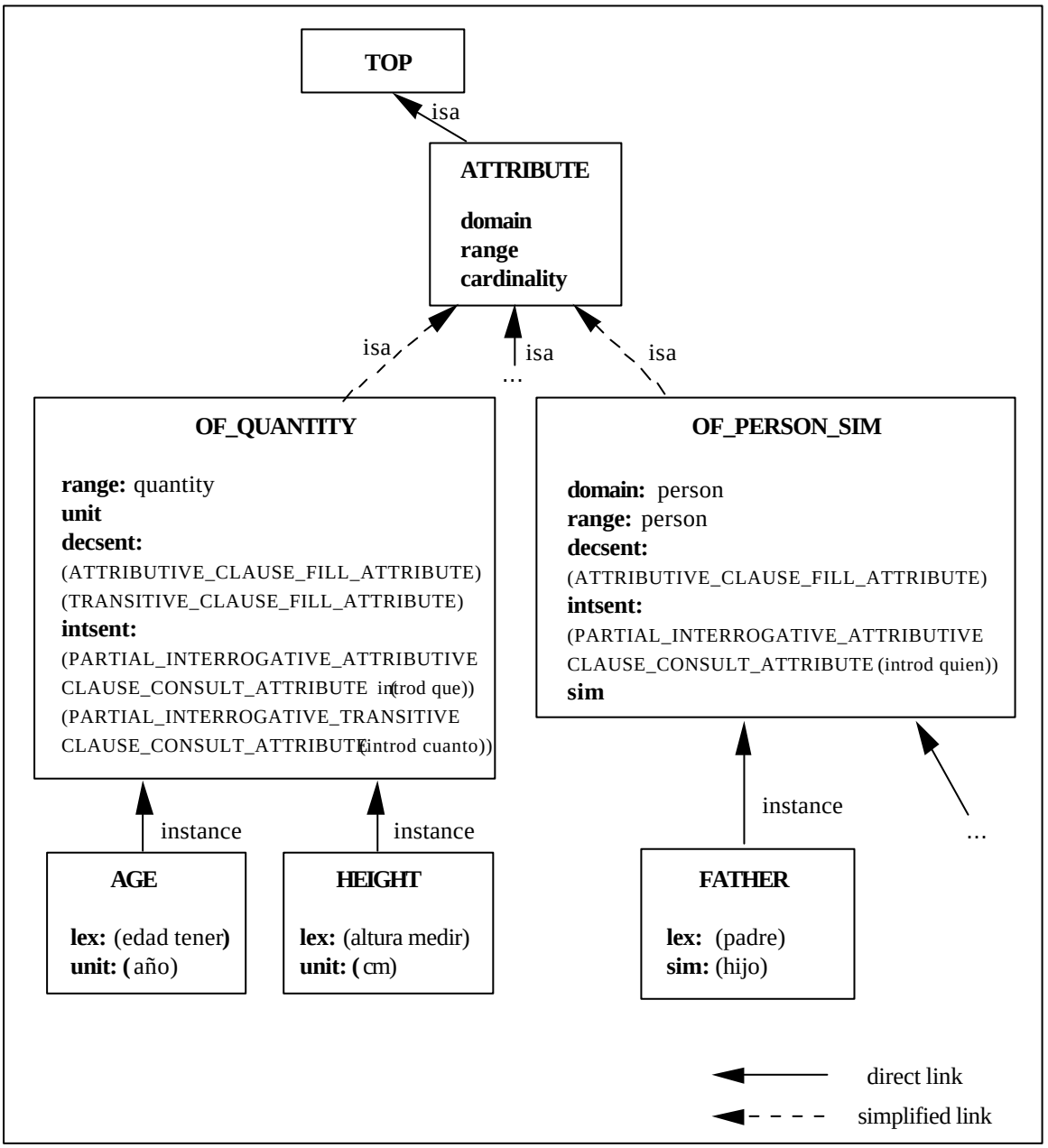

*Figure 4.4: The representation of the attributes describing the concept **PERSON***

The LO structures associated with the filling of the attributes are represented in the facet **decsent**. This facet relates the class **OF_QUANTITY** to the LO classes **ATTRIBUTIVE_CLAUSE_FILL_ATTRIBUTE**, representing the general form for filling attributes in class **OF** and **TRANSITIVE_CLAUSE_FILL_ATTRIBUTE**. The

class **OF_PERSON_SIM** is also related to the LO classes **ATTRIBUTIVE_CLAUSE FILL_ATTRIBUTE**.

The facet **intsent** relates the classes to the structures necessary for consulting them. The class **OF_QUANTITY** is related to the LO classes **PARTIAL INTERROGATIVE ATTRIBUTIVE_CLAUSE_CONSULT_ATTRIBUTE** and **PARTIAL INTERROGATIVE_TRANSITIVE_CLAUSE_CONSULT_ATTRIBUTE**. The interrogative pronoun associated with the clauses in the first class would be *qué* (*what*) and the interrogative adverb associated with the second would be *cuánto/cuántos* (*how much/how many*). The facet **intsent** relates the attributes belonging to the class **OF_PERSON_SIM** to the LO class **PARTIAL_INTERROGATIVE ATTRIBUTIVE_CLAUSE_CONSULT_ATTRIBUTE**. The interrogative pronoun associated with clauses in this class is *quién* (*who*).

The same classification would be used for describing these attributes in English. The only relevant difference in the realization of the classes of attributes described is that the attributes in the class **OF_QUANTITY** expressed by an specific adjective in English (such as in *Juan is 170 cm tall*), in Spanish are expressed by means of a specific verb (such as in *Juan mide 170 cm*).

From the representations in Figure 4.3 and Figure 4.4, in order to fill the attributes **age**, **height** and **father** of the instance **JUAN**, the interface would be able to accept the following sentences:

Juan tiene 30 años (*Juan is 30 years old*)
La edad de Juan es 30 años (*Juan's age is 30*)
Juan mide 170 cm. (*Juan is 170 cm. tall*)
La altura de Juan es 170 cm. (*Juan's height is 170 cm.*)
El padre de Juan es Pedro (*Juan's father is Pedro*)
Juan es hijo de Pedro (Juan is Pedro's son)

Examples of sentences that could never be accepted from this representation are the following:

Juan mide 30 años (*Juan is 30 years tall*),
Juan tiene 30 cm. (*Juan is 30 cm old*),

Juan is Pedro's father (*Juan is Pedro's father*)

All attributes describing concepts are represented in the attribute taxonomy. For example, all attributes describing the concept **TRAIN_STATION**, shown in Figure 4.4, were classified regarding its linguistic realization.

The attribute **name** belongs to the subclass **OF_NAME.** The attributes in the class **OF_NAME** can be consulted using the form general to attributes in the class **OF**. In this example, <concept_name> would correspond to the group *la estación* (*the station*) and the <*attribute_name*> to *el nombre* (*the name*). The resulting clause will be:

¿Cuál es el nombre de la estación? (*What is the station's name?*)

However, following the more specific form for consulting attributes in the class **OF_NAME** the resulting clause will be:

¿Cómo se llama la estación? (*What is the station's name?*)

The attribute **city** belongs to the basic class **OF_PLACE**. The filling of this attribute is expressed in the clause:

La estación de Sants está en la ciudad de Barcelona
(*[The] Sants station is in the city of Barcelona*)

The attribute **connects_to** belongs to the basic class **DOES**. The more natural form of expressing the operation of filling this attribute is:

La estación conecta con <attribute_value>
(*The station connects with <attribute_value>*)

Compound attributes, those expressed by two different terms, each representing a different grammatical role were also considered. Two subclasses of compound attributes were distinguished: the class **IS_SUBJECT** and the class **DOES_SUBJECT**. The first term associated with attributes in these two classes represents an entity related to the concept and the second term describes this entity. The class **IS_SUBJECT** is a subclass of the class **IS**. It details the being or state of an entity describing the concept. The class **DOES_SUBJECT** is a subclass of the class **DOES** describing the action carried out by an entity describing the concept. The expression of attributes in these classes requires a simple clause where the first term plays the role of subject. For example, the attribute **expenses_justified** describing the concept **PAYMENT_REQUIREMENT** belongs to the

compound class **IS_SUBJECT**. The range of this attribute is the set **yes/no**. The filling of this attribute is expressed by an attributive clause having as its subject the first term (*the expenses*) and as its attribute the second term (*justified*).

New classes can be added to the basic taxonomy when required.

The taxonomy of attributes is intended to be reusable across several languages. As mentioned above, the NL used in the interfaces generated by GISE is Spanish. Thus, the attribute classification is based on Spanish linguistic distinctions. Because most linguistic considerations in classifying attributes are relevant in other languages, this taxonomy could be reused with few changes. As an example of this, the same classification has been used to obtain interfaces in Catalan. Experiences in multilingual systems, such as those with the GUM, described in [Bateman94], demonstrate that linguistically motivated representations are easily reusable across other languages when the abstract distinctions considered are close enough to surface regularities, but are beyond specific surface realizations.

4.5 OPERATIONS

All communication tasks required during the exploitation phase are described in the CO. These tasks are represented as a taxonomy of operations over the concepts representing the application in the CO. This taxonomy of operations is included in the general level of the CO. The operations required for a particular application are obtained from the general operations. Specific application operations are represented in the application level.

Communication with the user is reduced to the expression of most of these operations. Although the proposed design has been focused on the operations expressed in the interaction with the user, a few of these operations perform tasks assuring the consistence of CO that are not expressed during the communication. These operations are described as internal entities, the value of the facet **interface_entity** is **no**. An example of such operations is the operation for removing unconnected concepts.

In this design, the consistence of the CO is assured by guiding the user in introducing correct sentences, those corresponding to consistent operations. However, because the

interface design allows the introduction of the user sentences not only by choosing the guided-menu options but also by typing them, operation-performing consistence tasks have also been considered.

The operations are considered both in isolation and in combination. They are classified as constructive or consultative and as simple or complex. Simple operations are those operations over one single object. Simple constructive operations include creating and removing unconnected concepts, filling attributes of a concept, adding or removing values and connecting instances. Simple consultative operations include querying for the instances of a class and obtaining the values of an attribute. Complex operations involve several of these simple ones, e.g., creating a new instance with its attributes and classifying it in the ontology.

All operations have a signature composed by a set of arguments (operands). Arguments can be optional or obligatory and can be constrained in several ways. Preconditions can be attached to operations to allow them to be triggered.

Operations, like concepts and attributes, are represented as objects in a taxonomic structure. The arguments and preconditions of each operation are represented as facets. The operation arguments are CO objects: concepts (classes and instances), attributes, values and other operations. For each specific application, all allowed operations are generated.

Figure 4.5 shows a fragment of the CO representing several operations. As shown in this figure, operations are represented as objects and their arguments as facets describing these objects. These facets are **con**, representing a conceptual class, **ins**, representing a conceptual instance, **attr**, representing a conceptual attribute and **val**, representing an attribute value. The facets **pco** and **pcc** describing the classes of operations represent preconditions governing the operation performance. The formalism defining the preconditions is explained in next section.

The operations described in Figure 4.5 belong to the class **SIMPLE_CONSTRUCTIVE_O**, representing all simple operations modifying the application case level of the CO. These operations perform the creation or modification of an instance of a conceptual class defined in the application case level.

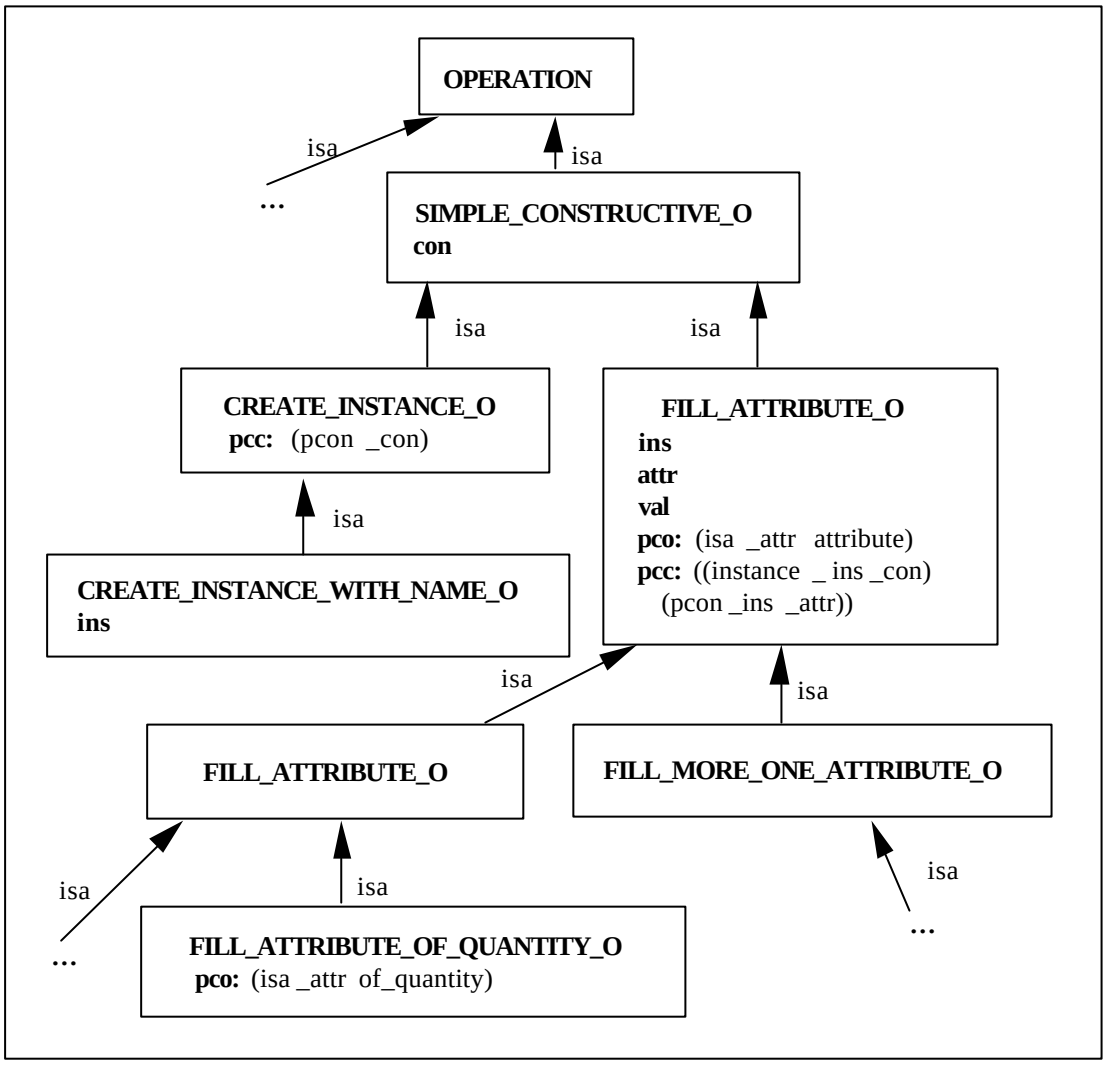

Figure 4.5: A fragment of the taxonomy of operations in the CO

Simple operations filling attributes of instances are classified according to the class of the attribute to fill. For example, the **FILL_ATTRIBUTE_OF_QUANTITY_O** class, shown in the figure, represents the operation of filling one attribute belonging to the class **OF_QUANTITY**.

In order to consider only reasonable modes of interaction and offer them to the user by means of the generated grammar, the system allows the definition of macro-operations or complex operations. These macro-operations are used to implement some common ways the users have of describing their information. People usually communicate their knowledge by following paths attached to the possible connections between the objects they are referring to. For example, after mentioning a new object, users usually want either

to describe it or to connect it to previously defined objects. This can be accomplished by defining the corresponding macro-operation in the CO application level.

An example of macro-operation is that involved in creating an instance, filling its attributes and creating and describing all concepts appearing in its description that were not previously created. The simple operation of filling one attribute (shown in Figure 4.5) allows relating the instance described to another instance only if both instances have previously been created. The macro-operation describing both a new and related instance allows a more natural way of introducing new information.

Macro-operations can also be used to force the user to fill a mandatory attribute after creating a new instance. Once the user has introduced a sentence to create an instance, only choices relating to the instance-mandatory attributes appear on the screen.

4.6 PRECONDITIONS

Preconditions are attributes used in operations to represent the conditions that must hold in order for an operation to be executed. Like other object descriptors, preconditions are inherited through taxonomic links. The preconditions associated with the operations are those concerning the correctness of its arguments. There are, in addition, preconditions that can be associated with the concepts. Those preconditions describe the conditions that must hold in order to create an instance of a specific concept, as well as those to fill each of its attributes described in the conceptual class.

Preconditions can only be defined in the classes of the objects. Preconditions are considered both during the process of obtaining the application-restricted grammar and during communication. They ensure that only consistent grammar rules are generated and that they are activated correctly during the communication process.

Two types of preconditions are distinguished: **ontology preconditions** and **case preconditions.**

Ontology preconditions are preconditions constraining specific CO conceptual classes and their attributes. Because this information would not be modified during the

communication, ontology preconditions can be evaluated during the process of obtaining the linguistic structures for an application. These preconditions are represented as the value of the facet **pco**.

Case preconditions are dynamic conditions that must be checked during the communication process. Most constraints between objects and their attributes cannot be checked when generating the subgrammar for an application. These constraints depend on the instances existing at each specific moment of the communication process. Case preconditions are conditions governing instance existence and the values of their attributes. These preconditions ensure, for example, that an instance exists before its attributes are filled. These preconditions are represented as the value of the facet **pcc**. They are incorporated into interface grammar rules and activated during the communication process.

The formalism used to represent preconditions is described in Figure 4.6.

As shown in Figure 4.5, the ontology precondition of the operation **FILL_ATTRIBUTE_O** states that the attribute to fill must be previously defined in the attribute taxonomy of the CO. This precondition is represented by the predicate: **(isa _attr attribute)**. This predicate indicates that the attribute to fill, represented by the variable **_attr** is related to the class **ATTRIBUTE** by the relation **isa**.

The arguments of the operations are represented as facets. In the operation preconditions, the arguments are represented by the variable which name consists of the underscore character followed by the facet identifier. For example, the variable **_attr** in the precondition described above, represents the value of the facet **attr**.

The case preconditions of the operation **FILL_ATTRIBUTE_O** ensure that the instance is created before its attribute is filled and that the preconditions associated with the attribute in the concept description are satisfied. The first case precondition is represented by the predicate **(instance _ins _con)**. It indicates that the instance, represented by the variable **_ins**, is related by the relation **instance** to a concept class represented by the variable **_con**. The last precondition, represented by the predicate **(pcon _ins _attr)**, is a reference to the preconditions on the attribute associated with the concept description. The attribute is represented by the variable **_attr** and the conceptual instance is represented by the variable **_ins**. All subclasses of the operation **FILL_ATTRIBUTE_O** inherit these preconditions.

```
< operation-preconditions > ::=
        < conceptual-preconditions > < case-preconditions >

< conceptual-preconditions > ::=
        pco: ({< consulting-class-predicate >})

< case-preconditions > ::=
        pcc: ({< consulting-instance-predicate >} /
              {< reference-concept-preconditions >})

< consulting-class-predicate > ::=
        < isa-predicate > / < has-value-predicate >

< consulting-instance-predicate > ::=
        < instance-predicate > / < has_value-predicate >

< reference-concept-preconditions > ::=
        (pcon  concept-identifier) /
        (pcon  concept-identifier attribute-identifier)

< concept-preconditions > ::=
        {(attribute-identifier < consulting-instance-predicate >)})

< isa-predicate >   ::=
        (isa  object-identifier  object-identifier)

< instance-predicate > ::=
        (instance object-identifier object-identifier)

< has-value-predicate >  ::=
        (has_value object-identifier object-identifier)

< object-identifier > ::=
      concept-identifier / attribute-identifier / value-identifier / < variable >

< variable > ::= _identifier
```

Figure 4.6: The formalism used to represent preconditions

An example showing how case preconditions are attached to a conceptual class is shown in Figure 4.7. This figure describes the conceptual class **BUILDING_REQUIREMENT**. This concept belongs to the CO representing SIREDOJ, the ES in law described in the previous chapter. SIREDOJ is specialized in demands on duty related to building contracts. The building requirement is one of the duties assumed in a building contract. If for a specific case this duty has not been fulfilled, the specialist has to know the reasons in order to advise the user about legal actions to be taken.

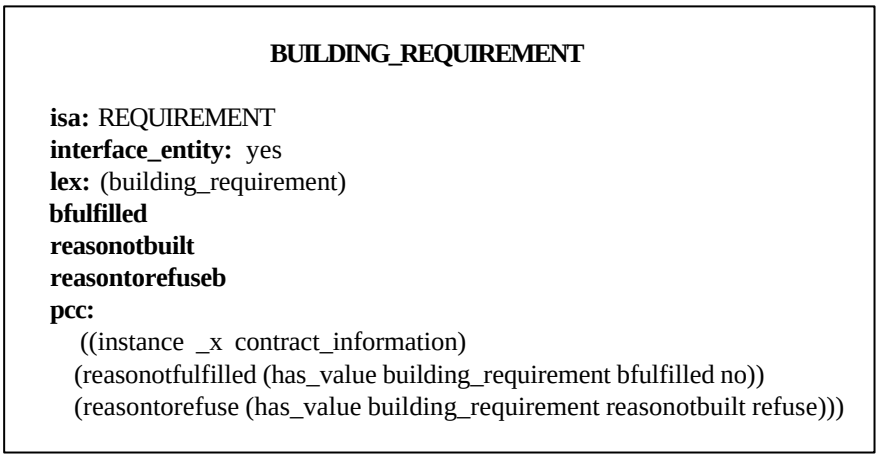

*Figure 4.7: The representation of the concept **BUILDING_REQUIREMENT***

The case preconditions associated with the concept **BUILDING_REQUIREMENT** are represented in the facet **pcc**. This facet is filled with a list of three specific preconditions following the formalism described in Figure 4.6. These preconditions will guide the user to introduce information about the building requirement for a specific contract. These three preconditions are described below.

The first precondition is **(instance _x contract_information)**. This precondition states that an instance of the class **BUILDING_REQUIREMENT** can only be created if an instance of the class **CONTRACT_INFORMATION** exists. In this precondition the variable **_x** represents any existing instance of the **CONTRACT_INFORMATION** class.

The second precondition states that the attribute **reasonotbuilt** (representing the reasons for not building) can only be filled if the value of the attribute **bfulfilled** is **no**. That is, the user will have to enter the reasons for not building only in the event of construction not being undertaken.

The third precondition states that the attribute **reasontorefuseb** (representing the reasons to refuse to build) can only be filled if the value of the attribute **reasonotbuilt** is **refuse**. That means the user will have to enter the reasons for refusing to build only if the party that assumed building obligations has refused.

The following section details the attachment of the case preconditions to the grammar rules generated for an application, as well as their evaluation during the communication process.

CHAPTER 5

LINGUISTIC ONTOLOGY

This chapter describes the linguistic ontology representing the linguistic knowledge required for communication with KBSs.

5.1 INTRODUCTION

None of the general available linguistic resources was easy to adapt to the general design proposed in this work. The main reason was the language, because there are few NLP systems supporting NL communication in Spanish. Another problem was that existing linguistic resources were not easy to adapt to generate application-restricted subgrammars supporting communication with KBSs.

Most works on grammar development environments are oriented to analysis. The few grammar development environments oriented to generation follow different approaches to those that are analysis-oriented. Linguistic resources for language processing and for generation have not yet come together, as pointed out by Bateman in [Bateman97b]. Since early work on generations, functional and pragmatic issues have been considered as the central areas, while most relevant works on analysis consider structure and syntax the central areas when organizing linguistic resources. Different approaches in generation focus the role of function to a greater or lesser degree.

Although the linguistic resources in GISE are not used for generating NL but for generating grammars to analyze user interventions, a linguistic organization focussing the role of function was considered appropriate. For this reason, the linguistic knowledge was organized following the basic principles of the Nigel grammar ([Nigel88]), a large systemic functional grammar (SFG) of English based on Halliday's work ([Halliday81]). This grammar has been implemented as a component of the Penman text generation system. As a result of many years of linguistic research and having been implemented for a NLP system independent of any particular knowledge domain, the Nigel grammar covers most important details to be considered in NLP. This grammar supports a broad coverage of linguistic descriptions that provides mappings from enriched semantic specifications to corresponding surface strings.

The main reason for adapting the basic principles followed by this grammar to our proposal is that it places the communication function in the foreground. The functional information concerning the communicative intent of some utterance determines the basic organizations.

Functional grammars interpret language as a resource used in context. These grammars assume that language is generated with some goal, to satisfy a particular need. For this reason, they are especially appropriate for systems generating language or linguistic resources for different domains and applications. Functional grammars are based on the assumption that the differentiation of syntactic phenomena is always determined by the function.

Grammatical structures (or units) are seen as configurations of functions in these grammars. The constituents are viewed as fulfilling identifiable grammatical functions in the context of the grammatical unit of which they form part.

Systemic-functional grammar is a particular kind of functional grammar. It is a theory of grammar as a resource for expressing meanings. Meanings are realized by a network of interlocking options. Grammatical forms are obtained by making choices in this network. Three different types of metafunctions are distinguished in order to organize the different kinds of functinalities that all uses of language achieve. These metafunctions cover all kinds of functionalities that all uses of language achieve. These metafunctions are the Interpersonal, the Ideational and the Textual. The Interpersonal metafunction is concerned with how we interact with others in the environment. It determines the type of interaction

(the mood, the modality, etc.). The Ideational metafunction is concerned with propositional meaning and content. The Textual metafunction involves organizing the information to reflect the emphases of the speakers. This metafunction is especially important in multisentential texts.

The Nigel component has been designed and implemented following these theoretical distinctions. It consists of the grammar and an interface between that grammar and the environment. The information from the environment is obtained by making inquiries about conceptual knowledge. The information required by the inquiries is provided by the linguistically-motivated taxonomy, called the **upper model**. The **upper model** serves to organize the conceptual knowledge that needs to be expressed in the language. All concepts in the conceptual base representing any domain are related to the concepts in the upper model taxonomy.

5.2 REPRESENTING THE LINGUISTIC STRUCTURES REQUIRED FOR COMMUNICATION WITH KBSs

The design and implementation of the linguistic knowledge in this proposal, although following the basic principles, differs significantly from that of the Nigel grammar. In the design proposed in this work only the linguistic knowledge necessary for Spanish communication with KBSs has been considered. However, this knowledge could easily be extended in order to consider other languages and applications following the Nigel grammar, which is a well-developed account of English grammar.

This general linguistic knowledge has been represented as an ontology. We have considered it to be the form of representation best suited to this our proposal.

5.2.1 The basic principles followed in the LO design

The linguistic structures constituting the general grammar are described as classes. The distinctions represented in the class subsume all kinds of grammatical variations.

Defining the general grammatical structures needed for communication with KBSs in a declarative and reusable form optimizes the process of obtaining the particular grammatical structures adapted to each application. Besides, it facilitates the enlarging of the linguistic resources when considering their different purposes.

The LO was designed by using the same form of representation as used in the CO, in an object-like fashion where basic elements are objects described by a set of facets. Using the same form of representation for conceptual and linguistic sources simplifies the implementation of the relations between the two data structures.

Following the basic principles appearing in the Nigel grammar, the linguistic knowledge is organized in two dimensions: **rank** and **metafunction**. Rank determines the scale at which the grammatical structures are represented: **clause**, **group** and **word**. Metafunction describes the three types of meaning described above: the Interpersonal, the Ideational and the Textual. These two dimensions intersect and give all required grammatical structures (or units). These structures are obtained by the making of appropriate choices of their functions. The grammatical units are represented as linguistic classes.

Two levels are distinguished in the LO: the **general level** and the **domain level**. The general level describes general linguistic classes. The various grammatical structures required for communication with all KBSs were defined in the domain level. In this level, the linguistic information in the general level is particularized to support the expressions of the communication acts required in interfaces to KBSs. Including the domain level in the LO improves the process of adapting the general grammatical structures to those required for each specific application. In the domain level, the general classes are further subclassified considering the taxonomy of operations representing the communication tasks in the CO. The resulting classes represent all grammatical structures required for communication with KBSs. These classes are used to obtain the grammatical resources required to express the specific operations that can be performed for each application.

The linguistic classes represented in the general level are described in next section. The classes described in the domain level are described in Section 5.4. The process of obtaining the specific linguistic structures for each application is described in Chapter 6.

5.3 GENERAL LINGUISTIC CLASSES

As mentioned above the linguistic knowledge is organized in two dimensions: **rank**, representing the scale and **metafunction**, representing the different functions. The most general classes in the LO are organized following the dimension **rank** and then are subclassified considering the different possible functions within each class.

5.3.1 The rank dimension

The top class of the LO is **rank**. Within the top class, three main subclasses were defined: the **CLAUSE** class, the **GROUP** class and the **WORD** class.

5.3.1.1 The CLAUSE class

A user intervention is simply any combination of clauses, minimally one clause. Clauses may combine to form clause complexes but these clause complexes do not constitute a rank higher than the clause.

The **CLAUSE** class has two subclasses: the **MAJOR_CLAUSE** class and **MINOR_CLAUSE** class. Figure 5.1 shows how clauses are classified in the dimension rank.

The **MAJOR_CLAUSE** class has a subject and a finite verb, although in Spanish the subject can be elliptic. An example of major clause is the user intervention *Las obligaciones del contrato están cumplidas* (*The contract requirements are fulfilled*), appearing in the communication with the ES SIREDOJ.

The **MINOR_CLAUSE** class represents those clauses not containing a finite verb. Such clauses contain one or more nominal groups. An example of minor clause is the user

intervention *Estación destino* (*Final station*), in a consulting train system. Calls (*Sir!*), greetings (*Hello!*) and exclamations (*Ole!*) are also included in the **MINOR_CLAUSE** class.

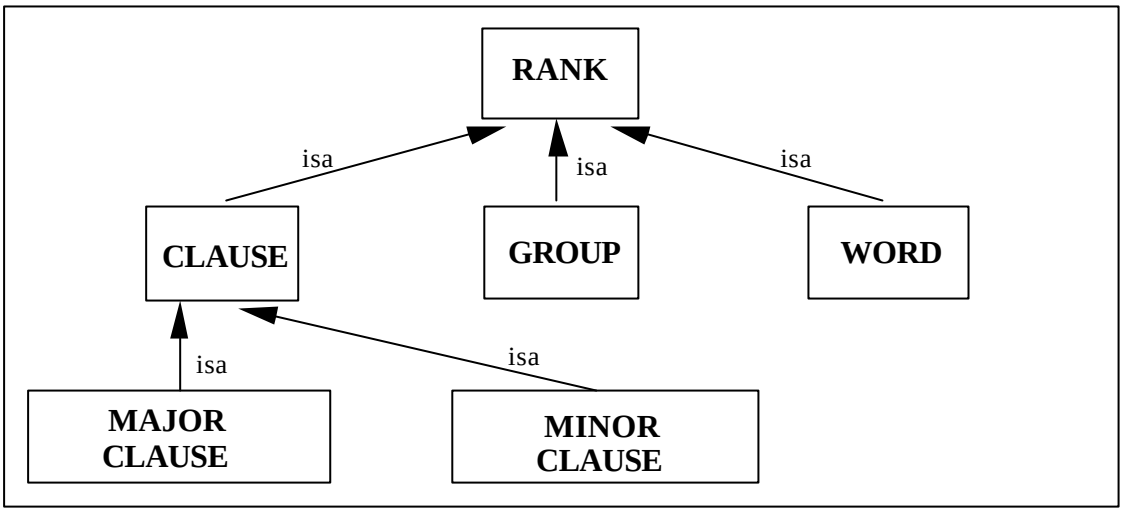

Figure 5.1: The classification of clauses in the dimension rank

5.3.2.2 The GROUP class

The grammatical unit **group** is structured as a group of words. The basic principle of structural organization in groups is modification. A group has a head and a variable number of modifiers.

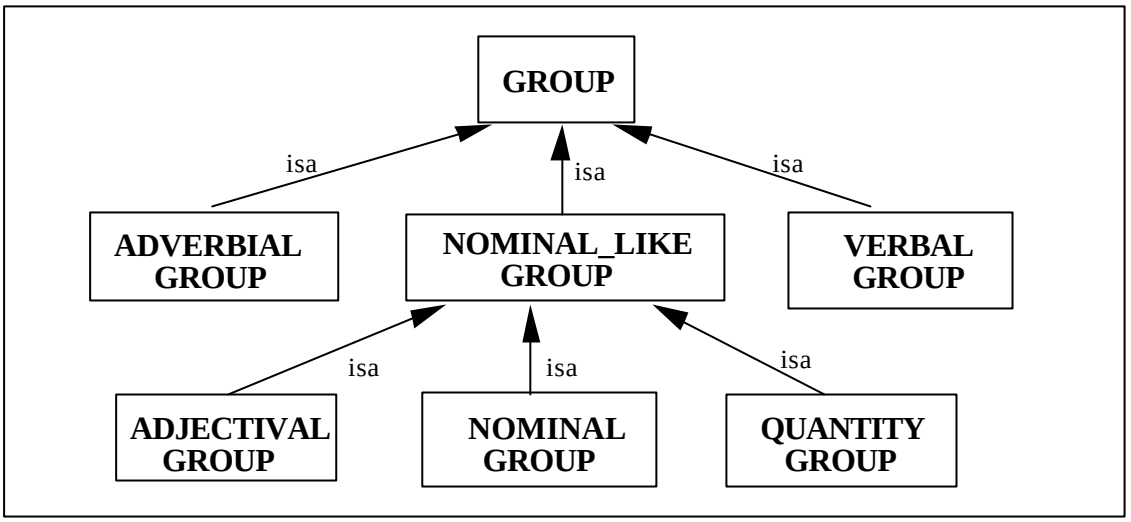

Figure 5.2: The classification of groups in the dimension rank

The class **GROUP**, representing all groups was subdivided into three subclasses: the class **NOMINAL_LIKE_GROUP**, the class **VERBAL_GROUP**, and the class **ADVERBIAL_GROUP**. These three classes cover all possible realizations of the participants in the class. Nominal-like groups represent actors, goals, attributes, etc.; verbal groups represent processes, and adverbial groups represent circumstances (manner, cause, etc.).

The **NOMINAL_LIKE_GROUP** class is subdivided into three classes according to the class of word serving of the head. The **NOMINAL_GROUP** class has a noun as head. The **ADJECTIVAL_GROUP** class has an adjective. The **QUANTITY_GROUP** class has a quantity.

Figure 5.2 shows the classification of groups in the dimension rank.

5.3.1.3 The WORD class

The **WORD** class represents the minimum linguistic unit. Elements in the class **WORD** cannot be decomposed. The classes distinguished in this rank are: **NOUN**, **VERB**, **ADJECTIVE**, **ARTICLE**, **ADVERB**, **PREPOSITION** and **CONJUNCTION** and their subclasses.

The **NOUN** class was further subclassified into three classes: the **COMMON_NOUN** class, the **PROPER_NOUN** class and the **PRONOUN** class.

Pronouns were considered as a subclass of nouns because pronouns have most of the same functions and positions as nouns. Following traditional Spanish grammar pronouns were subdivided into the following classes:

- **PERSONAL**, representing pronouns such as *yo* (*I*), *tu* (*you*)
- **POSSESSIVE,** such as *mío* (*mine*), *tuyo* (*yours*)
- **DEMOSTRATIVE**, such as *éste* (*this*), *ése* (*that*)
- **RELATIVE**, such as *que* (*that*), *cual* (*which*)
- **INDEFINITE**, such as *algo* (*something*), *alguien* (*someone*) and
- **INTERROGATIVE**, such as *qué* (*what*), *cuál* (*which*).

Verbs were classified according to the kind of complement they may have. The **VERB** class was subdivided into the two classes: **PREDICATING_VERB** and **COPULATIVE_VERB**.

Copulative or linking verbs are those of incomplete predication; they merely announce that the real predicate follows. Copulative verbs were subdivided into *ser/estar* (*be*) and all the others. One of the most important differences is that, whereas the other linking verbs are followed mostly by predicative adjectives, *be* may be followed by many types of complement such as adjectives, nouns, adverbs, noun clauses, etc.

The predicating verbs are those that are not copulative. A predicating verb always denotes an event. Predicating verbs are subdivided into two classes: **TRANSITIVE_VERB** and **INTRANSITIVE_VERB**. A transitive verb takes a direct object while an intransitive verb does not require an object. Verbs that may be used either transitively or intransitively are represented as members of both classes.

The **ADJECTIVE** class was subdivided into the **CONNOTATIVE_ADJECTIVE** and **NOT_CONNOTATIVE_ADJETIVE** classes according to the information they give. Connotative adjectives are those having meaning. They were classified in the **DESCRIPTIVE_ADJECTIVE** class and the **NUMERAL_ADJECTIVE** class. Descriptive adjectives usually indicate an inherent quality or a physical state such as age and size. Numeral adjectives are subdivided into **CARDINAL_ADJECTIVE**,

determining a quantity, and **ORDINAL_ADJECTIVE**, determining a position. The remaining adjectives are not connotative; they add no new meaning.

The **ARTICLE** class has two subclasses: The **DEFINITE_ARTICLE** and the **INDEFINITE_ARTICLE** classes.

The class **ADVERB** was subdivided, according to the function the adverb perform, in the following classes:

- **CONCEPTUAL_ADVERB**
- **CLAUSE_ADVERB**
- **CONJUNTIVE_ADVERB**
- **EXPLANATORY_ADVERB**
- **REALTIVE_ADVERB**
- **INTERROGATIVE_ADVERB**

Conceptual adverbs are those modifying the meaning of the verb, adjectives and other adverbs. They have been subdivided according to the meaning they have in the following classes:

- **MANNER_ADVERB**
- **PLACE_ADVERB**
- **TIME_ADVERB**
- **QUANTITY_ADVERB**

Clause adverbs modify the whole clause. Such adverbs may be considered as equivalents of a sentence or a clause. Three subclasses are distinguished in this class:

- **AFFIRMATION_ADVERB**, representing adverbs such as *si* (*yes*) and *ciertamente* (*certainly*)
- **NEGATION_ADVERB**, representing adverbs such as *no*
- **POSSIBILITY_ADVERB** class, representing adverbs such as *quizás* (*maybe*) and *posiblemente* (*possibly*)

Conjunctive adverbs establish a relationship between one clause and the preceding clause. Examples of these adverbs are *entonces* (*then*) and *sin embargo* (*however*).

Explanatory adverbs illustrate or enumerate. Examples of these adverbs are *por ejemplo* (*for example*) and *como* (*such as*).

Relative adverbs introduce subordinate clauses, such in the clause *El hombre que assignó un contrato de obra al constructor no ha pagado* (*The man who assigned a building contract to the constructor has not paid*).

Interrogative adverbs are those introducing questions, such in *¿Cuando llegará el tren?* (*When will the train arrive?*) and clauses derived from questions such as *He preguntado cuando llegará el tren* (*I asked when the train would arrive*).

5.3.2 The metafunction dimension

The linguistic classes representing clauses and nominal groups, described above, were further subclassified according to metafunctional information. New subclasses were obtained considering the several functions that the **CLAUSE** and **NOMINAL_GROUP** classes can achieve to express the communications tasks represented in the CO. This metafunctional information determines the linguistic features associated with each grammatical class. Three types of functions are distinguished: **Interpersonal**, **Ideational** and **Textual**. Functions belonging to these three different types of meaning are combined. Not all combinations of functions are possible. The resulting classes obtained when considering the information appearing in communication with KBSs are described bellow.

5.3.2.1 Ideational information

The ideational information is the resource for representing the world and its organization: configurations of processes, participants, and circumstances, objects, qualities, etc. Ideational information concerns logical and experiential information. The experiential information determines the content expressed while logical information determines the logical relations between the conceptual entities. The logical information basically determines the complexity at clause and group level.

In this work proposal, all this ideational information is represented in the functions: **verb**, **subject**, **attribute**, **direct object**, **indirect object**, **circumstances**, **reference** and **simple**.

The content information is the kind of meaning that has received the greatest attention.

Following the Nigel grammar, all major clauses describe a process, state or event.

The process/state/event consists of three components:

- The process/state/event
- The participants
- The circumstances associated with it

This content information about the components at clause level is represented by the functions: **verb**, **subject**, **attribute**, **direct object**, **indirect object**, **circumstances** and **reference**.

The function **verb** represents the process or state described by the clause. It is realized by a verbal group.

The function **subject** represents the main participant of the clause. In clauses describing a process, the function **subject** determines its actor. In clauses describing a state the subject determines the entity described. The subjects of clauses are realized by nominal groups. In Spanish, the participant subject is present in most clauses. In some clauses, however, the subject can be elliptic, which means it can be understood although it is not expressed in the clause. There are also impersonal clauses, clauses that do not include a subject, such as the Spanish clause *Se cumplieron las claúsulas esenciales.*

The function **attribute** represents the participant describing the subject in clauses outlining a state. This function is realized by nominal-like groups.

The different types of circumstances describing a process or state are represented by the function **circumstance**. They are realized by adverbial and nominal like groups.

Following this information, the **MAJOR_CLAUSE** class was subclassified into two classes: the **ATTRIBUTIVE_CLAUSE** and the **PREDICATIVE_CLAUSE** classes.

Attributive clauses are those defined by the functions **subject**, **verb** and **attribute**. The verb belongs to the class **COPULATIVE_VERB**. Attributive clauses always describe a state. The clause *La obligación de pago está cumplida* (*The payment requirement is fulfilled*) is an example of attributive clause.

The remaining major clauses are represented in the class **PREDICATIVE_CLAUSE**. The verb in these classes is a copulative verb. Predicative clauses are subdivided into two classes according to the function **direct object**: the class **TRANSITIVE_CLAUSE** and the class **INTRANSITIVE_CLAUSE**. In transitive clauses the participant **direct object** is always represented, while in intransitive clauses it is not. The clause *El propietario pagó la casa* (*The owner paid for the house*) is an example of transitive clause. An example of intransitive clause is the clause *El tren llega a las 8:15* (*The train arrives at 8:15*).

A subclass of transitive clauses is further distinguished: the class **INDIRECT_OBJECT_CLAUSE**, representing all clauses having the participants **direct object** and **indirect object**. An example of such clauses is *El propietario encargó una construcción al constructor* (*The owner assigned a building to the constructor*).

In Spanish, there are also intransitive clauses having an indirect object, that is, having the participant **indirect object** but not the participant **direct object**. Those clauses, however, were not considered in this proposal because they do not usually appear in the context of communication with KBSs.

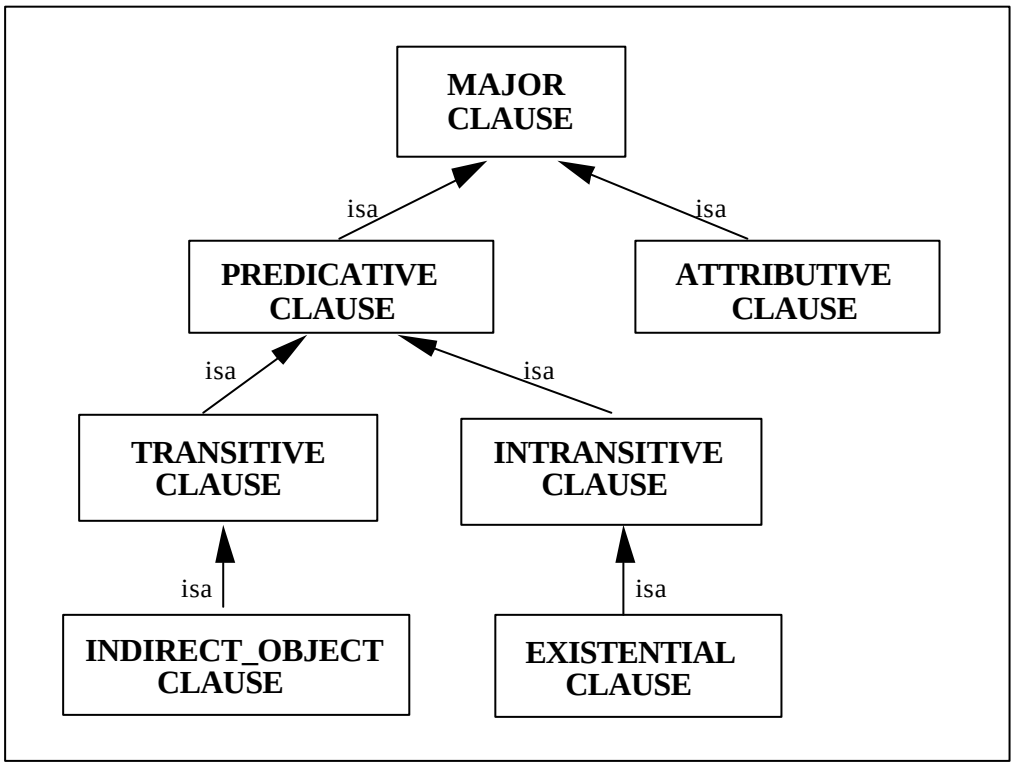

Figure 5.3: The classification of clauses according to the content information

The classification of major clauses according to the content information is shown in

Figure 5.3.

The class **EXISTENTIAL_CLAUSE** is a subclass of the class **INTRANSITIVE CLAUSE** representing those clauses having as constituents a subject and a verb representing existence. Examples of such clauses are *Existe un contrato* (*A contract exists*) and *Hay obligaciones* (*There are duties*). Existential clauses are important in this proposal because they express the operation to create an instance of a CO concept. The class **EXISTENTIAL_CLAUSE** is described in Figure 5.6.

A function that is also important in the language used in user interventions is that of **reference**. This function indicates whether or not there is a participant in the clause referring to a participant appearing in a previous clause. In the dialogues supported in this proposal, user interventions consist of clauses expressing operations over the CO concepts. If the clause does not contain the name of the concept, it must refer to a concept appearing in a previous clause. For instance, in the sentence *Existe un contrato y la obligación de pago está cumplida* (*There is a contract and the payment requirement is fulfilled*), the subject of the second clause, *la obligación de pago* (*the payment requirement*), refers to the subject of the first clause, the concept realized as *un contrato* (*a contract*). This information is obtained from the participants of the clause. All clauses have been subclassified into clauses that support reference and clauses that do not.

At group level, the content information considered is that determining the type and function of the nominal group, and if it refers to a nominal group appearing in a previous clause. Three subclasses were distinguished according to the nominal group type:

- **COMMON_NOMINAL_GROUP**, representing common names
- **PROPER_NOMINAL_GROUP**, representing proper names
- **PRONOMINAL_GROUP**, representing pronouns

Common nominal groups consist has as constituents the head and a variable number of modifiers. Proper nominal groups and pronominal groups consist of only on the constituent head.

Nominal groups can also be modifiers of other common nominal groups. Those modifiers were subdivided into direct and indirect. Direct modifiers are directly related to the nominal group. Indirect modifiers are linked to the nominal group by a preposition.

Information indicating whether or not there is a reference to a previous nominal group is also considered at group level.

Following logical information, major clauses and nominal groups are described by the function **simple**, indicating the complexity of the class. According to this function, the **MAJOR_CLAUSE** class was subdivided into **COMPLEX_CLAUSE** and **SIMPLE_CLAUSE** classes. Complex clauses are described as the combination of one or more simple clauses. Nominal groups have also been subdivided into complex nominal groups and simple nominal groups, complex nominal groups being those having more than a one member of the **WORD** class as its head.

5.3.2.2 Interpersonal information

Interpersonal information involves personal relationships (i.e. power, formality) and it determines types of interaction. The relation between speaker and listener can determine how a certain speech act is presented. For example, a command is normally realized as an imperative clause, such as the clause *Déme información sobre los trenes a Barcelona* (*Give me information about the trains to Barcelona*), but can also be realized as an interrogative clause, such as *¿Podría darme información sobre los trenes a Barcelona?* (*Would you give me information about the trains to Barcelona, please?*).

In the LO design, this information controls mood and polarity choice at the clause level and person at group level. That is, the functions **mood** and **polarity** are represented in all clauses and the function person in the groups.

The function **polarity** determines if the clause is positive or negative. The function **mood** determines whether a clause is declarative, interrogative or imperative. This feature constrains the ordering of the constituents **subject** and **verb**. It also determines the presence of other constituents, such as interrogative pronouns as well as interrogative marks. Therefore, three subclasses of the **CLAUSE** class were distinguished according to the value of the function **mood**. These classes are: **DECLARATIVE_CLAUSE**, **INTERROGATIVE_CLAUSE** and **IMPERATIVE_CLAUSE**.

Interrogative clauses were subdivided into two classes: **INDIRECT_INTERROGATIVE CLAUSE** and **DIRECT_INTERROGATIVE_CLAUSE**.

The indirect interrogative clauses are introduced by a major clause within a verb representing an interrogative action and, optionally a conjunction. An example of an indirect interrogative clause is *Quiero saber si el tren a Barcelona ha salido* (*I want to know if the train to Barcelona has left*).

Direct interrogative clauses are characterized in Spanish by a question mark at the beginning and the end. Direct clauses were further subdivided into two subclasses: **COMPLETE_INTERROGATIVE_CLAUSE** and **PARTIAL_INTERROGATIVE CLAUSE**.

Complete interrogative clauses are those having **yes** and **no** as possible answers. Examples of these clauses are the questions *¿Existe un tren directo a Terrassa?* (*Is there a direct train to Terrassa?* and *¿Está cumplida la obligación de pago?* (*Has the payment requirement been fulfilled?*). The rest of interrogative questions are partial. Partial clauses are always introduced by an interrogative pronoun or interrogative adverb, such as in the question *¿Cuándo sale el tren a Barcelona?* (*When does the train leave for Barcelona?*).

In the type of NL communication this work is concerned imperative clauses are equivalent to interrogative clauses, imperative clauses always express consulting operations. Usually these classes are represented as a nominal group, as in the user intervention *Trenes a Barcelona* (*Trains to Barcelona*).

At group level, the interpersonal information determines the **person** describing the nominal groups. Usually, in this type of communication, the third person is used.

5.3.2.3 Textual information

Textual information concerns the organization of the text. Textual information is used for creating text in context. This information is represented in the functions: **theme**, **conjunction**, **voice** and **ellipsis** at clause level, and **determination** at group level.

The function **theme** represents the theme or subject expressed in the clause. Voice is represented by the function **voice**. According to this function, clauses were classified into the class **PASSIVE_CLAUSE** and the class **ACTIVE_CLAUSE**.

The function **conjunction** determines how clauses are linked in a user intervention. Because interventions in dialogues between users and applications must be very simple and

concise, the only conjunction considered in the current design of the LO is the conjunction *y* (*and*), connecting two or more clauses. However, other conjunctions can easily be incorporated to consider a higher level of complexity in the organization of clauses.

At group level, the function **determination** represents textual information. Following this information, the classes **DEFINITE_NOMINAL_GROUP** and **INDEFINTE NOMINAL_GROUP** were distinguished in the class **NOMINAL_GROUP**.

The function **ellipsis** is concerned with the absence of the constituent **subject**. The ellipsis of the subject was restricted to specific cases due to the fact that, although it can improve conciseness, it increases the complexity of language processing. In current design, only the subject can be elliptic and only in the case of it corresponding to the name of a concept appearing in a previous clause.

5.4 SUBCLASSES EXPRESSING THE OPERATIONS

The general linguistic classes described above were adapted to support the expression of the different tasks performed during communication.

As described in the previous section, the tasks performed during the communication consist of operations over the CO. In order to obtain the linguistic structures expressing these operations, the general classes in the LO were adapted to the taxonomy of operations in the CO. The three ranks (**clause**, **group** and **word**) were considered when obtaining the grammatical structures (or classes) required to express these operations. In the LO domain level, new subclasses were defined at the clause level according to the basic operations describing and consulting the CO concepts and their attributes. New subclasses were defined at the group and word level considering the arguments of these operations.

The linguistic structures required to express each operation filling or consulting a specific conceptual attribute are determined according to the syntactic-semantic properties of the attribute class. The taxonomy of attributes as well as the taxonomy of operations in the CO represent the information from the environment required to adapt the general grammatical

structures to express all possible communication acts. The basic classes of attributes determine the content of the constituents (subject, verb, object, attribute, circumstances) of the clauses expressing the operations.

For a specific application, new subclasses of attributes can be added to the basic taxonomy in the CO. In such a case, the linguistic classes expressing the operations dealing with attributes that belong to the new classes of attributes are obtained from the classes represented in the LO domain level.

In the LO, the objects representing the linguistic structures necessary to express the operations consulting and modifying the CO concepts assumed to be common to all applications are represented as classes in the domain level. The specific structures supporting the expression of all operations that can be performed for a specific application are obtained adapting these structures.

The subclasses defined in the LO domain level supporting the expression of modifying and consulting operations are described below.

5.4.1 The CLAUSE class

5.4.1.1 Classes expressing the simple operations modifying instances

Creating instances

Two classes were defined as representing the expression of the two operations for creating an instance. A subclass of the class **ATTRIBUTIVE_CLAUSE,** the class **ATTRIBUTIVE_CLAUSE_CREATE_INSTANCE_WITH_NAME,** was defined to express the operation of creating a conceptual instance with a specific name. In this class, the subject expresses the name of the instance and the attribute, the conceptual class identifier. A subclass of the class **EXISTENTIAL_CLAUSE**, the class **EXISTENTIAL_CLAUSE_CREATE_INSTANCE_WITH_NO_NAME**, was defined to express the operation of creating the instance without giving its name. The subject of this class represents the concept identifier.

Filling attributes

Several classes were distinguished in the classes **ATTRIBUTIVE_CLAUSE** and **PREDICATIVE_CLAUSE** to express the operation filling one attribute.

One or more subclasses representing the realization of the simple operation filling one attribute are associated with each attribute class. These linguistic classes describe the attributive, transitive and intransitive declarative clauses expressing the operation, filling one attribute. The three parameters in this operation (the concept, the attribute and the value) are expressed in the clause. However, in the referential clauses the name of the concept does not appear. Referential clauses are used only when the name of the concept has appeared in a previous clause and it is not necessary to mention it again.

The attributive classes expressing the operation to fill one attribute are the following:

- **ATTRIBUTIVE_CLAUSE_FILL_ATTRIBUTE**
- **REFERENTIAL_ATTRIBUTIVE_CLAUSE_FILL_ATTRIBUTE**
- **ATTRIBUTIVE_CLAUSE_FILL_ IS**
- **REFERENTIAL_ATTRIBUTIVE_CLAUSE_FILL_IS**
- **REFERENTIAL_ATTRIBUTIVE_CLAUSE_FILL_IS_SUBJECT**

The first two subclasses represent the general form of the filling of the attributes belonging to the classes: **HAS**, **WHO_SUBJECT**, **WHAT_OBJECT**, **WHO_OBJECT**, **OF** and its subclasses. These two classes have three constituents: the constituent **subject**, representing the name of the attribute to be filled, the constituent **attribute**, representing the value of the attribute and the **verb** *ser* (*be*), linking the **subject** with the **attribute**. In the class **ATTRIBUTIVE_CLAUSE_FILL_ATTRIBUTE** the head of the subject corresponds to the name of the attribute modified by the name of the concept. In the class **REFERENTIAL_ATTRIBUTIVE_CLAUSE_FILL_ATTRIBUTE** the subject is the name of the attribute and the name of the concept does not appear.

The class **ATTRIBUTIVE_CLAUSE_FILL_IS** and the class **REFERENTIAL ATTRIBUTIVE_CLAUSE_FILL_IS** represent the realization of the operations filling attributes in the class **IS**. In the clauses represented by these classes the constituent **subject** is the name of the concept and the constituent **attribute** is the name of the attribute. Operations over the attributes in this class can also be expressed without mentioning the name of the concept in clauses where the subject is elliptic. The range of the attributes in this class is usually the closed set **yes/no**. If it is the case that the value of the attribute is

no, this value is represented as a negation adverb modifying the head of the constituent **attribute**. In the specific case of the value of the attribute being **yes**, then it does not appear in the clause. If the range of the attribute is a close set of values, then the value of the attribute is represented as an adverb or adjective modifying the head of the constituent **attribute**.

The **REFERENTIAL_ATTRIBUTIVE_CLAUSE_FILL_IS_SUBJECT** class represents those clauses expressing the filling of attributes in the class **IS_SUBJECT**. Those attributes are composed by two words, corresponding to the constituent **subject** and **attribute** respectively.

The transitive clauses expressing the filling of one attribute are represented in the two following subclasses:

- **TRANSITIVE_CLAUSE_FILL_ATTRIBUTE**, a subclass of the **TRANSITIVE_CLAUSE**
- **REFERENTIAL_TRANSITIVE_CLAUSE_FILL_ATTRIBUTE**, a subclass of the **REFERENTIAL_ TRANSITIVE_CLAUSE**

These two classes have three constituents: the **subject**, representing the name of concept, the **verb** and the **direct object**, representing the value of the attribute. If the subject is elliptic then these transitive clauses refer to the name of the concept that has appeared in a previous clause.

These classes represent the clauses expressing the filling of attributes in the class **HAS** and attributes in the class **OF_DESCRIPTION** having an associated verb with. When the attribute belongs to the class **HAS** the verb of the clause is *tener* (*have*). When the attribute belongs to the class **OF_NAME**, a subclass **OF_DESCRIPTION**, the verb of the clause is *llamarse*. In Spanish, different verbs are associated with attributes in the class **OF_QUANTITY**, for example *pesar* with *peso* (*weight*) and *medir* with *altura* (*height*).

The subclasses of the **INTRANSITIVE_CLASS** defined to express the filling of one attribute are the following:

- **INTRANSITIVE_CLAUSE_FILL_ATTRIBUTE**
- **REFERENTIAL_INTRANSITIVE_CLAUSE_FILL_ATTRIBUTE**
- **INTRANSITIVE_CLAUSE_FILL_WHO_SUBJECT**
- **REFERENTIAL_INTRANSITIVE_CLAUSE_FILL_WHO_SUBJECT**

- **REFERENTIAL_INTRANSITIVE_CLAUSE_FILL_DOES_SUBJECT**

The class **INTRANSITIVE_CLAUSE_FILL_ATTRIBUTE** and the class **REFERENTIAL_INTRANSITIVE_CLAUSE_FILL_ATTRIBUTE** represent intransitive clauses expressing the filling of one attribute belonging to the class **DOES** and to the classes **OF_TIME**, **OF_PLACE**, **OF_MANNER** and **OF_CAUSE**.

The clauses represented in these two first classes have three constituents: the **subject**, representing the concept and the **verb**, representing the attribute and the **circumstance**, representing the value. Prepositions can also be associated with the attributes. In Spanish, the subject may be elliptic. In this particular case, the clause would referrer to a previous clause describing the concept. And in such a case, the range of the attribute in the class **DOES** is **yes/no**, hence the value **no** is expressed by the adverb *no*, and the value **yes** is not expressed.

An example of attribute belonging to the class **OF_TIME** is the attribute **departure_time** describing the concept **TRAIN**. This attribute is associated with the verb *salir* (*leave*) and the preposition *de* (*at*). The filling of this attribute can be expressed in the clause *El tren sale a las 9* (*The train leaves at 9 o'clock*). Another example of attribute describing the concept **TRAIN** is **destination**. This attribute belongs to the class **OF_PLACE** and is associated with the verb *llegar* (*arrive*) and the preposition *a* (*at*). The clause expressing this filling will be *El tren llega a la estación de Sants* (*The train arrives at Sants Station*).

Intransitive clauses expressing the operation **FILL_ATTRIBUTE_WHO_SUBJECT_O** have only two constituents: the **subject**, representing the value of the attribute and the **verb**, representing either the attribute or the concept. Attributes in this class can also be associated with a verb. An example of attribute belonging to this class is the attribute **driver**, describing the concept **BUS_TRIP**. This attribute is associated with the verb *conducir* (*drive*). The filling of this attribute can be expressed in the referential intransitive clause <*driver-name*> *conduce* (<*driver-name*> *drives*).

There is also a specific class expressing the filling of an attribute in class **DOES_SUBJECT**. The attributes in the class **DOES_SUBJECT** are represented by two words: the first word corresponds to the **subject** and the second word to the **verb**.

The expression in a sentence of more complex operations modifying an instance, such as the operation filling more than one attribute and the operation creating an instance and

filling one or more of its attributes is represented in the class **COORDINATE_CLAUSE_MODIFY**. In this particular case the attributes to be filled belong to the classes **WHO_SUBJECT**, **WHO_OBJECT** and **WHAT_OBJECT**, the filling of these attributes can also be expressed by transitive clauses with indirect objects belonging to the class **INDIRECT_OBJECT_CLAUSE_MODIFY**. An example of this particular case is detailed in Chapter 7.

5.4.1.2 Classes expressing consulting operations

Consulting concepts

There are three classes expressing the two basic operations consulting the existence of a concept.

These classes are:

- **COMPLETE_INTERROGATIVE_CONSULT_CLASS**, a subclass of the class **COMPLETE_INTERROGATIVE**, expressing the operation consulting the existence of a conceptual class
- **COMPLETE_INTERROGATIVE_CONSULT_INSTANCE**, a subclass of the class **COMPLETE_INTERROGATIVE**, expressing the operation consulting if any instance of a conceptual class exist
- **PARTIAL_INTERROGATIVE_CONSULT_CLASS**, a subclass of the class **PARTIAL_INTERROGATIVE** expressing the operation requesting for all members of a conceptual class

The complete interrogative clauses consulting the existence of specific classes and instances of concepts consist of existential clauses where the subject represents the class and the instance respectively. The clauses consulting the existence of instances are the interrogative form of the declarative clauses for the creation of those instances without

giving their name. In Spanish, the only difference is that interrogative clauses are delimited by question marks. For example, the declarative clause for creating an instance of the concept **ARCHITECT**, *existe un arquitecto* (*There is an architect*), is expressed in an interrogative form, *¿Existe un concepto?* (*Is there an architect?*), for consulting its existence.

Consulting attributes

The expression of the operations consulting a conceptual attribute was represented in a similar form of that of operations filling an attribute. Several classes representing the different forms of expressing the operations consulting an attribute were defined considering the basic classes of the attributes. These classes correspond to the interrogative form of the declarative classes for filling the attributes.

The interrogative complete clauses for consulting an attribute class have the same constituents and patterns as the declarative clauses for filling it. In Spanish, the only difference is that interrogative clauses are introduced and finalized by question marks.

The partial interrogative clauses also include all constituents in the declarative clauses for filling an attribute. The patterns, however, differ. In Spanish, the main difference between the declarative and interrogative clauses consists of the interrogative pronoun or adverb introducing the clause as well as the question marks. The interrogative pronoun or adverb is associated with the class of the attribute. The syntactic function is not the same for all interrogative words. The order of the constituents also differs in interrogative clauses.

Complete interrogative clauses express the consulting of attributes having as value the closed set **yes/no**. The attributes belonging to the classes **IS**, **IS_SUBJECT**, **HAS**, **DOES**, **DOES_SUBJECT** may have the set **yes/no** as value. Therefore, the class **COMPLETE_INTERROGATIVE** and the class **REFERENTIAL_COMPLETE INTERROGATIVE** were subclassified to express the consulting of all the attributes having that range.

The resulting classes are:

- COMPLETE_INTERROGATIVE_CONSULT_ATTRIBUTE
- REFERENTIAL_COMPLETE_INTERROGATIVE_CONSULT_ ATTRIBUTE

- REFERENTIAL_COMPLETE_INTERROGATIVE_CONSULT_IS_SUBJECT
- COMPLETE_INTERROGATIVE_CONSULT_HAS
- COMPLETE_INTERROGATIVE_CONSULT_DOES
- REFERENTIAL_COMPLETE_INTERROGATIVE_CONSULT_ DOESSUBJECT

The consulting of an attribute in the class **IS** having as value **yes/no** is supported by the attributive clause for filling this attribute, described above, expressed as a complete interrogative clause. The clause representing the realization of this consulting operation is **COMPLETE_INTERROGATIVE_CONSULT_ATTRIBUTE**. Its constituents are those appearing in the declarative attributive clause for filling the attributes is this class: the **subject** representing the name of the concept, the **verb** *ser/estar* (*be*) and the **attribute** representing the attribute. Those clauses having the subject elliptic are represented by the class **REFERENTIAL_COMPLETE_INTERROGATIVE_CONSULT_ATTRIBUTE**

The consulting of an attribute in the class **IS_SUBJECT** is represented in the class **REFERENTIAL_COMPLETE_INTERROGATIVE_CONSULT_IS_SUBJECT** which represents the consulting form of the class supporting the filling of attributes in this class. The constituent **subject** and the constituent **attribute** of these clauses correspond to the two words associated with the attribute.

The consulting of the attributes in the class **HAS** having as value **yes/no** is supported by the transitive clauses for filling these attributes expressed in a consulting form. These clauses are represented by the class **COMPLETE_INTERROGATIVE_CONSULT HAS**, having as constituents the **subject**, expressing the concept, the verb **has** and the **direct object** expressing the value.

The consulting of the attributes in the class **DOES** and **DOES_SUBJECT** having as value **yes/no** are supported by the intransitive clauses for filling these attributes expressed as complete interrogative clauses. The **COMPLETE_INTERROGATIVE_CONSULT DOES** class has two constituents: the **subject**, expressing the concept, and the **verb**, expressing the attribute.

In the **REFERENTIAL_COMPLETE_INTERROGATIVE_CONSULT_DOES SUBJECT** class the subject and the verb correspond to the two words expressing the attribute.

The consulting of the remaining attributes is expressed by partial interrogative clauses. That is, attributes belonging to the classes **OF**, **WHO_SUBJECT**, **WHAT_OBJECT**, **WHO_OBJECT** and those belonging to the classes **HAS**, **IS** and **DOES** and having a range different of the set **yes/no**. The classes **WHO_SUBJECT**, **WHO_OBJECT** and **OF_PERSON** are associated with the interrogative pronoun *quién* (*who*). This interrogative pronoun always represents the constituent **subject**.

The class **HAS** is associated with the interrogative pronoun *qué* (*what*). This interrogative pronoun, introducing the interrogative transitive clauses asking for the value of the attributes in class **HAS**, represents the constituent **direct_object**.

The classes **WHAT_OBJECT**, **OF** and its subclasses **OF_DESCRIPTION** and **OF_TYPE** are associated with the interrogative pronoun *cuál* (*which*). The consulting of the value of the attributes in these classes is expressed by partial interrogative attributive clauses where the pronoun *which* represents the constituent **attribute**.

The classes **OF_MANNER** as well as the classes **DOES** and **IS,** having a different range of **yes/no,** are associated with the interrogative adverb *cómo* (*how*). The class **OF_PLACE** is associated with the adverb *dónde* (*where*). The class **OF_TIME** is associated with the adverb *cuándo* (*when*). The class **OF_CAUSE** is associated with the adverb *por qué* (*why*). The value of the attributes in these classes is expressed by partial interrogative intransitive clauses. The function of the interrogative adverbs introducing these clauses is to describe the circumstances of the action.

Finally, the class **OF_QUANTITY** is associated with the adverbs *cuánto/cuántos* (*how much/how many*). The consulting of attributes in this class is expressed by partial interrogative transitive clauses where the adverb represents the constituent **direct_object**.

New linguistic classes, not considered in the current design, could also be easily incorporated to cover more complex operations. For example, to express the operation to fill the attribute of a conceptual instance when this instance is not expressed by its name, but by the value of one of its attributes, the function **subordination** will have to be incorporated in the LO general level.

5.4.2 The GROUP class

The expression of the arguments of the operations is represented in the classes belonging to the rank **group** and **word**. Subclasses representing all types of arguments in the operations were distinguished.

At group level, subclasses of the **NOMINAL_LIKE_GROUP**, **VERBAL_GROUP** and **ADVERBIAL_GROUP** classes were defined to express concepts, attributes and values.

Nominal groups expressing the name of concepts are represented in the classes:

- **DEFINITE_NOMINAL_GROUP_CONCEPT**
- **INDEFINITE_NOMINAL_GROUP_CONCEPT**
- **INDIRECT_NOMINAL_GROUP_CONCEPT**

Nominal groups expressing the name introduced by the user to identify a conceptual instance are described in the following classes:

- **DEFINITE_NOMINAL_GROUP_INSTANCE**
- **INDEFINITE_NOMINAL_GROUP_ INSTANCE**
- **INDIRECT_NOMINAL_GROUP_ INSTANCE**

Nominal groups representing the identifiers of all instances of a conceptual class are represented in the classes **PROPER_NOMINAL_GROUP_INSTANCE** and **INDIRECT_PROPER_ NOMINAL_GROUP_INSTANCE**.

Nominal groups expressing the name of conceptual attributes are represented in the classes:

- **DEFINITE_NOMINAL_GROUP_ATTRIBUTE**
- **INDEFINITE_NOMINAL_GROUP_ATTRIBUTE**
- **INDIRECT_NOMINAL_GROUP_ATTRIBUTE**

Nominal groups expressing the values of the attributes are represented in the classes:

- **DEFINITE_NOMINAL_GROUP_VALUE**
- **INDEFINITE_NOMINAL_GROUP_VALUE**
- **INDIRECT_NOMINAL_GROUP_VALUE**

The class **ADJECTIVAL_GROUP** was further subclassified in order to represent attributes as well as values. The resulting classes are:

- **ADJECTIVAL_GROUP_ATTRIBUTTE**
- **ADJECTIVAL_GROUP_VALUE**

A subclass of the **QUANTITY_GROUP** class was distinguished, the class **QUANTITY_GROUP_VALUE**, representing the values of those conceptual attributes expressing quantities.

The class **VERBAL_GROUP** was further subclassified into the two following classes:

- **VERBAL_GROUP_CONCEPT**
- **VERBAL_GROUP_ATTRIBUTE**

The class **ADVERBIAL_GROUP** was further subclassified into the two following classes:

- **ADVERBIAL_GROUP_ ATTRIBUTE**
- **ADVERBIAL_GROUP_VALUE**

5.4.3 The WORD class

At word level, the **NOUN**, **VERB**, **ADJECTIVE** and **ADVERB** classes expressing concepts, attributes and values were subdivided.

The class **PROPER_NOUN_INSTANCE**, a subclass of the **PROPER_NOUN** class, was defined to represent conceptual instances.

The class **COMMON_NOUN** was subdivided in the following classes:

- **COMMON_NOUN_CONCEPT**
- **COMMON_NOUN_ATTRIBUTE**
- **COMMON_NOUN_VALUE**

The class **TRANSITIVE_VERB** and the class **INTRANSITIVE_VERB** were subdivided into the classes:

- **TRANSITIVE_VERB_CONCEPT**

- **TRANSITVIE_VERB_ATTRIBUTE**
- **INTRANSITIVE_VERB_CONCEPT**
- **INTRANSITVIE_VERB_ATTRIBUTE**

The class **DESCRIPTIVE_ADJECTIVE** was subdivided into the classes:

- **DESCRIPTIVE_ADJECTIVE_ATTRIBUTE**
- **DESCRIPTIVE_ADJECTIVE_VALUE**

The class **CONCEPTUAL_ADVERB** was subdivided in:

- **CONCEPTUAL_ADVERB_ATTRIBUTE**
- **CONCEPTUAL_ADVERB_VALUE**

The process of obtaining the linguistic structures required for an specific application from the LO domain level is detailed in Chapter 6.

5.5 INFORMATION REPRESENTED IN THE LINGUISTIC CLASSES

In the LO, information describing a class is represented as a set of descriptors or facets. The LO allows the same type of inheritance as the CO. The facets described in the general classes are inherited by all subclasses. The same default mechanism is valid here. Orthogonal multiple inheritance is also allowed; more than one dimension can be considered when defining linguistic classes.

The information describing the linguistic structures in the LO consists of general linguistic features describing the classes as well as features giving detailed information about their constituents. The ideational, interpersonal and textual functions describing a class are represented as a set of facets. These facets are **simple**, **reference**, **ellipsis**, **polarity**, **mood**, **voice** and **theme** at clause level and **simple**, **reference**, **type**, **person** and **determination** at

group level. The range of these facets is a closed set of values. Specific linguistic information about constituents of the linguistic structures is also represented as facets.

As mentioned before, the specific application-restricted linguistic structures are obtained from those represented in the LO. Although for reasons of efficiency the linguistic structures generated for a specific application are represented in a different formalism, a definite context grammar (DCG), all information expressed by this formalism can also be expressed in the LO. All information appearing in DCG rules (the left-hand side category, representing the linguistic structure, the right-hand side categories, representing its constituents, the semantic and syntactic features associated with these categories as well as all possible presentations of these constituents) are represented in the LO as facets describing the classes.

Linguistic classes may have one or more than one constituent. Linguistic objects belonging to the class **MAJOR_CLAUSE** have more than one constituent. Most of the objects in the classes **MINOR_CLAUSE** and **GROUP** have more than one constituent, although there are also elements in these classes having only one constituent. Linguistic objects belonging to the class **WORD** have only one constituent.

When generating the grammar, the objects having more than one constituent are represented as non-terminal categories, those that appear in the left-hand of a grammar rule. The LO objects having only one constituent are represented as terminal categories in the generated grammar.

The information describing the LO objects having more than one constituent is not the same than that describing the objects having only one constituent. The facets needed to describe all LO objects are detailed below.

5.5.1 LO objects having more than one constituent

All linguistic objects have an associated linguistic category, described in the facet **category**. Each constituent of the class is represented by a facet whose value corresponds to an existing linguistic object. As in lexico-functional grammars, the sequence of the syntactic constituents that occur in each class is represented in the constituent set,

represented by the facet **cset**, and information on the superficial presentations of class by means of a different set, represented by the facet **pattern**.

The value of the facet **pattern** is the set of all presentations allowed for the constituents of a class. A list of numbers indicating the order of interpretation of the constituents is associated with each possible pattern. This list of numbers represents information for further semantic interpretation. The semantics is based on lambda calculus.

If the linguistic category representing the class is augmented with syntactic features, those are represented in the facet **synfeatures**. The syntactic agreement between the class constituents is represented by the facet **synagreement**. The value of this attribute is a list in which the constituents are associated with their syntactic features. Only constituents having associated linguistic features are represented in the list.

The description of the most general classes, **CLAUSE** and its two subclasses, **MAJOR_CLAUSE** and **MINOR_CLAUSE** is shown in Figure 5.4.

The metafunction information associated with the **MAJOR_CLAUSE** class is represented in the facets **polarity**, **mood**, **reference**, **theme**, **simple**, **voice** and **ellipsis**. The first four facets and their value are inherited fom the upper class **CLAUSE**. The facet **polarity** indicates that all clauses can be expressed both in a positive and negative form. The value of the facet **mood** is **declarative /interrogative / imperative**, indicating that the modality of the clauses can be declarative, interrogative and imperative. The facet **reference** indicates whether or not there is a reference in the clause to a constituent appearing in a previous clause. Its value can be **yes** or **no**. The facet **theme** indicates the theme expressed in a clause. In the LO domain level objects, the value of this facet always corresponds to the identifier of an operation.

The facets **simple**, **ellipsis** and **voice** are defined only for major clauses. The value of the facet **simple** is **yes/no**, indicating that the major clauses can be simple or complex. The **voice** indicates that elements in this clause can be presented both in an active and in a passive voice. The facet **ellipsis** indicates whether the subject of the clause is elliptic or not.

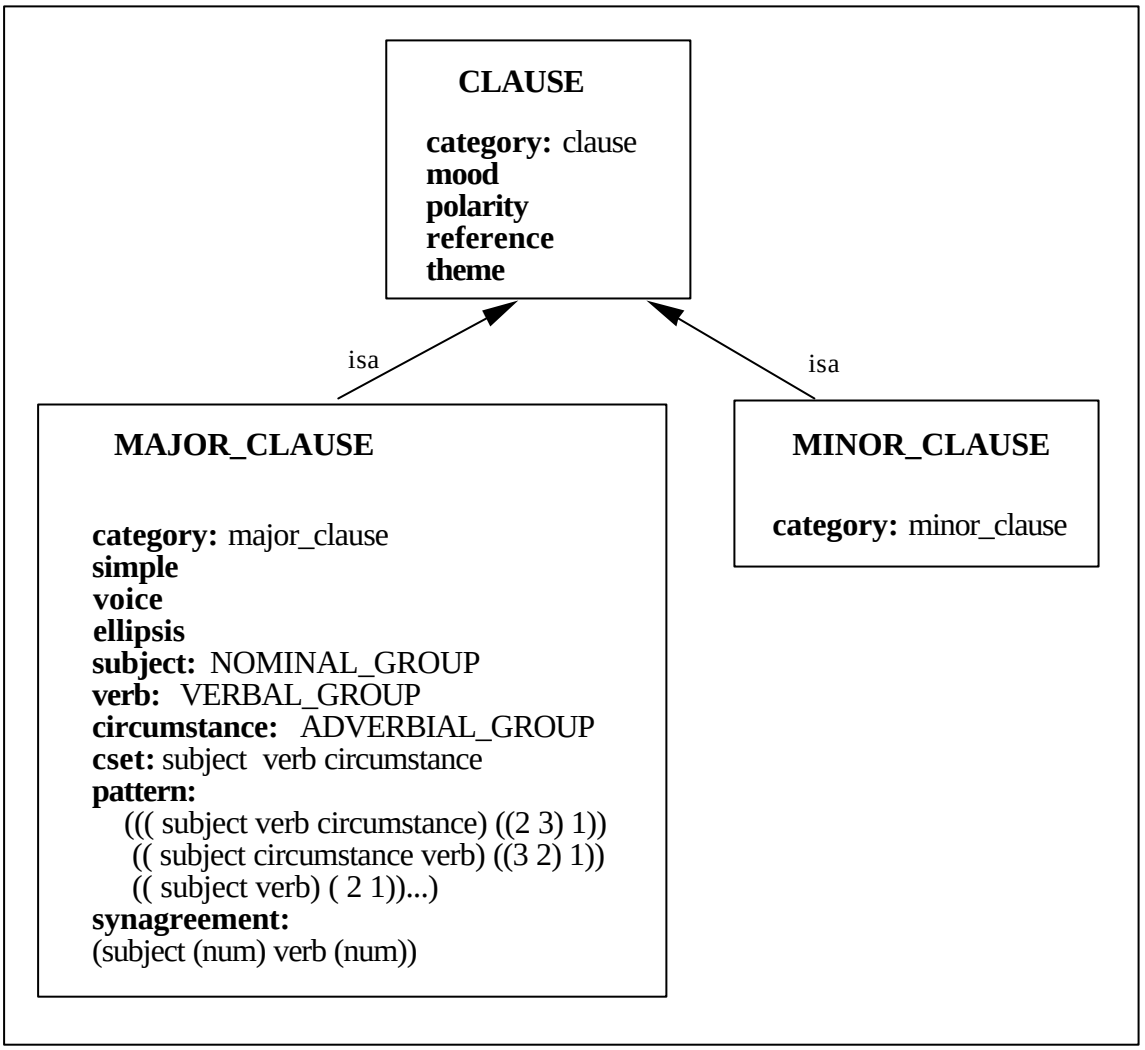

Figure 5.4: The representation of the classes: CLAUSE, MAJOR_CLAUSE and MINOR_CLAUSE

Information about the constituents of the major clauses is represented in the rest of the facets. The facet **cset** of the class **MAJOR_CLAUSE** represents the constituents of the major clauses considered in the LO design. These constituents are **subject**, **circumstance** and **verb.** There is a facet representing each of these constituents, its value being the name of a LO object. The constituent subject must be a member of the class **NOMINAL_GROUP**. The constituent verb is an element in the class **VERBAL_GROUP**. The constituent circumstance belongs to the class **ADVERBIAL_GROUP**. All these constituents belong to the class **GROUP**.

The facet **synagreement** indicates that the value of the feature **num** (number) of the constituent **subject** and that of the constituent **verb** must be the same.

When representing a LO object having more than one constituent following the DCG formalism, each possible representation of the constituents will correspond to a different grammar rule. The category associated with the LO object will correspond to the left-hand part of the rules and the constituents to the right-hand part. Each category in the resulting grammar can be augmented with syntactic and semantic features. The syntactic features associated with the category will be obtained from the facet **synfeature**, incorporated, when necessary in the LO object description. The semantic features will be obtained from the facet **semfeature**.

The name of each feature is represented as a *functor* associated with the category and its value in brackets. If the value is not defined is represented as a variable (i.e. the object **NOMINAL_GROUP** will be represented by the category **nominal_group(syn(gen(G),num(N)))** where **gen(G)** represents the feature gender and **num(N)** the number). During the analysis this variable will be instantiated with a specific value. A feature agreement between two or more constituents is represented by the same value (or the same variable representing the value) associated with their *functors* representing the corresponding feature. For example, the number agreement between the constituents **subject** and **verb** in a grammar rule will be represented by using the same variable associated with the functor **num**, in the categories representing the two constituents.

Figure 5.5 shows how information describing the **MAJOR_CLAUSE** class is inherited by its two subclasses: the **ATTRIBUTIVE_CLAUSE** and the **PREDICATIVE_CLAUSE** classes. The **ATTRIBUTIVE_CLAUSE** class represents a subset of major clauses: the attributive clauses. In this class, the information inherited from the **MAJOR_CLAUSE** class is restricted. Only attributes adding new information are incorporated into the description of this class. These facets are **category**, **voice**, **verb**, **attribute**, **cset**, **pattern** and **synagreement**.

The facet **voice** indicates that attributive clauses can only be expressed in an active form. The facet **cset** defines the four possible constituents of these clauses: subject, verb, attribute and circumstances. The facet **pattern** represents the possible superficial presentation of these constituents. There is no additional information describing the subject and the circumstances in attributive clauses. The verb belongs to the class **COPULATIVE_VERB**. The attribute belongs to the class **NOMINAL_LIKE_GROUP**. The syntactic agreement between these constituents indicates that, as in all major clauses,

the number and person of the subject and the verb must be the same. It also indicates that the number and gender of the subject and attribute must be the same.

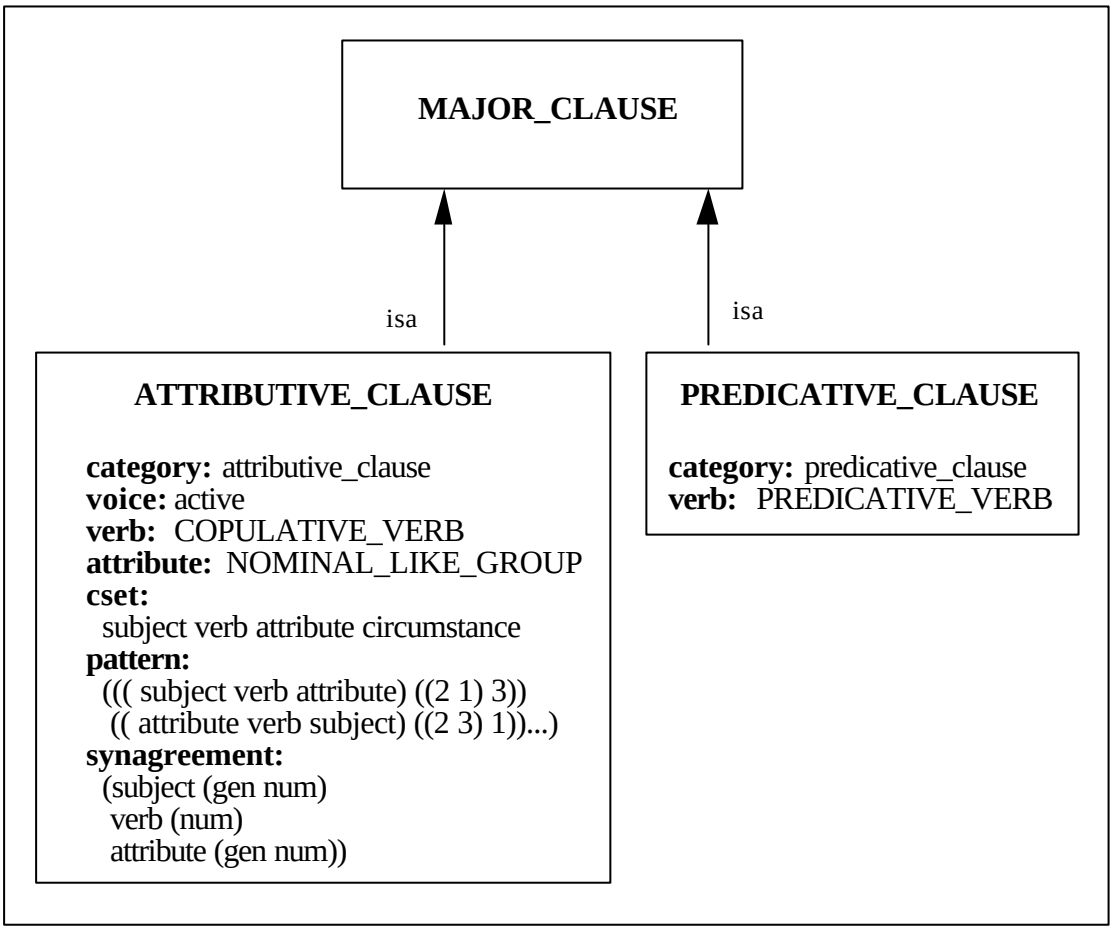

Figure 5.5: The classes ATTRIBUTIVE_CLAUSE and the PREDICATIVE_CLAUSE

Only two facets, the facet **category** and the facet **verb** describe the **PREDICATIVE_CLAUSE** class.

In the LO domain level, the description of the classes incorporates linguistic and conceptual information. The conceptual information about the class and its constituents is represented in its categories and the conceptual features associated with them.

The category associated with the class is an abbreviation of the name of the class. The names of the LO classes incorporate conceptual information about the operations or the arguments expressed. For example, the category **ec_cinn** is the abbreviation of the name of the class **EXISTENTIAL_CLAUSE_CREATE_INSTANCE_WITH_NO_NAME**,

described in Figure 5.6. The category **indefngcon** is associated with the class **INDEFINITE_NOMINAL_GROUP_CONCEPT**, represented in Figure 5.9.

The semantic features associated with the category representing the class are described by the facet **semfeature**. These features represent the concept and/or the attribute appearing in the linguistic class.

The agreement between the semantic features associated with the class constituents is described by the facet **semagreement**. This attribute is represented as a list where each constituent is associated with its semantic features.

Classes expressing operations incorporate the name of the operation in the facet **theme**. They also incorporate the facet **pcc**, representing the case preconditions associated with it.

An example of a class belonging to the domain level is described in Figure 5.6. The **EXISTENTIAL_CLAUSE_CREATE_INSTANCE_WITH_NO_NAME** class shown in the figure represents simple and declarative existential clauses expressing the operation of creating an instance of a CO concept, without giving its name. It is a subclass of the **SIMPLE_DECLARATIVE_EXISTENTIAL_CLAUSE** class, which is a **EXISTENTIAL_CLAUSE** subclass.

Only the facets adding new information to the upper class are incorporated into the description of the class. In this example, these facets are **category**, **polarity**, **reference, simple**, **subject**, **verb**, **cset** and **pattern**, giving linguistic information and **semfeature**, **semagreement**, **theme** and **pcc**, giving conceptual information.

The value of the category associated with this class is **ec_cinn**. The constituent set is **subject** and **verb**. The constituent subject belongs to the class **INDEFINITE_NOMINAL_GROUP_CONCEPT**. The constituent **verb** belongs to the class **VERB_EXISTIR**, a subclass of **EXISTENTIAL_VERB**. The most natural presentation of these two constituents is represented by the facet **pattern**.

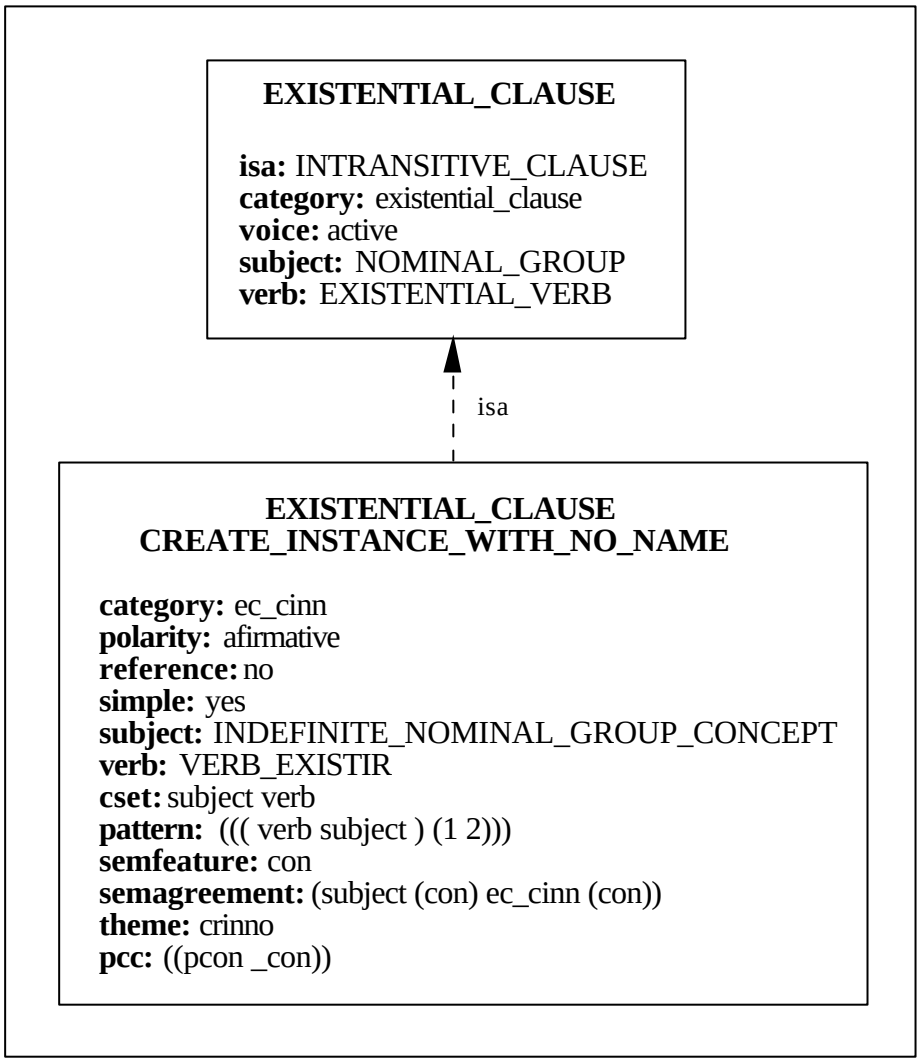

Figure 5.6: The class EXISTENTIAL_CLAUSE_CREATE_INSTANCE_WITH_NO_NAME

The facet **semfeature** represents the semantic features associated with the class, the feature **con**. When representing this object as a grammar rule, the category in the left-hand part of the rule will be **ec_cinn** and will be augmented with the feature **con**. This feature will be represented as the *functor* **con** and a value in brackets. The value associated with the *functor* **con** will represent the parameter **con** (corresponding to the concept) of the operation (i.e. **ec_cinn(sem(con(architect)))**).

The facet **semagreement** represents the semantic agreement between the constituents in the class. It indicates that the value associated with the functor **con** in the category representing the class and in that representing the constituent **subject** must be the same.

The facet **theme** represents the abbreviation of the name of the operation, **crinno**. The facet **pcc** represents the case preconditions associated with the operation, indicating the preconditions that must be satisfied to create a conceptual instance.

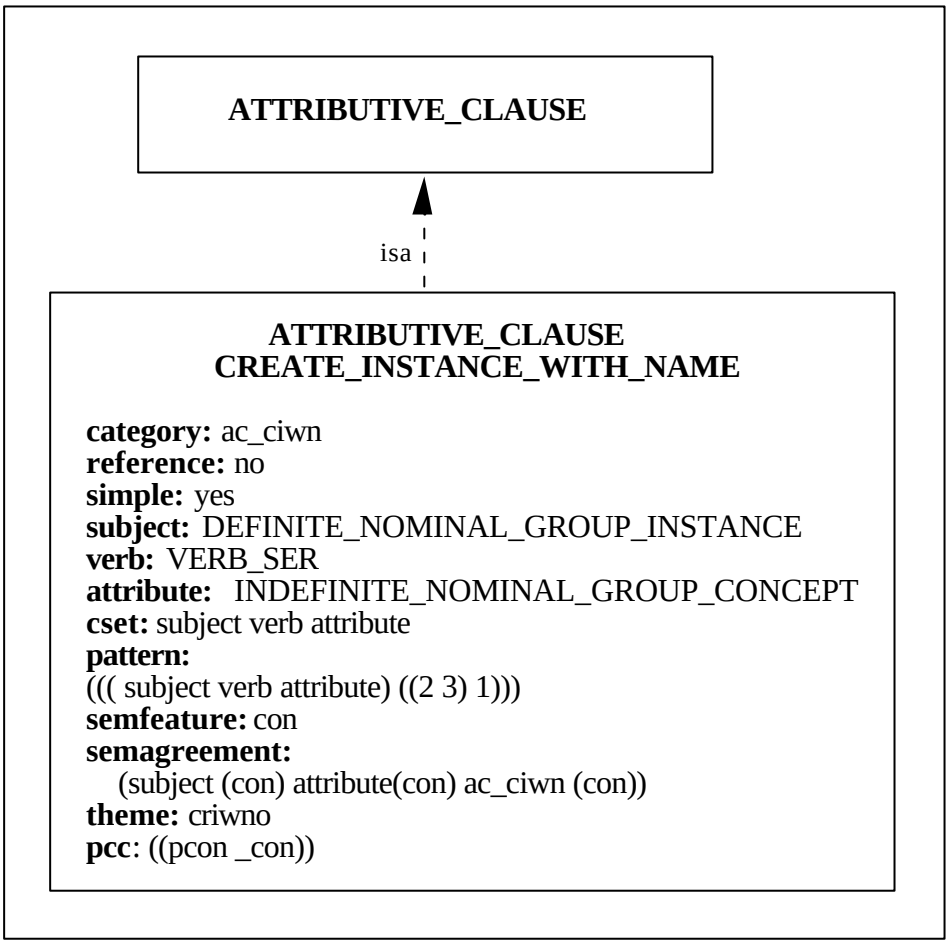

Figure 5.7: The class ATTRIBUTIVE_CLAUSE_CREATE_INSTANCE_WITH_NAME

Figure 5.7 shows the description of the **ATTRIBUTIVE_CLAUSE** class and one of its subclasses, the **ATTRIBUTIVE_CLAUSE_CREATE_INSTANCE_WITH_NAME** class. This class represents all simple declarative clauses expressing the operation of creation a conceptual instance giving its name.

The facets adding new linguistic information to those facets inherited from its upper classes were incorporated into the class description. These facets are: **category**, **reference**, **simple**, **cset**, **subject**, **verb**, **attribute** and **pattern.** Conceptual information is represented by the facets: **semfeature**, **semagreement**, **theme** and **pcc**.

The value of the category associated with this class is **ac_ciwn**. The constituent set consists of the subject, verb and attribute. The constituent **subject** is an element in the class **DEFINITE_NOMINAL_GROUP_INSTANCE**, representing conceptual instances which name is introduced by the user. The constituent subject is inherited from the upper class. The constituent **verb** corresponds to the verb *ser* (*be*), represented by the class **VERB_SER**. It is a subclass of the **COPULATIVE_VERB** class. The constituent **attribute** is a member of the class **INDEFINITE_NOMINAL_GROUP_CONCEPT**. The only possible presentation of these constituents is described by the facet **pattern**.

The facet **semfeature** represents the semantic feature associated with the class, **con**. In the grammar generated, the category representing this LO object, **ac_ciwn**, will be augmented with the feature **con**. The value associated with this value represents the parameter **con** of the operation.

The facet **semagreement** represents the semantic agreement between the constituents in the class. It indicates that the value of the feature **con** associated with the class, the value of the feature **con** associated with the constituent **subject** and that associated with the constituent **attribute** must be the same.

The facet **theme** represents the name of the operation. The facet **pcc** represents the case preconditions associated with the operation.

The **SIMPLE_COMMON_NOMINAL_GROUP** class, the **DEFINITE_SIMPLE COMMON_NOMINAL_GROUP** class and the **INDEFINITE_SIMPLE COMMON_NOMINAL_GROUP** class are described in Figure 5.8.

The **SIMPLE_COMMON_NOMINAL_GROUP** class is a subclass of the **COMMON_NOMINAL_GROUP** class. As can be seen in the figure, this class is described by the facets giving information about the class and facets describing its constituents. The facets giving information about the class are **category**, **reference**, **simple**, **determination**, **type**, and **synfeature**.

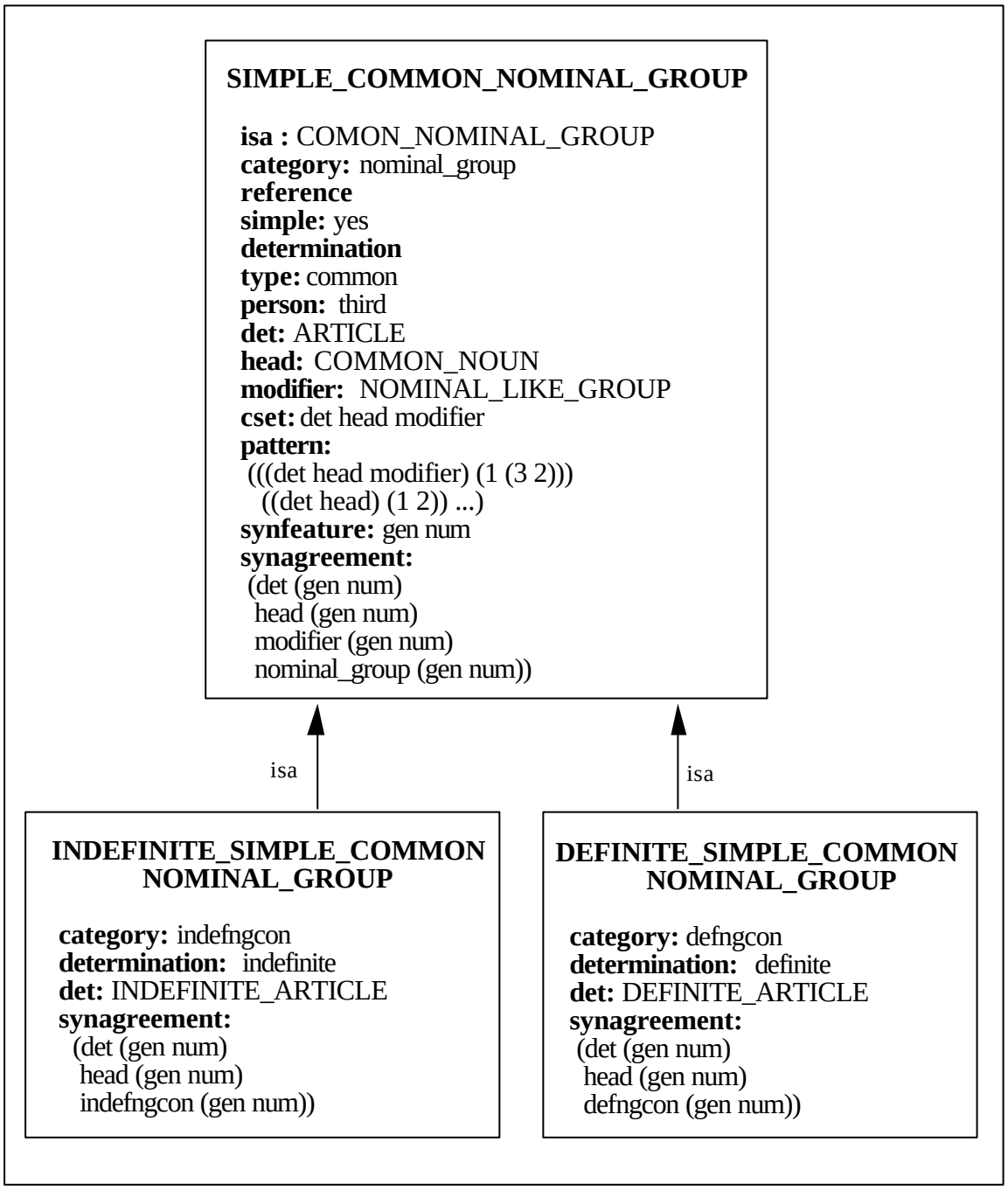

Figure 5.8: The SIMPLE_COMMON_NOMINAL_GROUP class and its subclasses

The facets describing the constituents are **cset**, **det**, **head**, **modifier**, **pattern** and **synagreement**. The constituent set of the class is **det** (determiner), **head** and **modifier**. The facet **det** represents the class **ARTICLE**. The facet **head** represents the class **COMMON_NOUN**. The **modifier** is an object belonging to the class **NOMINAL_LIKE_GROUP**.

The possible superficial representations of the simple common nominal groups are **<det head modifier>**, **<det modifier head>** and **<det head>**. These three representations, together with the list of numbers indicating the constituent order interpretation for each, are represented in the facet **pattern**.

The facet **synfeature** represents the syntactic features associated with the class; in Spanish the gender (**gen**) and the number (**num**). The facet **synagreement** indicates that the value of these features in the nominal group and its constituents **det**, **head** and **modifier** must be the same.

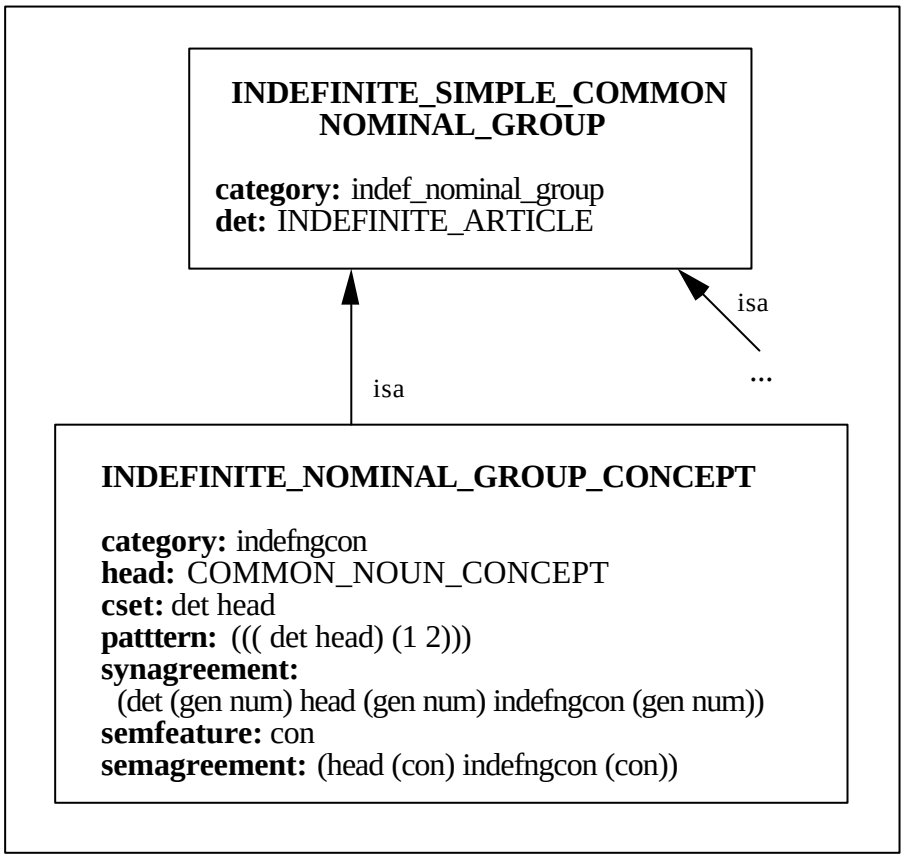

Figure 5.9: The INDEFINITE_NOMINAL_GROUP_CONCEPT class

The **DEFINITE_SIMPLE_COMMON_NOMINAL_GROUP** class represents those nominal groups having the definite article as constituent **det**. The **INDEFINITE SIMPLE_COMMON_NOMINAL_GROUP** class represents those nominal groups having introduced by an indefinite article. One example of its subclasses in the LO domain level is the **INDEFINITE_NOMINAL_GROUP_CONCEPT** class, described in Figure

5.9. The two only constituents of this class are described by the facet **cset**. The facet **pattern** describes its allowed presentation. The facet **synfeature** the syntactic agreement between the constituent **det** and **head**. The facet **semfeature** represents the semantic feature associated with the class, **con** (concept). The facet **semagreement** indicates that the feature **con,** associated with the constituent **head** and with the class, must be the same.

For each interface, an instance of this class is created from each concept in the CO that can be realized as a nominal group.

5.5.2 The LO objects having one constituent

Linguistic objects having only one constituent are further classified as **closed** and **open**. Closed classes represent a closed set of words that are the same for all applications. Examples of these classes are the class **ARTICLE** and the class **PREPOSITION**. Open classes, such as the class **NOUN**, represent a wide range of words, different for each application.

The description of classes having only one constituent consists of category, the syntactic features, and the semantic information. The linguistic category is represented in the facet **category**. If syntactic features are associated with the object, these are represented in the facet **synfeature**. The semantic features are represented in the facet **semfeature**. The facet **sem** represents semantic interpretation.

The semantic interpretation of objects having one constituent consists of a lambda function or value. Lambda functions are represented by a list, in which the first part defines the parameters the function has, and the second part defines the value to be returned. The semantic interpretation associated with closed classes consists of a lambda function giving information about how the other constituents in the clause should be linked. In open classes, semantic interpretation is a function related to the CO (i.e., the name of the concept represented by the linguistic instance).

Words are represented in the LO as *terminal* objects belonging to the classes having only one constituent. The words belonging to the closed classes are represented in the LO. These objects represent the lexical entries common to all applications. The words representing application terms belong to the open classes, such as **NOUN**, **ADJECTIVE**

and **VERB**. These objects are generated for each application. They are also represented as LO terminal objects.

In terminal objects, the features and their value are incorporated into the category. An additional facet, **type**, is incorporated in the definition of terminal objects representing the application terms. This facet indicates if the term is associated with one word (or more) or with a dynamic function. The value **lex** indicates the object is associated with a word. The value **dyn** indicates it is associated with a function asking the user to introduce a value at run-time.

An example of a terminal object is that representing the adverb *no*, described in Figure 5.10. This object belongs to the class **NEGATION_ADVERB**, a subclass of the **CLAUSE_ADVERB** class. It is a closed class having only one constituent. Its category is **no**. Its semantic interpretation is the lambda function: **((l _x) (no _x))**. The first part of the function (**(l _x)**) indicates that it has one parameter; the second part (**(no _x)**) indicates that this function returns a list with the value **no**, followed by the value of the parameter. The string associated with this object is *no*.

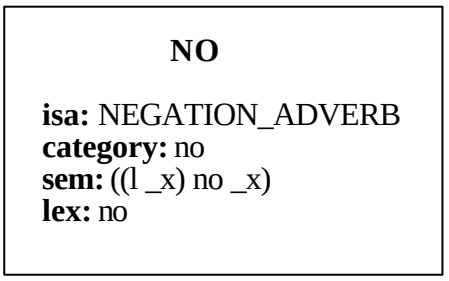

Figure 5.10: The representation of the adverb no

A new example of class having one constituent is described in Figure 5.11. This figure describes the class **DEFINITE_ARTICLE**, belonging to the class **ARTICLE** and two of its four members, the articles *el* and *la*. The category of the class is **defart**. The semantic interpretation of all members in the class is the lambda function **((l _x) (_x))**, indicating that definite articles are functions with a parameter and that these functions do not modify the semantic value of the parameter. The syntactic features associated with the class are number and gender. The articles *el* and *la* are represented as terminal objects, subclasses of the **DEFINITE_ARTICLE** class. The features **gen** and **num** and their values were incorporated into the category.

This description also includes the superficial presentation, represented in the facet **lex**.

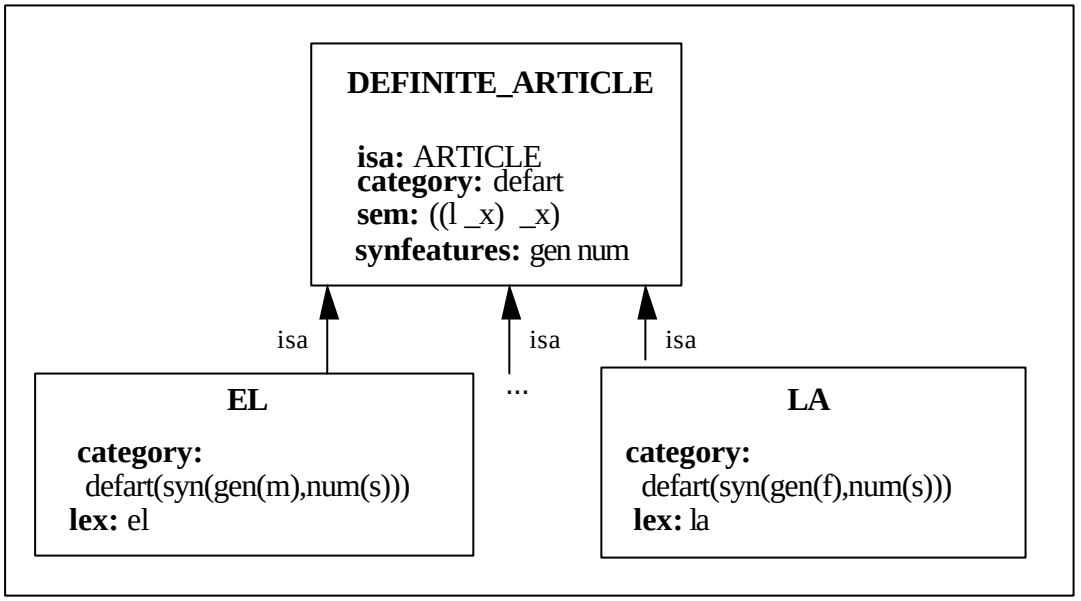

Figure 5.11: The representation of the DEFINITE_ARTICLE class and two of its members

CHAPTER 6

THE CONTROL RULES

6.1 INTRODUCTION

In this work, general control rules performing the creation, modification and consulting of ontology concepts have been designed. These rules make it possible to establish general relations between concepts in different ontologies as well as between concepts in the same ontology. The control rules are of the form:

conditions -> actions

They are applied on the objects in the ontology satisfying the conditions. Because they can be easily adapted to relate to different types of knowledge represented in one or more ontologies, they can be used for several purposes and applications. For example, they could be used to perform the integration of various ontologies or the creation of a specialized ontology from a general ontology. In this work, however, the control rules are used to perform the mapping of the CO onto the LO.

The control rules represent the relations between the concepts and operations represented in the CO as well as the relations between the CO objects and the linguistic structures represented in the LO. Representing this knowledge separately from the CO and LO allows the easy adaptation of the design proposed in this thesis to new domains and applications.

The control rules are responsible for adapting the general conceptual and linguistic knowledge to different applications. The general relations established by the control rules are intended to be application independent. A set of rules defining basic relations between the application conceptual knowledge and its specific realization has been reused for different types of applications.

The control rules perform the process of obtaining the linguistic resources necessary in the application-restricted interfaces by relating the application knowledge represented in the CO to the general linguistic knowledge represented in the LO. First, application concepts are related to the CO operations representing the general communication tasks. Then, the application specific communication tasks are related to the linguistic structures in the LO supporting their expression.

This chapter has been organized in five sections. The general process of relating the application representation in the CO to the LO in order to obtain the application-restricted grammar is described in second section. The formalism of the control rules ensuring this process is defined in the next section. The basic set of control rules designed and implemented for generating the interfaces for a broad type of applications is detailed in the fourth section. Finally, in the last section, an example following the performance of this basic set of control rules is detailed.

6.2 RELATING THE CO TO THE LO IN ORDER TO OBTAIN APPLICATION RESTRICTED INTERFACES

As described in Chapter 3, application knowledge is represented in the CO and general linguistic knowledge necessary to express this knowledge is represented in the LO. The lexical information associated with the terms appearing in an application is represented in a set. This information consists of a set of surface realization associated with a term. Each linguistic realization is associated with a syntactic category and its syntactic features. Information in the CO, LO and application terms set has to be related in order to obtain the

linguistic resources of the application-restricted interfaces. One of the main goals in the design of the CO and the LO was to facilitate the performance of this generation task.

In the CO, pointers to the set containing all possible linguistic realizations were incorporated into the descriptions of the concepts and attributes. A taxonomy of the attributes linguistically motivated was established. In the LO domain level, general conceptual information was incorporated into the description of the linguistic classes (the name of the category, the semantic features and the preconditions). However, in order to assure a separate design, implementation and update of the CO and the LO, the specific relations between the two ontologies have been defined in a separate knowledge base. This knowledge base consists of a set of control rules ensuring the CO mapping onto the LO. By relating the CO to the LO, these rules control the process of obtaining the linguistic structures necessary in application-restricted interfaces.

Control rules have been chosen to express conceptual and linguistic relations because of their clarity and explicitness. They are a powerful and flexible way to express different relations between ontology objects. The control rules are applied over the objects in the CO and the LO, satisfying the conditions. The actions performed by the rules are operations consulting and modifying objects in the CO and LO. The control rules establish relations at three different levels: relations between objects in the CO, relations between CO and LO objects and relations between LO objects.

Different control rules can establish different relations between ontology objects. Linguistic coverage can be as broad as necessary. Current LO, through the performance of the appropriate control rules, makes it possible to generate grammars covering ellipsis, subordinate clauses, a variety of anaphoric references, etc.

Different control rules can be built to encode the conceptual information representing the application functionality in the grammar rules and lexical categories, following different criteria. Rules for different types of user (casual or habitual, expert or novice) can also be incorporated.

A basic set of control rules has been built defining the conceptual and linguistic relations for a broad set of applications. These relations are intended to be domain and application-independent. Although these relations between conceptual and linguistic information are the same for different applications; they also cover details about the linguistic realization of each application concept. New relations can be incorporated by considering different

linguistic details. The basic set of control rules is responsible for generating the grammar rules to express, in a natural way, operations over the CO for different applications. Several linguistic realizations for the same operation were considered.

The proposed design allows the extension of the CO and the LO in order to enrich the linguistic and conceptual coverage without modifying the control rules. However, it is also possible to adapt the proposed design to different applications by modifying the basic set of rules.

The basic set of rules performs the process of obtaining the linguistic structures required for an application in two steps. The first step consists of relating the general communication tasks represented in the CO to the specific application concepts. The second step consists of relating the application specific communication tasks to the LO classes representing its linguistic expression. Next, the resulting linguistic structures are translated into DCG.

In the first step, a set of rules ensures the process of adapting the CO operations representing the general communication tasks to the concepts modeling a specific application. For each concept representing the application, instances of operations creating, modifying and consulting it are generated.

In the second step of the process, a different set of control rules ensure the relating of operations generated in the first step to the LO. In this step, LO instances covering the expression of the application operations are created. As described in Chapter 5, the LO classes are assumed to be common to all applications, while linguistic objects representing the specific aspects of the information to be expressed for each application are represented as instances of the LO classes. When obtaining the appropriate linguistic objects for an application, instances of the linguistic classes in the rank **clause**, **group** and **word** are generated. Instances in the rank **clause** are created to express the operation instances generated in the first step. Instances of the rank **group** and **word** are created to express the arguments of these operations. When generating these instances, the class of the operation, the class of the CO concept and the class of the concept attributes involved in the operation are considered.

Finally, there is a third step consisting of generating the grammar rules and lexical entries necessary for an interface. Basically, this step consists of writing the LO instances generated in the second step as a DCG.

A more detailed description of the performance of the basic set of control rules is given in Section 6.4.

6.3 THE FORMALISM OF THE CONTROL RULES

Control rules are implemented in the Production Rules Environment (PRE), a rule-based environment specially built for NL, described in [Ageno93]. PRE incorporates the capabilities necessary for control rules performance: the use of rulesets, a powerful and flexible control mechanism and a dynamic data storage device, the working memory (WM), where objects can be created, consulted and modified.

The function of the WM is that of a blackboard where objects can be modified and consulted efficiently (more about blackboard architecture can be found in [Engelmore88]). In the WM, partial and complete descriptions of objects of the CO and LO are represented. A word or name representing the type of the object and a list of attribute-value pairs describe all objects in the WM. Each pair attribute-value is represented by the attribute followed by a value. The attribute is represented by the character ^ followed by the attribute identifier. A variable, a name or a list of variables and/or names can represent the value. The symbol **?** precedes the names of variables. Variables in the WM can be matched by any value.

An example of a WM object is shown in Figure 6.1. The object in this figure represents a partial description of a particular concept in the CO. The descriptions of this concept consists of the word **object**, the attribute ^**con** and the attribute ^**pcc**. The value of the attribute ^**con** is **obra**, representing the identifier of the concept. The attribute ^**pcc** represents the case preconditions associated with the concept, its value **nil** indicates that there are no preconditions associated with the concept **obra**.

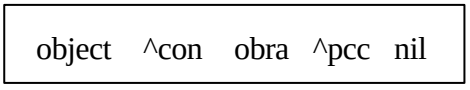

*Figure 6.1: The WM object representing a partial description of the concept **OBRA***

The general WM object describing the identifier and preconditions of CO concepts is represented in Figure 6.2. The value of the attribute ^**con** and ^**pcc** are represented by the variables **?con** and **?pcc**, respectively.

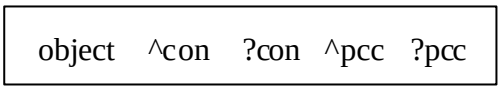

Figure 6.2: The WM object representing a partial description of the CO concepts

There are basic operations managing the WM objects for creating, deleting, consulting and modifying WM objects.

There are two operations for creating WM objects. The operation **create-wm** creates one WM object represented by a name (or the identifier of the type of the object) and an optional list of pair attribute-value. The operation **create-all-wm** creates a set of objects (one or more) having the same type and attributes, but different attribute values.

There are three operations for deleting WM objects. The operation **delete-wm** deletes the first object in the WM matching a specific definition (a name and an optional list of attribute-value pairs). The operation **delete-all-wm** deletes all objects in the WM having the name specified in the operation call. The operation **empty-wm** deletes all objects in the WM.

There are two operations for consulting WM objets. The operation **get-wm** obtains the first object in the WM matching a specific definition (a name and an optional list of attribute-value pairs). The operation **get-all-wm** returns all WM objects having a specific name.

There is also an operation for modifying WM objects, this is the operation **modify-wm**, changing the attribute-value pairs of a WM object matching a specific definition.

In PRE, rules are grouped into rulesets. Each ruleset has a variable number of rules. Two basic operations ensure the activation of rulesets: **apply-ruleset** and **apply-ruleset-top**. The operation **apply-ruleset** activates a ruleset given its name. The operation **apply-ruleset-top** activates the TOP ruleset, described below.

The programs responsible for applying the control rules can perform the basic operations managing the WM objects and activating the rulesets. The actions performed by the rules

are higher operations over these basic operations. The syntax of the basic operations is described below in Figure 6.3.

```
< WM-operations >  ::=
            < create-operation > / < delete-operation > /  < consult-operation > /
            < modify-operation > / < apply-ruleset-operation >
< create-operation > ::=
            (create-wm  < wm-object > {< attribute-value-pair >} ) /
            (create-all-wm  < wm-object > ({< attribute-id > < list-val >}) )
< delete-operation > ::=
            (delete-wm  < wm-object > {< attribute-value-pair >} ) /
            (delete-all-wm  < wm-object > ) / (empty-wm)
< consult-operation > ::=
            (get-wm  < wm-object > {< attribute-value-pair >} ) /
            (get-all-wm  < wm-object  > )
 < modify-operation > ::=
           (modify-wm  < wm-object > {< attribute-value-pair >}
           {< attribute-value-pair >})
< apply-ruleset-operation > ::=
           (apply-ruleset  < ruleset-name >) / (apply-ruleset-top)
 < wm-object > ::= identifier
 < attribute-value-pair >  ::= < attribute-id > < value-attr >
 < attribute-id> ::= ^identifier
 < list-val > ::=  (< value-attr > {< value-attr >})
 < value-attr > ::= < variable > / identifier / (< variable > < variable >)
 < variable > ::= ?identifier
```

Figure 6.3: The syntax of the basic operations

In PRE, the rulesets are organized in a multilevel hierarchy allowing inheritance. Each ruleset has an upper ruleset and inherits features from it. The top of the hierarchy is the TOP ruleset. The TOP ruleset is the only ruleset not having any upper ruleset. This ruleset must be present in any PRE application, whilst other rulesets are optional. Each ruleset controls the activation of the rules that belong to it. The rulesets are described by a set of features. These features are the ruleset name, the name of the upper ruleset, the type of control activating the rules in the ruleset, the name and the type of the sort procedure applied to the rules, and the final condition. If the features describing the type of control, the sort mechanism and the final condition do not appear in the ruleset definition, they are inherited by default from the upper ruleset.

```
< rulesets definition >  ::=  < ruleset definition > {< ruleset definition >          }

< ruleset definition    > ::=
          (ruleset     < ruleset-name >
             isa        < ruleset-name >
             [control   < control-ruleset >]
             [sort-proc  < sort-proc>  ]
             [sort-type  < sort-type > ]
             [final-cond  < final-condition >]
            ) /
            (ruleset     top
              control   < control-ruleset >
              sort-proc  < sort-proc >
              sort-type  < sort-type >
              final-cond  < final-condition >
             )

< ruleset-name > ::=  identifier
< control-ruleset > ::=  forever / until/ one-cycle / first
< sort-proc > ::=  < Lisp-sort-proc  >
< sort-type > ::=   dynamic / static
< final-cond > ::=  nil / (< Lisp-final-function > {< Lisp-final-function > } )
< Lisp-sort-proc > ::=  stardard-sort-procedure  / < Lisp-sort-function >
```

Figure 6.4: The syntax for defining the rulesets

The type of control mechanism states the ruleset application mode. There are four types of control in the rulesets: **forever**, **until**, **one-cycle** and **first**. The control type **forever** states the activation of all the rules in the ruleset until there is an empty cycle, which means until there are no rules satisfying their own conditions. The control type **until** states the activation of all the rules in the ruleset until the final condition of the ruleset is satisfied. The control type **one-cycle** states the activation of one cycle over all rules in the ruleset. Finally, the type **first** activates the first rule satisfying its own conditions.

The sort procedure applied over the rules in the ruleset can be the **standard-sort-procedure**, which sorts the rules by their priority or any other sort procedure previously defined. The type of sort mechanism can be **dynamic**, if it is evaluated each time the ruleset is applied, or **static**, if it is evaluated only once. The final condition is a list of a variable number of final predicates (which can be empty).

The syntax for defining the rulesets is described below in Figure 6.4. The rules belonging to each set are declared following the syntax shown in Figure 6.5.

For each rule, the condition set and the action set have to be declared. The condition set consists of all the statements describing the rule and the conditions governing the rule application. The action set consists of all the actions to be performed when applying the rule. The symbol -> separates the two sets.

```
<rules definition>  ::=  <rule definition> {<rule definition        > }
< rule definition   >   ::=
            (rule         < rule-name >
             ruleset      < ruleset-name >
             priority     number
             control      < control-set >
             < pattern-conditions >
              ->
             < rule-actions >
              )
< rule-name > ::=  identifier
< control-rule > ::=  forever / one / stop
< pattern-conditions > ::=  {< simple-pattern-condition > }
< simple-pattern-condition > ::=
       ([< negation >] < wm-object > {< attribute-value-pair >} )
< negation > ::= no
```

Figure 6.5: The syntax for defining the rules

The features provided in the condition set describing the rule are the rule identifier, the ruleset identifier, the type of rule control mechanism, the rule priority in the ruleset and the statements establishing the conditions governing the working memory objects in applying the rule. There are three types of control mechanism stating the rule application mode: **forever**, **one** and **stop**. The type **forever** states the application of the rule until no WM object satisfies the conditions, the type **one** states its application once at each cycle and the type **stop** only once. The priority of the rule is a number controlling the application rule order within the ruleset. The conditions governing the rule application are a variable number of object descriptions. This description consists of an object identifier and a variable number of attribute-value pairs. Optionally, the word **no** can precede the object identifier to express the negation of the condition.

```
< rule-actions >  ::=
        < create-statement > / < modify-statement >/
        < delete-statement >/ < assign-statement > /
        < apply-ruleset-assignment >
< create-statement > ::=
        (create < wm-object > {< attribute-value-pair >})
< delete-statemnet > ::=
         (delete  < wm-object >) /
         (delete  < wm-object >{< attribute-value-pair >}) /
         (delete  < wm-object-number >)
< modify-statement > ::=
         (modify  < wm-object > {< attribute-value-pair >}{< attribute-value-pair >})
         (modify  <wm-object-number> {<attribute-value-pair>})
< assign-statement > ::=
          (assign  < variable > < expression >) /   <variable > :=  <expression >
< apply-ruleset-statement > ::=
          (apply-ruleset-top){(apply-ruleset   < ruleset-name >)} /
          (apply-ruleset  < ruleset-name >){(apply-ruleset   < ruleset-name >)}
< expression > ::=  < Lisp-function >
< wm-object-number > ::=  number
```

Figure 6.6: The syntax of the allowed operations in the action part of the PRE rules

The action set consists of all the actions to be performed when applying the rule. Only five statements are accepted in the action part of the rule:

- **create**
- **delete**
- **modify**
- **apply-ruleset**
- **assignment**

The syntax defining the possible statements in the action set of the rules is shown in Figure 6.6. The statements **create**, **delete**, **modify** and **apply-ruleset** are implemented using the basic WM operations described above, in Figure 3. The assignment statement performs the assignment of values and functions to variables. The use of functions in this statement enriches the power of the formalism. In the basic set of control rules defined for obtaining the application-restricted interfaces, these functions have been restricted to a set of predefined ones.

The definition of all rulesets and rules performing a task are included in a file. The functions necessary for controlling and sorting these rulesets and rules are included in an

auxiliary file. For each file containing rulesets, there is a particular file containing the auxiliary functions used in the rulesets. This file includes, at least, the sort procedure.

Once the set of control rules organized into rulesets is complete, the top rule can be applied in order to perform the obtention of application-restricted linguistic resources. The top rule is activated when the operation **apply-ruleset** is called from any program executed in the environment.

PRE includes a menu-based user interface to simplify the use of this environment during the tuning and debugging phase. The PRE user interface consists of a menu in which all available tools and actions are displayed.

The options appearing in the PRE menu are four: **Switch Explain**, **Load Rulesets**, **Load aux fn** (for loading auxiliary functions) and **Execute Rulesets**.

The first option either enables or disables the full trace of rulesets and rule execution. The option **Load Ruleset** loads and tests the file containing the rulesets. The syntactic errors detected when testing this file are described on-screen. The option **Load aux fn** loads the file containing the auxiliary functions. Finally, the **Execute Rulesets** option calls the operation **apply-ruleset-top**, which activates the ruleset TOP. This ruleset must be in the file, previously loaded, containing all rulesets. The operation **apply-ruleset-top,** which starts the application of all rulesets loaded, can also be called from any program executed in the environment.

Examples of the explanations appearing on-screen when the **Switch Explain** option is active are shown in Figure 6.7. These explanations correspond to the execution of the ruleset **creating_instance** (described in the next section) over the WM object described in Figure 6.1. The details of the performance of the rules in the ruleset appear on-screen, as shown in the Figure 6.7. When the ruleset TOP calls the action **apply-ruleset creating_instance**, all rules in this ruleset are activated. First, the description of this action is displayed on-screen (the name of the action, the name of the object, the type of the object and the pattern). Then, the ruleset name, **creating_instance** and the type of control mechanism applied in the ruleset, **one-cycle,** appear.

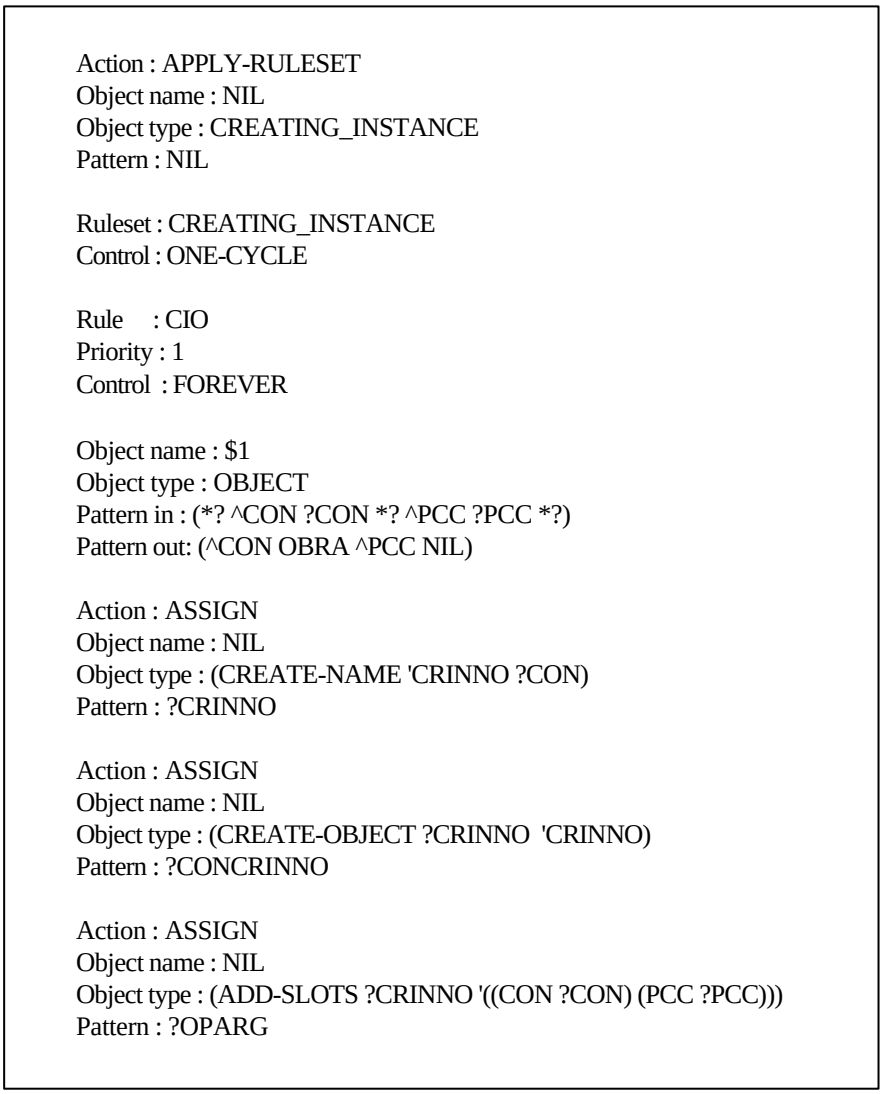

Figure 6.7: A trace of the ruleset creating_instance

Next, the first rule in the ruleset to be applied is described: its name, **cio**, its priority, 1, the control mechanism stating the rule application mode, **forever**, and the condition governing the rule. In rule **cio**, this condition consists of the description of the WM object shown in Figure 6.2. It represents all WM objects of type **object** matching the pattern **^con ?con ^pcc ?pcc**.

The description of the actions performed by the rule also appears on-screen. In the figure, the first three statements executed by the rule are described. Each one of the three statements performs an assignment. The first statement assigns to the variable **?crinno** the symbol returned by the function **create-name**. This function concatenates two or more words. In this statement, it concatenates the name **crinno** (representing the

CREATE_INSTANCE_WITH_NO_NAME_O) to the concept identifier represented in the variable **?con**. In the example considered, the function **create-name** returns the symbol **crinnoobra**. In the second statement, the operation assign followed by the call to the function **create-object** creates a new object in the CO. In this example, the operation **crinnoobra** (for creating an instance of the concept **obra**) will be represented in the CO as an instance of the operation **crinno.** In the third statement, the operation assign followed by the call to the function **add-slots** fills the facet **con** of the operation generated with the concept identifier (in the example **obra**) and the facet **pcc** with the concept preconditions (**nil**).

PRE has been implemented in LISP. The PRE interface has been integrated into the Common Lisp user interface[3].

6.4 THE BASIC SET OF CONTROL RULES

The formalism supporting the control rules makes it possible to define a rich variety of rules relating objects in one or more ontologies. Different rules can be defined according to the desired level of coverage. However, to incorporate a new rule it is necessary to know the definition formalism (described above) as well as the details of the CO and LO implementation. In order to make the implementation of the system transparent to the user and to facilitate its use, a general basic set of control rules for generating application-restricted NLIs was defined and implemented. This basic set of rules establishes general relations between the CO and the LO in order to obtain the linguistic resources necessary for application-restricted interfaces.

The same basic set of control rules can be used for different types of applications. Basically, three different types of interfaces have been taken into account when defining the basic set of rules. These types are interfaces guiding the users to describe particular cases of the application general knowledge, interfaces allowing the users to consult application knowledge, and interfaces supporting both, consultations and descriptions.

The most important goals in the design of the basic set of control rules were the friendliness and efficiency of the NLIs generated. The control rules in this set have been defined to generate the minimum number of grammar rules to express all possible operations over the CO in a natural way. One or two linguistic realizations for each operation were considered. Current implementation ensures the generation of linguistic structures covering elliptical reference, coordination, direct and indirect interrogation and other linguistic phenomenon. The basic set of control rules does not cover other complex linguistic phenomenon not favoring the efficiency of the NL processing.

Additionally, there is an alternative set of rules controlling the generation of larger grammars. The process of obtaining this grammar is the same for the two sets of rules. The alternative set of rules only differs from the basic set in that several linguistic realizations for each operation are generated. The alternative set only differs in a couple of rules from the basic set. For this reason, the description of the basic set of rules given in this section covers both the alternative and the basic set of rules.

The grammars representing only one or two paraphrases of each allowable operation are appropriate when the input is introduced using a menu-system displaying all possible options. However, when users type the sentences without any guide, grammars supporting several forms of expressing each operation are more appropriate. That is, when a menu-system is used, the grammars generated by the alternative set are not as efficient as those generated by the basic set, but they are more appropriate when the menu-system is not used.

The syntax of the control rules allows the definition of different rules for other purposes, such as the generation of explanations about the tasks performed by the applications and the generation of the linguistic resources for other types of NLIs.

6.4.1 The functions used in the basic set of control rules

The basic set of control rules was implemented in PRE, following the formalism described above. Although PRE allows indiscriminate use of functions in the assignment operation, the formalism used in the basic set constrains such a powerful mechanism, limiting the

[3] There is also a recent implementation of PRE in PEARL.

functions to a set of predefined ones. This set of functions performs all actions needed when relating the CO to the LO in order to generate the linguistic sources in the application-restricted NLI. The set consists of eight functions.

These functions are:

- **opco**
- **create_name**
- **create-object**
- **add-slots**
- **get-slot-value**
- **get-immediate-slot-value**
- **obtain_semfeatures**
- **create-lambda-function**

The syntax of these eight functions is described in Figure 6.8.

Four of these functions perform the basic operations of creating, modifying and consulting ontology objects. These functions are: **create-object**, performing the creation of ontology instances, **add-slots**, performing the modification of them and **get-slot-value** and **get-immediate-slot-value**, both performing the consult of the attributes (and facets) of the ontology objects.

The parameters of the function **create-instance** are two identifiers, the first name corresponds to the instance to be generated and the second one to the name of the ontology object. The parameters of the function **add-slots** are the identifier of an instance and a list of attribute-value pairs. The two functions consulting the ontology objects, **get-immediate-slot-value** and **get-slot-value**, have two parameters: the object identifier and the attribute identifier. The first function returns all values of the attribute object while the second only returns the first value.

The other four functions used in the basic set of control rules are functions especially built for dealing with specific aspects in the generation of the application-restricted linguistic structures. The performance of these functions is described below.

The function **opco** performs the creation of all WM objects representing the concepts describing the application in the CO. This function has only one argument: a CO object

identifier. For each interface concept in the CO, this function creates the WM representing the concept identifier and the WM object representing each of its attributes.

```
< Lisp-function-NLI > ::=
      < inicialization-function > /
      < create-name-function > / < create-object-function >/
      < add-slots-function > / < get-slot-value-function > /
      < create-lambda-f-function > / < obtain-semfeatures-function >
< inicialization-function > ::=
      (opco <object-identifier>)
< create-object-function > ::=
      (create-object  < object-identifier > < object-identifier > )
< add-slots-function > ::=
      (add-slots  <object-identifier> {(< attribute-identifier > {<attribute-value>})})
 < get-slot-value-function > ::=
     (get-slot-value  < object-identifier >< attribute-identifier >) /
     (get-immediate-value  < object-identifier >< attribute-identifier >)
< create-instance-function > ::=
    (create-name identifier   ({< name >}))
< create-lambda-f-function > ::=
    (create-lambda-function  < name > ({< attribute-identifier >}) )
< obtain-semfeatures-function > ::=
      (obtain_semfeatures ({< attribute-identifier >}) < operation-identifier >)
< object-identifier > ::=  variable / identifier
< attribute-identifier > ::=  variable / identifier
< attribute-value> ::=  variable / identifier
< name > ::= variable / identifier
< operation-identifier > ::= < object-identifier >
```

Figure 6.8: The predicates used in the assign-statement in the basic set of control rules

The function **create-name** performs the concatenation of two or more words. It is used to create names of ontology objects. The resulting names are used in the four functions accessing the ontologies described above.

The function **obtain-semfeatures** returns the value of one or two facets (or attributes) describing a CO object. This function has two parameters. The first parameter consists of a list of one or two facets of the CO object. The second parameter of the function is the identifier of the CO object. The function returns a list containing the facets passed as the first parameter and their values.

The function **obtain-semfeatures** is used to obtain the semantic features associated with terminal linguistic instances generated for an application. More precisely, it obtains the semantic features associated with the terminal linguistic instances expressing the parameters of the allowable operations. These parameters can be concepts (**con)**, attributes

(**attr)** and values (**val**). The terminal instances representing concepts are associated with the semantic feature **con**. The terminal instances representing attributes and values are associated with the semantic features **con** and **attr**. When generating the terminal linguistic instances necessary to express an operation, the value of these features is obtained by means of the function **obtain-semfeatures**. This function is called with two parameters: the list of the semantic features associated with the linguistic instance and the operation identifier.

For example, the operation **FILL_ATTRIBUTE_IS_BUILDING_REQUIREMENT FULFILLED** (for filling the attribute **fulfilled** of the concept **BUILDING_REQUIREMENT**) has three parameters: the parameter **con** (representing the concept **BUILDING_REQUIREMENT**), **attr** (representing the attribute **fulfilled**) and **val** (representing the value). The LO instance expressing the value of the parameter **attr** of this operation, **fulfilled**, is shown in the Figure 6.9. This LO instance is described by the facets: **category**, **lex**, **sem** and **type**.

DESCRIPTIVE_ADJECTIVE_FULFILLED

instance: DESCRIPTIVE_ADJECTIVE_ATTRIBUTE_IS
category: dadjattris(syn(gen(f),num(p)),sem(con(buildingrequirement),attr(fulfilled)))
lex: cumplidas
sem: fulfilled
type: lex

Figure 6.9: The LO object representing the attribute fulfilled of the concept BUILDING_REQUIREMENT

The category of the instance is **dadjattris**, associated with descriptive adjectives (**dadj**) representing attributes belonging to the class **IS**. This category is augmented with syntactic and semantic features. The syntactic category, the syntactic features as well as the superficial presentation, **cumplida**, are obtained from the set containing the syntactic description of all application terms.

The semantic interpretation is **fulfilled** and it corresponds to the attribute identifier. The semantic features associated with all LO objects representing conceptual attributes are **con** and **attr**. The value of these features is obtained by calling the function **obtain-semfeatures**. The call to the function **obtain-semfeatures** with the parameters **(con attr)**

and **FILL_ATTRIBUTE_IS_BUILDING_REQUIREMENT_FULFILLED**, returns the list (**con(buildingrequirement),attr(fulfilled)**). The values of the features are in brackets. This list represents the semantic features associated with the category **dadjattris**.

Finally, the function **create-lambda-function** generates the lambda functions corresponding to the semantic interpretation of the LO instances expressing the arguments of specific operations.

As described in Chapter 5, the semantic interpretation associated with the LO instances expressing the CO objects involved in the communication is represented using the lambda calculus. The semantic interpretation of LO objects having more than one constituent consists of a list of numbers indicating the interpretation order of its constituents. This information is obtained from the linguistic upper class. The semantic interpretation of a LO object representing one word (or more) consists of a lambda function or a lambda value, depending on the linguistic class. For example, articles are always associated with a lambda function and nouns with a lambda value.

In the basic set of rules, the linguistic instances representing the parameters of the simple operations (filling or consulting a conceptual attribute) have as semantic interpretation a lambda value. A terminal linguistic instance expressing a concept, an attribute or a value in these operations has the concept identifier, the attribute identifier or the value, respectively, as semantic interpretation.

The instances of transitive verbs representing a concept in operations expressing the filling of more than one attribute have a lambda function as semantic interpretation. The name of this lambda function is the concept identifier and its parameters correspond to the conceptual attributes to be filled. The function **create-lambda-function** returns the lambda function for the terminal linguistic instances representing a concept in these operations. The function **create-lambda-function** has two parameters: the identifier representing the concept and the identifier representing the operation.

In current implementation, only one class of operation that fills more than one conceptual attribute is considered, the class representing operations realized as transitive clauses. In these operations, the attributes to fill belong to the attribute classes **WHO_SUBJECT**, **WHAT_OBJECT** and **WHO_OBJECT**. In the clauses expressing these operations, the argument representing the concept is realized as a transitive verbal group. The arguments

representing the attributes to be filled are represented by linguistic instances with the roles **subject**, **direct_object** and **indirect_object**, respectively.

For example, the operation **FILL_TRANSITIVE_CONCEPT_BUILDING CONTRACT_PARTS** fills the attributes **subject1**, **subject2** and **object** of the concept **BUILDING_CONTRACT_PARTS**. This operation is shown in Figure 6.10.

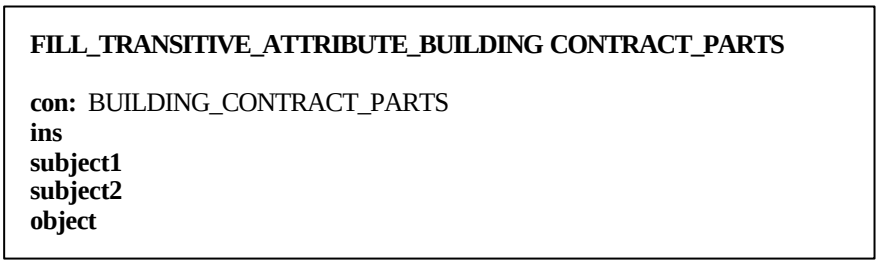

Figure 6.10: A fragment of the operation FILL_TRANSITIVE_CONCEPT_BUILDING_CONTRACT_PARTS

The semantic interpretation associated with the linguistic instance expressing the argument **con** of this operation (whose value is **BUILDING_CONTRACT_PARTS**) is obtained by the function **create-lambda-function**. When calling this function with the concept identifier (**BUILDING_CONTRACT_PARTS**) as a first parameter and the operation identifier (**FILL_TRANSITIVE_CONCEPT_BUILDING_CONTRACT_PARTS**) as a second parameter, a lambda function is return. The returned function has the form:

(((l, _subject1),(l, _object),(l, _subject2)),
(buildingcontractparts, subject1, _subject1, object, _object, subject2, _subject2))

The first part of the function is a sublist defining the three parameters. Each parameter (i.e. **(l, _subject1))** is represented by a sublist of two elements: the symbol **l** and a variable (represented by an underscore and the attribute identifier, as in **_subject1**). The second part of the function corresponds to the resulting list where the variables representing the parameters must be substituted by the value when applying the function.

When applying this function over the lambda value **owner**, associated with the nominal group expressing the value of the attribute **subject1**, the result will be:

(((l, _object),(l, _subject2)),
(buildingcontractparts, subject1, owner, object, _object subject2, _subject2))

The resulting function will be applied next on the two lambda values associated with the nominal group expressing the value of the attribute **object** and the value of the attribute **subject2**, respectively.

The functions used in the basic set of rules are implemented in Lisp. They have been implemented using the predicates creating and consulting objects of the language **frame-kit** ([Carbonell86]).

6.4.2 The rulesets

Following the PRE formalism, in the basic set of control rules proposed, rules are grouped into rulesets. Each ruleset performs a different action and each rule in the ruleset is applied over a different type of object. The basic set of control rules is organized into 8 rulesets. The definition of these rulesets is shown in Figure 6.11.

These basic rulesets are activated by typing the activation function in the PRE user interface. There are three possible activation functions: **inic**, **inim** and **inicm**. These functions state the initial conditions for generating the appropriate linguistic resources for each type of interface: consulting, describing and consulting-describing.

The activation functions load the initial lexicons necessary for each type of application. They also create a WM object representing the type of the interface that must be generated.

The first ruleset is the TOP ruleset. The TOP ruleset is responsible for the initialization process. This ruleset is described in Figure 6.16. It checks the initial conditions and activates the appropriate ruleset for each case. The initial conditions state the type of interface to be generated. These conditions are represented in the WM, just before the TOP ruleset is applied. The ruleset TOP activates the rulesets responsible for the first step of the generation process. In this step, the rulesets perform the creation of operations for each application concept represented in the CO. Four different rulesets ensure this step of the process. Each one of these rulesets performs the generation of instances of a different class of operation.

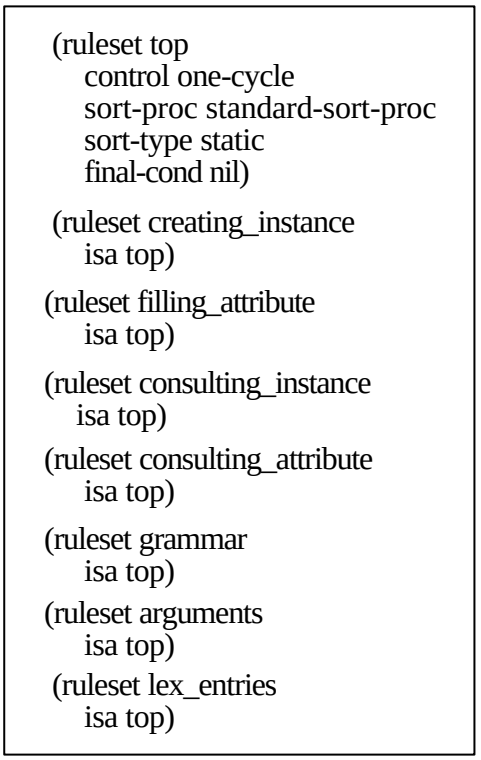

Figure 6.11: The rulesets definition in the basic set of rules

The rulesets are:

- **creating_instance**
- **filling_attribute**
- **consulting_instance**
- **consulting_attribute**

These rulesets are described in Figures 6.17-6.20.

Three different rulesets are responsible for the second step of the process, when instances are created of the LO objects supporting the NL expression of the allowed operations for an application. These rulesets are the ruleset **grammar**, relating operations created for an application to the LO, the ruleset **arguments**, classifying the different types of operation arguments and the ruleset **lexical_entries,** creating the LO objects representing these arguments. They are described in Figures 6.21-6.29. The ruleset activation is described in Figure 6.14.

6.4.3 The performance

As mentioned before, the basic set of control rules is responsible for the process of generating the linguistic resources needed in the application-restricted interfaces. This generating process is carried out in three steps.

The first step

The first step of the process consists of generating operations representing the communication tasks required for an application. For this purpose, the general operations are adapted to the CO concepts representing the application. Instances of the necessary operations are generated for each concept. A scheme of this first step is shown in Figure 6.12.

```
for each CONCEPT in APPLICATION_ONTOLOGY do
    generate_CO_instance_operations_modifying_concept (CONCEPT)
    generate_CO_instance_operations_consulting_concept (CONCEPT)
endfor
```

Figure 6.12: The first step of the process of obtaining application-restricted linguistic resources

The CO operations modifying or consulting attributes are classified according to the attribute class. As described in the scheme in Figure 6.12, instances of the appropriate operation subclasses for filling and consulting attributes are created for each concept attribute. In the basic set of control rules, only simple operations are generated, that is, operations performed only over one concept.

Operations for filling more than one attribute of a conceptual instance are represented as instances of the class **FILL_MORE_ONE_ATTRIBUTE_O**, shown in Figure 4.5. In current implementation, these operations have been considered as combinations of the simpler operations filling a conceptual attribute. However, a specific subclass of this operation, the **FILL_TRANSITIVE_CONCEPT_O**, has been considered as a special case because it represents operations expressed more naturally in a simple transitive clause. This class represents operations filling the conceptual attributes belonging to the classes **WHO_SUBJECT**, **WHAT_OBJECT** and **WHO_OBJECT**.

The second step

The most complex step of the process for obtaining the appropriate linguistic resources for an application is the second one. In this step, the LO general structures are adapted to cover the expression of the CO objects generated in the first step. A scheme of the performance of this step is shown in Figure 6.13.

```
for each OPERATION_INSTANCE in CASE_ONTOLOGY do
   generate_CLAUSE_instances (OPERATION_INSTANCE)
   for each ARGUMENT in OPERATION_INSTANCE do
     generate_GROUP/WORD_instances (OPERATION_INSTANCE, ARGUMENT)
   endfor
endfor
```

Figure 6.13: The second step of the process of obtaining application-restricted linguistic resources

As described in Chapter 5, the LO general linguistic classes were adapted to represent the taxonomy of operations described in the CO. In the LO domain level, subclasses supporting the expression of all operations were described. In this second step, instances of the LO domain level classes were defined in order to support the specific operations for an application.

Three rulesets are responsible for this step: the ruleset **grammar**, the ruleset **arguments** and the ruleset **lexical_entries**.

The ruleset **grammar** ensures the process of generating LO instances representing the CO operations created in the first step of the process. Different linguistic instances are generated to support the expression of each class of operation.

If operations creating conceptual instances have been generated then the linguistic classes **EXISTENTIAL_CLAUSE_CREATE_INSTANCE_WITH_NO_NAME** (shown in Figure 5.6) and **ATTRIBUTIVE_CLAUSE_CREATE_INSTANCE_WITH_NAME** (shown in Figure 5.7) are marked as active rules. Grammar rules representing these classes will be incorporated into the application-grammar at the third step of the process.

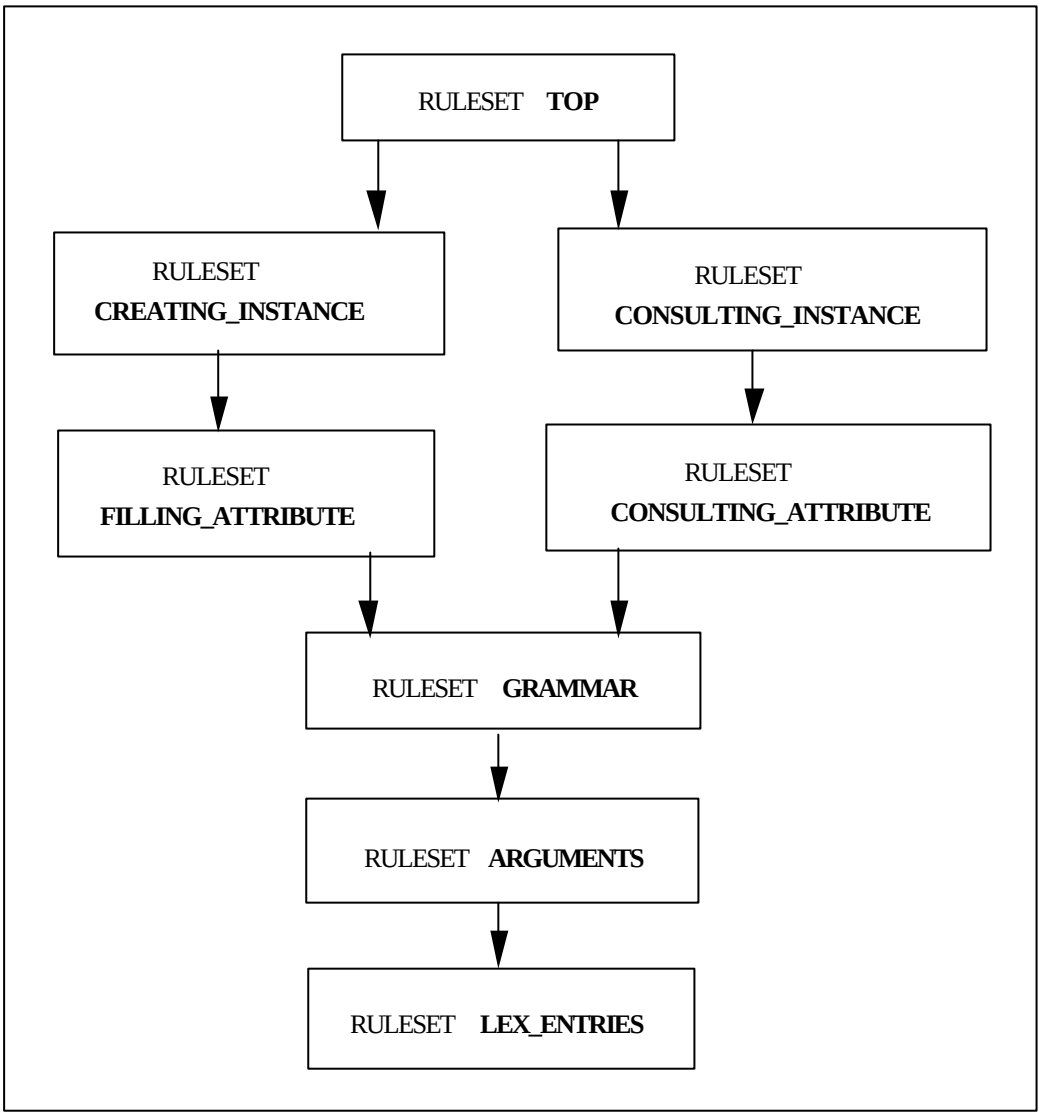

Figure 6.14: The ruleset activation in the basic set

If operations consulting conceptual instances have been generated then the linguistic classes **COMPLETE_INTERROGATIVE_CONSULT_CLASS**, **COMPLETE INTERROGATIVE_CONSULT_INSTANCE** and **PATIAL_INTERROGATIVE CONSULT_INSTANCE** are marked as active rules.

The realization of operations for filling one attribute of a conceptual instance is represented as instances of the classes **ATTRIBUTIVE_CLAUSE**, **TRANSITIVE_CLAUSE** and **INTRANSITIVE_CLAUSE**.

Operations for consulting the existence of a conceptual instance as well as those consulting one of its attributes are represented as instances of the classes

COMPLETE_INTERROGATIVE_CLAUSE and **PARTIAL_INTERROGATIVE CLAUSE**.

Operations filling more than one instance attribute are related to the class **COORDINATE_CLAUSE**. The constituents of this class are the simple major clauses expressing operations creating conceptual instances and filling one of its attributes. Operations in the class **FILL_TRANSITIVE_CONCEPT_O** are related to the class **INDIRECT_OBJECT_CLAUSE_MODIFY**.

The linguistic objects representing operations to create and to consult concepts and operations for filling more than one attribute are application independent. They are always expressed by the same linguistic structures. For this reason, the linguistic objects representing these operations belong to the general linguistic knowledge represented in the LO and the rules in the ruleset grammar *mark* these instances when they are necessary for an application.

The linguistic structures representing operations to fill and to consult one attribute concept depend on the classes of the attributes of the application concepts. Operations filling and consulting one concept attribute are subclassified regarding the attribute class. New subclasses of attributes can be incorporated when representing the application in the CO, thus, different subclasses of operations filling and consulting attributes can be created for different applications.

The rules in the ruleset **grammar** generate different linguistic instances for each operation created in the first step. The linguistic information in the instances generated is inherited from their linguistic classes. In the linguistic structures expressing operations, this information consists of the pattern or correct distribution of the constituents and the syntactic and semantic agreement between them. Conceptual information describing the operation is also incorporated. The operation parameters are represented as the constituents of the linguistic instances generated. The category associated with each constituent represents the type of argument (i.e. **defngcon**, for definite nominal groups representing concepts). For arguments representing attributes and values, the category includes information about the class of the attribute (e.g. **defngattrof**, for definite nominal groups representing attributes in the class **OF** and **defngvalwhodoes** for values of attributes in the class **WHO_DOES**). The name of the operation and the operation preconditions are also incorporated into the linguistic instances.

Once the ruleset **grammar** has been applied, the rulesets **arguments** and **lexical_entries** are activated. These two rulesets ensure the process of creating terminal linguistic instances expressing each parameter of the operations created for an application. The ruleset **arguments** creates WM objects representing all the semantic and syntactic information necessary for generating these linguistic objects. These WM objects are used by the ruleset **lexical_entries** to generate the terminal linguistic instances.

The ruleset **arguments** obtains the syntactic information associated with each parameter of the operation from the corresponding entries in the set defining the application terms. All the parameters of operations (concepts, attributes and values) are linked to one (or more) entries in this set. The linguistic information contained in this set for each entry consists of the syntactic category, the syntactic features (number, gender, and tense) and the surface realization.

Different types of parameters are considered when obtaining the appropriate syntactic and semantic information associated with the linguistic instances generated. The rules in the ruleset **arguments** considers the several types of parameters: conceptual classes, conceptual instances, simple attributes, compound attributes, closed values, open values, menu values and values representing concepts. These types of parameters are classified into lexical and dynamic objects depending on whether their value is set during the generation process (by the ruleset argument) or at run-time.

The description of a lexical instance consists of its linguistic realization, its category and its semantic interpretation. The semantic interpretation associated with lexical objects is a lambda function or value.

Dynamic linguistic instances are terminal instances described by the category and the name of the function that will obtain the argument value at run-time. As has been said above, there are three different types of dynamic functions: functions obtaining instances of concepts existing in the case level, functions asking the user to introduce a value at run-time and functions activating a menu screen where a set of allowed attribute values are displayed.

The ruleset **lexical_entries** ensures the creation of linguistic instances incorporating all syntactic and semantic information describing the operation parameter. This information is obtained from the WM objects created by the ruleset **arguments**. For each of the linguistic realizations associated with an operation argument, a linguistic instance is created. The

linguistic realizations are obtained from set of application terms. An example of a linguistic instance generated, representing the concept **ARCHITECT**, is shown in Figure 6.15.

DEFNGCONARCHITECT

instance: defngcon
type: lex
cat: defngcon(syn(gen(m), num(s)), sem(con(architect)))
lex: el_arquitecto
sem: architect

*Figure 6.15: The definite nominal group representing the concept **ARCHITECT***

The ruleset **lexical_entries** incorporates semantic information into the linguistic category. This information consists of the type of argument (**con**, **attr**, **val**), the class of the attribute (only for attributes and values, such as **attrhas** and **valof**) and the semantic restrictions associated with the category (the concept and attribute name). The semantic features associated with a category are obtained from the LO class. The values of these semantic features for a specific parameter are obtained from the instance of the operation.

As shown in Figure 6.15, in the terminal instances the syntactic and semantic features describing the linguistic object are directly associated with the category. For example, the object in Figure 6.15 is a terminal lexical instance representing the definite nominal group *el_arquitecto* (*the_architect*). The category of this instance is **defngcon**. This category has been augmented with syntactic and semantic features. The syntactic feature is **num** (abbreviation of number) and its value is **s** (the abbreviation of singular). The semantic feature associated is **con** (representing the concept) and its value, **architect**, is obtained from the operation parameter **con**. The semantic interpretation **architect** and it is represented in the facet **sem**. Its linguistic realization is represented in the facet **lex**. The type of the terminal instance is described in the facet **type**. Its value can be **lex** (if the instance is associated with a string) or **dyn** (if it is associated with a dynamic function).

A more detailed description of this basic set of rules is given below.

The ruleset TOP

The ruleset TOP is the initial one. This ruleset checks the initial conditions indicating the type of the interface that has to be generated and activates the appropriate rulesets for each case.

The ruleset TOP contains two rules: the rule **consulting**, which starts the process of generating interfaces for consulting, and the rule **modifying**, which initiates the generation of interfaces guiding the user to describe particular information. When the communication with a specific application requires user consults and descriptions, both rules are applied to obtain the appropriate interface.

The implementation of the two rules in the ruleset TOP is shown in Figure 6.16.

As can be seen in the figure, the condition set of each rule states: rule name, ruleset name, rule priority and the conditions governing the rule. The two rules in the ruleset have priority 1, maximum priority. The type of control in these rules is **one**, meaning that they are applied just once over the initial conditions. The initial conditions are represented as object identifiers in the WM. If the initial condition is the identifier **consult**, then the rule **consulting** is applied; if it is the identifier **modify**, then the rule **modifying** is applied. If both the identifier **consult** and the identifier **modify** are represented, then the two rules in the TOP ruleset are applied.

```
(rule consulting
   ruleset top
   priority 1
   control one
   (consult)
   ->
   (?x1 := (opco 'concept))
   (apply-ruleset consulting_instance))

(rule modifying
   ruleset top
   priority 1
   control one
   (modify)
   ->
   (?x1 := (opco 'concept))
   (apply-ruleset creating_instance))
```

Figure 6.16: The ruleset TOP

The action part of the two rules consists of two statements. The first statement is a call to the function **opco** responsible for representing the CO concepts in the WM. The second statement in both rules is the activation of the next ruleset to be applied: the rule **consulting** activates the ruleset **consulting_instance** and the rule **modifying** the ruleset **creating_instance**.

The CO application concepts and their attributes are represented as WM objects and the following rules are applied over these WM objects. The parameter of the function **opco** indicates the CO concepts that have to be represented in the WM. In the basic set of rules, all concepts and attributes described as interface entities**,** representing all objects appearing in the communication, are represented in the WM.

The ruleset creating_instance

The ruleset **creating_instance** is responsible for generating instances of the two operations for creating instances of application concepts: **CREATE_INSTANCE_WITH_NO_NAME_O**, for creating instances of concepts without giving their identifier and **CREATE_INSTANCE_WITH_NAME_O**, for creating conceptual instances having a given identifier. This ruleset is applied over the CO application concepts described as interface entities. For each concept, instances of the two operations are created in the CO case level. The implementation of the two rules is shown in Figure 6.17.

The ruleset **creating_instance** contains two rules: the rule **cio**, creating the instances of operations and the rule **next_fa**, activating the next ruleset to be applied.

The control type of the rule **cio** is **forever**, which states the application of the rule until no object in the WM satisfies the conditions. The rule condition is the description of a CO concept represented in the WM. This description corresponds to the WM shown in Figure 6.2. Each CO concept is described by the word **object**, its name (represented by the attribute **^con** and the variable **?con**, containing the concept name) and the preconditions associated with the concept (**^pcc ?pcc**). This rule is applied over all WM objects satisfying this description.

```
(rule cio
  ruleset creating_instance
  priority 1
  control forever
  (object ^con ?con ^pcc ?pcc)
  ->
  (?crinno := (create-name 'crinno ?con))
  (?concrinno := (create-object ?crinno 'crinno))
  (?oparg := (add-slots ?crinno '((con ?con) (pcc ?pcc))))

  (?criwno := (create-name 'criwno ?con))
  (?concriwno := (create-object ?criwno 'criwno))
  (?oparg := (add-slots ?criwno '((con ?con) (pcc ?pcc))))

  (create ocinn ^name ?crinno ^con ?con ^pcobject ?pcc)
  (create ociwn ^name ?criwno ^con ?con ^pcobject ?pcc)
  (delete 1))


(rule next_fa
  ruleset creating_instance
  priority 3
  control one
  ->
 (apply-ruleset filling_attribute))
```

*Figure 6.17: The ruleset **creating_instance***

The action part of the rule consists of nine statements. The three first statements are responsible for creating a CO instance of the operation creating a conceptual instance without giving its name, for each WM object satisfying the rule condition. The first statement assigns (**:=**) to the variable **?crinno** the symbol resulting from concatenating the abbreviation of the name of the operation (**crinno**)**,** to the name of the concept (represented in the variable **?con**). In the second statement, the operator **assign** followed by the call to the function **create-object** creates the instance of the operation **crinno**. The name of this instance is given by the variable **?crinno**, which has been instantiated in the previous statement. In the third statement, the function **add-slots** fills the attribute **con** of the operation generated with the concept name and the attribute **pcc** with the concept preconditions. The three next statements are responsible for creating an instance of the **CREATE_INSTANCE_WITH_NAME_O** (abbreviated **criwno**). The seventh and eighth statements create the WM objects representing the new instances of operation created in the case level in the CO. The first attribute of the two WM objects generated is **^name** and its value is given by the variable representing the instance name (**?crinno** and **?criwno** respectively). The second attribute is **^con** and its value is represented by the variable **?con**. The third attribute is **^pcobject** and its value is represented in the variable **?pcc**.

Finally, the last statement uses the operator **delete** to remove the previously matched object from the WM.

Once this rule has been applied for all objects satisfying the condition statement, other rules in the same ruleset are applied. In this ruleset, there is only one further rule, the rule **next_fa**. This rule has the lowest priority, and it is applied once. The only action of the rule is a call to the next ruleset to be executed by means of the operator **apply-ruleset** and the name of the ruleset, **filling_attribute**.

The ruleset filling_attribute

The ruleset **filling_attribute** is responsible for generating instances of operations filling the attributes of conceptual instances in the case ontology. In this ruleset, instances of two different simple operations filling attributes are generated: instances of the operation filling one attribute of a conceptual instance and instances of the operation filling more than one attribute of a conceptual transitive instance. The rules contained in this ruleset are shown in Figure 6.18.

Each one of the first four rules in the ruleset generates instances of a different class of operation. The fifth rule activates the next ruleset to be applied. The rule **fa_op** generates one instance of the simple operation filling one attribute of a conceptual instance for each conceptual attribute in the CO. The rules **fatid_op**, **fati_op** and **fad_op** generate instances of operations filling more than one attribute concept that can be expressed by transitive clauses. The rule **fatid_op** creates instances of the operation to fill transitive concepts, that is, to fill conceptual attributes belonging to the three classes **WHO_SUBJECT**, **WHO_OBJECT** and **WHAT_OBJECT**. The rule **fad_op** creates instances of operations to fill conceptual attributes belonging to the classes **WHO_SUBJECT** and **WHAT_OBJECT**. The rule **fati_op** creates operations to fill the attributes belonging to the classes **WHO_SUBJECT** and **WHO_OBJECT**.

The first four rules in the ruleset have a similar structure. The rule condition consists of the description of the WM object representing a CO concept and one (or more) of its attribute(s). The statements in the action part of the rules create an instance of the corresponding operation in the CO case level as well as in the WM. The first statement creates the name of the instance. The second statement creates the instance in the CO case

level. The third statement fills the arguments of the instance generated with the concept and attribute(s) identifiers. The forth statement creates the WM representing the instance. The sixth statements deletes the WM matched.

The rule **fa_op** creates the appropriate subclass of the **FILL_ATTRIBUTE_O** for each conceptual attribute. It is applied over all WM objects representing the attribute of the application concepts in the CO. These WM objects are described by the word **attrcon**, the attribute class (**^attrclass ?attrclass**), the concept name (**^con ?con**), the attribute name (**^attr ?attr**) and the concept preconditions (**^pcc ?pcc**).

Because of the organization of operations filling an instance attribute according to the class of the attribute to be filled, one instance of the appropriate operation subclass is generated for each conceptual attribute. Finally, the rule **next_grammar** is responsible for activating the next ruleset to be applied, the ruleset **grammar**.

```
(rule fa_op
  ruleset filling_attribute
  priority 1
  control forever
  (attrcon ^attrclass ?attrclass ^con ?con ^attr ?attr  ^pcc ?pcc)
  ->
  (?opclass := (create-name 'fa ?attrclass))
  (?opgen := (create-name ?opclass '(?con ?attr)))
  (?op := (create-object ?opgen ?opclass))
  (?oparg := (add-slots ?opgen '((con ?con)(attr ?attr)(pcc ?pcc))))
  (create argop ^attrclass ?attrclass ^name ?opgen ^attr ?attr )
  (create operation ^opclass ?attrclass ^name ?opclass ^mod decsent)
  (delete 1))

(rule fatid_op
  ruleset filling_attribute
  priority 1
  control forever
  (cfat ^con ?con ^whos ?whos ^whob ?whob ^whato ?whato ^pcc ?pcc)
  ->
  (?opfatran := (create-name 'fatid ?con))
  (?op := (create-object ?opfatran 'fat))
  (?oparg := (add-slots ?opfatran '((con ?con)(?whos)(?whob)(?whato)(pcc ?pcc))))
  (create argtrop ^op ?opfatran ^con ?con ^whos ?whos ^whob ?whob ^whato ?whato )
  (delete 1))

(rule fati_op
  ruleset filling_attribute
  priority 2
  control forever
  (cfat ^con ?con ^whos ?whos ^whob ?whob ^pcc ?pcc)
   ->
  (?opfatran := (create-name 'fati ?con))
  (?op := (create-object ?opfatran 'fat))
  (?oparg := (add-slots ?opfatran '((con ?con)(?whos)(?whob)(pcc ?pcc))))
  (create argtrop ^op ?opfatran ^con ?con ^whos ?whos ^whob ?whob)
  (delete 1))

(rule fad_op
  ruleset filling_attribute
  priority 2
  control forever
  (cfat ^con ?con ^whos ?whos ^whato ?whato ^pcc ?pcc)
  ->
  (?opfatran := (create-name 'fatd ?con))
  (?op := (create-object ?opfatran 'fat))
  (?oparg := (add-slots ?opfatran '((con ?con)(?whos)(?whato)(pcc ?pcc))))
  (create argtrop ^op ?opfatran ^con ?con ^whos ?whos ^whato ?whato )
  (delete 1))

(rule next_grammar
  ruleset filling_attribute
  priority 3
  control one
  ->
 (apply-ruleset grammar))
```

*Figure 6.18: The ruleset **filling_attribute***

The ruleset consulting_instance

The ruleset **consulting_instance** is responsible for generating instances of the three operations for consulting ontology concepts. These three operations are: the **CONSULT_CONCEPT_CLASSES_O**, for consulting whether a conceptual class exists, the **CONSULT_CONCEPT_INSTANCES_O**, for consulting whether a specific conceptual instance exists and the **CONSULT_ALL_CONCEPT_INSTANCES_O**, for asking all existing instances of a specific conceptual class.

For each CO concept, instances of the three operations are created in the CO case level. Rules in this ruleset are similar to the rules in the ruleset **creating_instance**. The implementation of the two rules in this ruleset is described in the Figure 6.19.

The rule **ci_op** controls the generation of an instance of the three consulting operations for each application concept in the CO. This rule is similar to the rule **cio**, belonging to the ruleset **creating_instance**. The action part of the rule consists of thirteen statements. These statements are responsible for creating instances of the consulting operations for each CO concept in the case level together with the WM objects representing them. The first three statements are involved in the construction of the instances of the operation **CONSULT_CONCEPT_INSTANCES_O**. The first statement generates the name of the instance, the second generates the instance in the CO case level and the third statement fills the facet **con** (representing the argument **con**). The next three statements perform the same functions for generating an instance of the **CONSULT_CONCEPT_CLASSES_O**: the fourth statement generates the name, the fifth creates the instance and the sixth fills the facet **con**. The next three statements carry out the same functions for creating an instance of the **CONSULT_ALL_CONCEPT_INSTANCES_O**. The ninth, tenth and eleventh statements create the three objects representing the generated instances in the WM. Finally, the last statement deletes the conceptual object matched.

Once all instances of the consulting operations have been created, the ruleset **next_ca** activates the next ruleset to be applied, the ruleset **consulting_attribute**.

```
(rule ci_op
  ruleset consulting_instance
  priority 1
  control forever
  (object ^con ?con)
  ->
  (?cio := (create-name 'cio ?con))
  (?concioi := (create-object ?cio 'cio))
  (?opargi := (add-slots ?cio '((con ?con) )))
  (?cco := (create-name 'cco ?con))
  (?concioc := (create-object ?cco 'cco))
  (?oparg := (add-slots ?cco '((con ?con) )))
  (?callio := (create-name 'callio ?con))
  (?concioc := (create-object ?callio 'callio))
  (?oparg := (add-slots ?callio '((con ?con) )))
  (create oconsi ^name ?cio ^con ?con)
  (create oconsc ^name ?cco ^con ?con)
  (create oconsalli ^name ?callio ^con ?con)
  (delete 1))

(rule next_ca
  ruleset consulting_instance
  priority 3
  control one
  ->
 (apply-ruleset consulting_attribute))
```

*Figure 6.19: The ruleset **consulting_instance***

The ruleset consulting_attribute

The ruleset **consulting_attribute** is represented in Figure 6.20. This ruleset is responsible for generating instances of the simple operations for consulting the attributes of the concepts in the CO.

The rule **ca_op** creates one instance of the operation for consulting the value of an attribute for each conceptual attribute in the CO. Operations consulting a conceptual attribute are organized as subclasses of the **CONSULT_ATTRIBUTE_O** according to the class of the attribute. As shown in Figure 6.20, the condition set of the rule states: the rule name, the ruleset name, the rule priority and the conditions over the WM objects according to which the rule must be applied. In this rule the control type is **forever,** which states the application of the rule while the conditions over the WM objects are satisfied. The condition on rule application is a WM object representing a concept attribute.

```
(rule ca_op
  ruleset consulting_attribute
  priority 2
  control forever
  (attrcon ^attrclass ?attrclass ^con ?con ^attr ?attr )
  ->
  (?opclass := (create-name 'ca ?attrclass))
  (?opgen := (create-name ?opclass '(?con ?attr)))
  (?op := (create-object ?opgen ?opclass))
  (?oparg := (add-slots ?opgen '((con ?con) (attr ?attr) )))
  (?role := (get-slot-value ?attrclass 'es_un))

  (create argop ^attrclass ?attrclass ^name ?opgen ^attr ?attr)
  (create operation ^opclass ?attrclass ^name ?opclass ^mod intsent)
  (delete 1))


(rule catid_op
  ruleset consulting_attribute
  priority 1
  control forever
  (cfat ^con ?con ^whos ?whos ^whob ?whob ^whato ?whato ^pcc ?pcc)
  ->
  (?opcatran := (create-name 'catid ?con))
  (?op := (create-object ?opcatran 'cat))
  (?oparg := (add-slots ?opcatran '((con ?con)(?whos)(?whob)(?whato)(pcc ?pcc))))

  (create argtrop ^op ?opcatran ^con ?con ^whos ?whos ^whato ?whato ^whob ?whob)
   (delete 1))


(rule cad_op
  ruleset consulting_attribute
  priority 2
  control forever
  (cfat ^con ?con ^whos ?whos ^whato ?whato ^pcc ?pcc)
  ->
  (?opcatran := (create-name 'catd ?con))
  (?op := (create-object ?opcatran 'fat))
  (?oparg := (add-slots ?opcatran '((con ?con)(?whos)(?whato)(pcc ?pcc))))

  (create argtrop ^op ?opcatran ^con ?con ^whos ?whos ^whato ?whato)
  (delete 1))


(rule next_grammarc
  ruleset consulting_attribute
  priority 3
  control one
  ->
  (apply-ruleset grammar))
```

*Figure 6.20: The ruleset **consulting_attribute***

The action part of the rule **ca_op** consists of seven statements. In the first statement, the result of the function **create-name**, concatenating the name of the operation **ca** to the

attribute class (represented in the variable **?attrclass**), is assigned to the variable **?opclass**. The second concatenates the name resulting in previous statement to the concept and attribute identifier. The third statement creates an instance of the CO operation **?opclass** (a subclass of the operation **ca**) by calling the predicate **create-object.** The fourth statement calls the **add-slots** function to fill the attributes **con**, **attr** and **val** of the instance generated. The fifth and sixth statements represent the operation instance generated as a WM object. The WM object generated in the fifth statement represents information necessary to generate the LO objects corresponding to the parameters and that generated in the sixth is used when generating the LO clause expressing the operation. Finally, the last statement deletes the WM object representing the attribute of the CO concept.

The rules **catid_op**, **cad_op** and **cao_op** generate instances of operations consulting one attribute of concept that can be expressed by transitive clauses. The expression of these operations includes the description of other conceptual attributes belonging to the classes **WHO_SUBJECT**, **WHO_OBJECT** and **WHAT_OBJECT**. These three rules are similar to the rules **fatid_op**, **fati_op** and **fad_op** in the ruleset **filling_attribute**.

The ruleset **next_grammarc** is responsible for activating the next ruleset to be applied, the ruleset **grammar**.

The ruleset grammar

The ruleset **grammar** ensures the process of generating LO instances representing the CO operations created in the first step of the process. This ruleset has 14 rules, shown in Figure 6.21, Figure 6.22 and Figure 6.23. There are different rules for controlling the linguistic structures required for each operation.

The linguistic instances representing operations creating and consulting concepts as well as operations on more than one attribute are the same for all applications. For this reason, the linguistic structures representing these operations belong to the LO domain level. These linguistic objects are *marked* (their identifier is stored in the set **activerule**) when they are necessary for an application. The rules responsible for marking these linguistic objects are shown in Figure 6.21.

```
(rule cc_s
   ruleset grammar
   priority 1
   control one
   (modify)
   ->
    (?adcc := (add-to-value 'activerule 'value 'rcc))
    (?adroot := (add-to-value  'root 'value 'rcc)))

 (rule ci_s
   ruleset grammar
   priority 1
   control one
   (consult)
   ->
    (?adcc := (add-to-value 'activerule 'value 'rcic))
    (?adroot := (add-to-value 'root 'value 'rcic)))

(rule cinn_s
   ruleset grammar
   priority 1
   control one
   (ocinn)
   ->
   (?adas := (add-to-value 'activerule 'value 'ac_ciwn)))

(rule ciwn_s
   ruleset grammar
   priority 1
   control one
   (ociwn)
   ->
   (?adas := (add-to-value 'activerule 'value 'ec_cinn)))

(rule ci
   ruleset grammar
   priority 1
   control one
   (oconsi)
   ->
   (?adas := (add-to-value 'activerule 'value 'cic_ci )))

(rule cc
   ruleset grammar
   priority 1
   control one
   (oconsc)
   ->
   (?adas := (add-to-value 'activerule 'value 'cic_cc)))

(rule allc
   ruleset grammar
   priority 1
   control one
   (oconsalli)
   ->
   (?adas := (add-to-value 'activerule 'value 'pic_cc)))
```

*Figure 6.21: Rules of the ruleset **grammar***

The rule **cc_s** (shown in Figure 6.21) marks linguistic objects representing the expression of more than one operation creating and modifying instances at the CO case level. The rule **ci_s** marks linguistic objects representing the expression of more than one operation consulting concepts. The linguistic objects representing the realization of operations creating conceptual instances are obtained by the rule **cinn_s** (for instances with no name) and the rule **ciwn_s** (for instances with name). The rules **ci**, **cc** and **allc** are responsible for obtaining the linguistic objects expressing operations consulting ontology concepts.

The three rules in Figure 6.22 ensure the generation of the most appropriate linguistic instances for each of operations over a conceptual attribute created for an application. The rule **attr_c** controls the generation of different linguistic instances for the different subclasses of the **FILL_ATTRIBUTE_O** and the **CONSULT_ATTRIBUTE_O**. The expression of operations on concept attributes depends on the attribute class. For this reason, different operations have been created regarding the attribute class. The most appropriate linguistic instances associated with each operation are obtained from the attribute class (which is also considered in the class of the operation). The description of the classes of attributes includes a facet (**decsent**) indicating the linguistic structures expressing the filling of the attributes in this class, and a facet (**insent**) for linguistic structures associated with their consulting.

For each operation subclass generated in the first step of the process, the rule **attr_c** obtains the linguistic instances necessary to express this operation in a clause. It considers two possible expressions for each operation: one corresponds to a referential clause and one to a non-referential clause. The constituents of the linguistic instances generated are also obtained from the information represented in the facets. Two types of constituents are distinguished: open and closed.

```
(rule attr_c
   ruleset grammar
   priority 2
   control forever
   (operation ^opclass ?class ^name ?opgen ^mod ?mod)
   ->
   (?assclass := (get-slot-value ?class ?mod))
   (?upclass := (first ?assclass))
   (?attro := (second ?assclass))
   (?attrc := (third ?assclass))

   (?cfa := (create-name ?upclass  ?class))
   (?claufa := (create-object ?cfa ?upclass))
   (?catr := (add-slots ?cfa '((oper ?opgen))))

   (?rupclass := (create-name 'r ?upclass))
   (?rcfa := (create-name ?rupclass ?class))
   (?clauserfa := (create-object ?rcfa ?rupclass))
   (?cattr := (add-slots ?rcfa '((oper ?opgen))))

   (?adc := (add-to-value 'activerule 'value ?cfa ))
   (?adrc := (add-to-value 'activerule 'value ?rcfa ))
   (delete 1)
   (create opattr ^c ?cfa ^upc ?upclass ^opclass ?class ^attr ?attro)
   (create opattr ^c ?rcfa ^upc ?upclass ^opclass ?class ^attr ?attro)
   (create clattr ^c ?cfa ^attr ?attrc)
   (create clattr ^c ?rcfa ^attr ?attrc))


(rule add_opconstituents
   ruleset grammar
   priority 3
   control forever
   (opattr ^c ?cfa ^upc ?upclass ^opclass ?class ^attr (?constituent *rest))
   ->
   (?cons := (car (get-value ?upclass ?constituent)))
   (?conclass := (create-name ?cons ?class))
   (?sconst := (add-slots ?cfa '((?constituent ?conclass))))
   (delete 1)
   (create opattr ^c ?cfa ^upc ?upclass ^opclass ?class ^attr (*rest)))


(rule add_clconstituents
   ruleset grammar
   priority 3
   control forever
   (clattr ^c ?cfa  ^attr ((?constituent ?value) *rest))
   ->
   (?sconst := (add-slots ?cfa '((?constituent ?value))))
   (delete 1)
   (create clattr ^c ?cfa ^attr (*rest)))
```

*Figure 6.22: Rules of the ruleset **grammar***

The condition over the rule **attr_c** is a WM object representing an instance of an operation over a conceptual attribute. The first statement in the action part of the rule obtains the linguistic information related to the most appropriate form of expressing the operation over the attribute. This information is obtained from the attribute class. The variable **?mod**

indicates if the operation over the attribute is to fill it (the value is **decsent)** or to consult (the value is **intsent**). The second statement obtains the most appropriate LO class representing the operation over the attribute. The third and fourth statements obtain the constituents associated with the class. The third statement obtains the open constituents (i.e. nouns and verbs). The fourth statement obtains the close constituents (i.e. prepositions and interrogative pronouns). The next three statements generate an instance of the linguistic class obtained in the first statement: one generates the instance name, the next creates the instance in the LO and the third adds the operation identifier to the instance generated. Then, the next four statements generate an instance of the same linguistic class represented in referential form. The next two statements mark the two linguistic instances generated as active. Then, the delete statement deletes the WM object over which the rule has been applied. Finally, the last four statements create the WM necessary to create the linguistic structures representing the constituents of the generated linguistic instance.

The rules **add_opconstituents** and **add_clconstituents** are responsible for adding the constituents in the linguistic instances generated. The operation parameters are represented as constituents in these instances generated. The category associated with each constituent represents the type of the argument (**con**, **attr**, **val**) and the class (e.g. **defngattrof**, **defngvalwho_does**). In the linguistic instances generated, information about the pattern or correct distribution of the constituents and the syntactic and semantic agreement between them is inherited from their linguistic classes.

Figure 6.23 shows the other rules in the ruleset **grammar**. The rule **fa_pt** marks the linguistic objects representing operations filling more than one instance attribute expressed by transitive clauses. The rule **ca_pt** marks the linguistic objects representing interrogative transitive clauses expressing operations consulting more than one instance attribute. The rule **ind_ng** marks the indirect nominal groups expressing concepts. The rule **next_arguments** activates the next ruleset to apply, the ruleset **arguments**.

```
(rule fa_pt
  ruleset grammar
  priority 4
  control one
  (modify)
  (argtrop)
  ->
  (?tc := (add-to-value 'activerule 'value 'tc_fma ))
  (?rtc := (add-to-value 'activerule 'value 'rtc_fma ))
  (?indc := (add-to-value 'activerule 'value 'indtc_fma ))
  (?indrc := (add-to-value 'activerule 'value 'rindtc_fma )))

(rule ca_pt
  ruleset grammar
  priority 4
  control one
  (consult)
  (argtrop)
  ->
  (?tc := (add-to-value 'activerule 'value 'itc_cma ))
  (?rtc := (add-to-value 'activerule 'value 'ritc_cma ))
  (?indc := (add-to-value 'activerule 'value 'iindtc_cma ))
  (?indrc := (add-to-value 'activerule 'value 'riindtc_cma )))
(rule ind_ng
  ruleset grammar
  priority 4
  control one
  ->
  (?ading := (add-to-value 'activerule 'value 'indngcon))
  (?ading := (add-to-value 'activerule 'value 'indngconi)))

(rule next_arguments
  ruleset grammar
  priority 5
  control one
  ->
  (apply-ruleset arguments))
```

*Figure 6.23: Rules of the ruleset **grammar***

The alternative set of rules only differs from this basic set in that it considers all linguistic instances associated with the expression of each operation. That is, there are two alternative rules obtaining all linguistic instances associated with the class of the attribute involved in the operation. The two new rules incorporated in the alternative set are shown in Figure 6.24.

```
(rule attr_tc
   ruleset grammar
   priority 2
   control forever
   (operation ^opclass ?class ^name ?opgen ^mod ?mod)
   ->
   (?asclass := (get-immediate-value ?class ?mod))
   (create cl ^op ?opgen ^class ?class ^asclass ?asclass)
   (delete 1))

(rule attr_uc
   ruleset grammar
   priority 3
   control forever
   (cl ^op ?op ^class ?class ^asclass (?asclass *rasclass))
   ->
   (?upclass := (first ?asclass))
   (?attro := (second ?asclass))
   (?attrc := (third ?asclass))

   (?cfa := (create-name ?upclass  ?class))
   (?claufa := (create-object ?cfa ?upclass))
   (?catr := (add-slots ?cfa '((oper ?op))))

   (?rupclass := (create-name 'r ?upclass))
   (?rcfa := (create-name ?rupclass ?class))
   (?clauserfa := (create-object ?rcfa ?rupclass))
   (?cattr := (add-slots ?rcfa '((oper ?op))))

   (?adc := (add-to-value 'activerule 'value ?cfa ))
   (?adrc := (add-to-value 'activerule 'value ?rcfa ))
   (delete 1)

   (create opattr ^c ?cfa ^upc ?upclass ^opclass ?class ^attr ?attro)
   (create opattr ^c ?rcfa ^upc ?upclass ^opclass ?class ^attr ?attro)
   (create clattr ^c ?cfa ^attr ?attrc)
   (create clattr ^c ?rcfa ^attr ?attrc)

   (create cl ^op ?op ^class ?class ^asclass (*rasclass)))
```

*Figure 6.24: The alternative rules in the ruleset **grammar***

The ruleset arguments

The ruleset **arguments** is responsible for creating WM objects representing all semantic and syntactic information necessary for generating the terminal linguistic instances corresponding to the arguments of operations created. Rules in this ruleset generate WM objects representing the concepts, attributes and values of the operation instances generated in the first step of the process. These rules obtain the linguistic information associated with each operation argument from the linguistic description of the application terms and the semantic information from the operation instance.

The rules generating the WM objects representing the parameter **con** (representing the concept) of the simple operations creating and consulting concepts are shown in Figure 6.25.

```
(rule crcon_ng
   ruleset arguments
   priority 1
   control forever
   (ocinn  ^name ?op ^con ?con ^pcobject ?pcc)
   ->
   (?adpcc := (add-to-value 'precobject 'valor (cons ?con ?pcc)))
   (?lexcon := (get-immediate-value ?con 'lex))
   (create con ^op ?op ^sem ?con ^lex ?lexcon)
   (delete 1))

(rule coni_ng
   ruleset arguments
   priority 1
   control forever
   (ociwn  ^name ?op ^con ?con)
   ->
   (?lexcon := (get-slot-value ?con 'lex))
   (?lexdic := (get-immediate-value ?lexcon 'inf))
   (?nomins := '(name))
   (create termdyn ^op ?op ^arg (ins ins) ^sem ?nomins ^lex ?lexdic)
   (create termdyins ^op ?op ^sem ?con)
   (delete 1))

(rule cocon_ng
   ruleset arguments
   priority 1
   control forever
   (oconsc  ^name ?op ^con ?con )
   ->
   (?lexcon := (get-immediate-value ?con 'lex))
   (create con ^op ?op ^sem ?con ^lex ?lexcon)
   (delete 1))

(rule cocoi_ng
   ruleset arguments
   priority 1
   control forever
   (oconsi  ^name ?op ^con ?con )
   ->
    (?lexcon := (get-slot-value ?con 'lex))
    (?lexdic := (get-immediate-value ?lexcon 'inf))
    (?nomins := '(name))
    (create termdyn ^op ?op ^arg (ins ins) ^sem ?nomins ^lex ?lexdic)
    (create termdyins ^op ?op ^sem ?con)
    (delete 1))
```

Figure 6.25: Rules of the ruleset arguments

The rule **crcon_ng** generates a WM object representing the parameter **con** (concept) from each operation creating a conceptual instance with no name. The semantic information associated with these objects is the concept identifier. The preconditions associated with these concepts are represented in the object **precobject**. These preconditions will be consulted at run-time to activate the appropriate lexical entries and grammar rules at each state of the communication. Next, the rule **con_ng** will use the WM objects created by this rules to generated a different WM for each pointer to the application terms included in the concept description (represented in the facet **lex**).

The rule **coni_ng** generates WM objects corresponding to the parameter **ins** of the **CREATE_INSTANCE_WITH_NAME_O**, representing the name of the instances of concepts that will be created by these operations. As this name will be set by the user at run-time, the semantic interpretation associated with these objects is the name of a function (a PROLOG predicate) responsible for requesting the user to introduce the name of the instance.

The rule **cocon_ng** creates WM objects representing the parameter **con** of the operation by consulting whether there is a conceptual class in the CO. The rule **cocoi_ng** creates WM objects representing the parameter **ins** of the operation by consulting whether there is a conceptual instance in the CO case level.

Rules responsible for generating the WM objects representing the parameters of operations to fill and consult the values of the concept attributes are shown in Figure 6.26. The rule **cofat_o** generates a WM object representing the parameters **attr** (attribute) and **val** (value) of the simple operations filling and consulting one concept attribute.

The syntactic and semantic information associated with concepts and attributes is obtained by other rules in this ruleset, shown in the Figure 6.26. Figure 6.27 shows rules obtaining this information for the different types of values (yes/no value, function value, function and associated unit value and instance value).

```
(rule fat_o
   ruleset arguments
   priority 1
   control forever
   (argop ^attrclass ?class ^name ?opgen ^attr ?attr)
   ->
   (?lexattr := (get-immediate-value ?attr 'lex))
   (?range := (get-slot-value ?attr 'range))
   (?card := (get-slot-value ?attr 'cardinality))
   (create attr ^opclass ?class ^op ?opgen ^sem ?attr ^lex ?lexattr)
   (create val ^opclass ?class ^op ?opgen ^range ?range ^card ?card)
   (delete 1))
  (rule fatid_o
   ruleset arguments
   priority 1
   control forever
   (argtrop ^op ?op ^con ?con ^whos ?whos ^whob ?whob ^whato ?whato )
   ->
   (?lexdic := (get-immediate-value ?con 'lexverb))
   (?semverb := (create-lambda-function ?con '(?whos ?whob ?whato)))
   (create con ^op ?op ^sem ?semverb ^lex ?lexdic)
   (delete 1))
(rule fati_o
   ruleset arguments
   priority 2
   control forever
   (argtrop ^op ?op ^con ?con ^whos ?whos ^whob ?whob )
   ->
   (?lexdic := (get-immediate-value ?con 'lexverb))
   (?semverb := (create-lambda-function ?con '(?whos ?whob)))
   (create con ^op ?op ^sem ?semverb ^lex ?lexdic)
   (delete 1))
(rule fatd_o
   ruleset arguments
   priority 2
   control forever
   (argtrop ^op ?op ^con ?con ^whos ?whos ^whato ?whato )
   ->
   (?lexdic := (get-immediate-value ?con 'lexverb))
   (?semverb := (create-lambda-function ?con '(?whos ?whato)))
   (create con ^op ?op ^sem ?semverb ^lex ?lexdic)
   (delete 1))
(rule con_ng
   ruleset arguments
   priority 3
   control forever
   (con ^op ?op ^sem ?sem ^lex (?lex *rest))
   ->
   (?lexdic := (get-immediate-value ?lex 'inf))
   (create termlex ^op ?op ^arg (con con) ^sem ?sem ^lex ?lexdic)
   (create con ^op ?op ^sem ?sem ^lex (*rest))
   (delete 1))
(rule attr_ng
   ruleset arguments
   priority 3
   control forever
   (attr ^opclass ?class ^op ?opgen ^sem ?attr ^lex (?lex *rest))
   ->
   (?attrty := (create-name 'attr ?class))
   (?lexdic := (get-immediate-value ?lex 'inf))
   (create termlex ^op ?opgen ^arg (?attrty attr) ^sem ?attr ^lex ?lexdic)
   (create attr ^opclass ?class ^op ?opgen ^sem ?attr ^lex (*rest))
   (delete 1))
```

Figure 6.26: Rules of the ruleset arguments

```
(rule valyn
   ruleset arguments
   priority 3
   control forever
   (val ^opclass ?class ^op ?op ^range yes/no)
   ->
   (delete 1))
(rule valset
   ruleset arguments
   priority 3
   control forever
   (val ^opclass ?class ^op ?op ^range (cj ?val) ^card ?card)
   ->
   (?valty := (create-name 'val ?class))
   (?lexdic := (get-immediate-value ?val 'inf))
   (?insf := (create-name 'instance ?card))
   (?semval := '(?insf (?val)))
   (create termdyn ^op ?op ^arg (?valty val) ^sem ?semval ^lex ?lexdic)
   (delete 1))
(rule valdynunit
   ruleset arguments
   priority 3
   control forever
   (val ^opclass ?class ^op ?op ^range (?dyn ?val ?unit) ^card ?card)
   ->
   (?valty := (create-name 'val ?class))
   (?lexval := (get-slot-value ?val 'lex))
   (?lexdic := (get-immediate-value ?lexval 'inf))
   (?dynf := (create-name ?func ?card))
   (?semval := '(?dynf (?unit)))
   (create termdyn ^op ?op ^arg (?valty val) ^sem ?semval ^lex ?lexdic)
   (delete 1))
(rule valmenu
   ruleset arguments
   priority 3
   control forever
   (val ^opclass ?class ^op ?op ^range (menu ?name ?val) ^card ?card)
   ->
   (?valty := (create-name 'val ?class))
   (?lexdic := (get-immediate-value ?name 'inf))
   (?funcf := (create-name 'menu ?card))
   (?semval := '(?funcf (?name)))
   (create termdyn ^op ?op ^arg (?valty val) ^sem ?semval ^lex ?lexdic)
   (delete 1))
(rule valfunction
   ruleset arguments
   priority 4
   control forever
   (val ^opclass ?class ^op ?op ^range (?func ?val) ^card ?card)
   ->
   (?valty := (create-name 'val ?class))
   (?lexdic := (get-immediate-value ?val 'inf))
   (?funcf := (create-name ?func ?card))
   (?semval := '(?funcf (?val)))
   (create termdyn ^op ?op ^arg (?valty val) ^sem ?semval ^lex ?lexdic)
   (delete 1))
```

Figure 6.27: Rules of the ruleset arguments

The ruleset lexical_entries

The ruleset **lexical_entries** ensures the creation of linguistic instances representing all possible realizations of the operation parameters. These linguistic instances incorporate syntactic and semantic information associated with each. This information consists of the surface realization, the category, the semantic interpretation, the general linguistic class the category belongs to and the type of instance (lexical or dynamic). The semantic information, that is, the semantic restrictions associated with the category and the semantic interpretation, is obtained from the operation argument. The syntactic information, the category, the syntactic restrictions (number and gender) and the string, is obtained from the set containing the linguistic information of the application terms by the ruleset **arguments**. These instances represent the interface lexical entries.

The ruleset **lexical_entries** contains three rules: the rule **lexentry**, the rule **dynentry** and the rule **dynentryins**. The three rules are shown in Figure 6.28. The rule **lexentry** performs the generation of lexical instances. The surface realization of these instances is obtained from the syntactic description of the application terms. The rule **dynentry** is responsible for generating the instances representing arguments associated with a dynamic predicate. The rule **dynentryins** controls the generation of conceptual instances existing in the CO case level.

As can be seen in the figure, the WM objects representing all semantic and syntactic information associated with an operation argument are used by the two rules to create the linguistic instances.

The semantic information represented in the WM object consists of the operation indentifier (**op**), the type of argument (**arg**) and the semantic interpretation associated with the parameter (**sem**). The syntactic information is represented in the attribute **lex**. It consists of the syntactic category and, for lexical objects, the surface realization and the syntactic features. It is obtained from the entries in the term description set associated with an argument.

```
(rule lexentry
   ruleset lex_entries
   priority 1
   control forever
   (termlex ^op ?op ^arg (?arg ?targ) ^sem ?sem ^lex ((?cat ?str ?synf) *rest))
   ->
   (?upsemcat := (create-name ?cat ?targ))
   (?semr := (get-slot-value ?upsemcat 'semf))
   (?semf := (obtain-semfeatures ?semr ?op))
   (?semcat := (create-name ?cat ?arg))
   (?catfi := '(?semcat '(?synf ?semf)))
   (?aux := (create-object ?semcat ?upsemcat))
   (?lexname := (create-name ?op ?str))
   (?termf := (create-object ?lexname ?semcat))
   (?tfslotg := (add-slots ?lexname '((type 'lex)
            (cat ?catfi)(lex ?str)(sem ?sem))))
   (?adcl := (add-to-value 'activelex 'value ?lexname))
   (delete 1)
   (create termlex ^op ?op ^arg (?arg ?targ) ^sem ?sem ^lex (*rest)))
(rule dynentry
   ruleset lex_entries
   priority 1
   control forevee
   (termdyn ^op ?op ^arg (?arg ?targ)  ^sem ?val ^lex ((?cat *str) *rest))
   ->
   (?upsemcat := (create-name ?cat ?targ))
   (?semr := (get-slot-value ?upsemcat 'semf))
   (?semf := (obtain-semfeatures ?semr ?op))
   (?semcat := (create-name ?cat ?arg))
   (?catfi := '(?semcat '(?semf)))
   (?aux := (create-object ?semcat ?upsemcat))
   (?lexdin := (create-name ?semcat ?op ))
   (?termf := (create-object ?lexdin ?semcat))
   (?tfslotg := (add-slots  ?lexdin '((type 'dyn)(cat ?catfi)(lex ?val))))
   (?adcl := (add-to-value 'activelex 'value ?lexdin))
   (delete 1)
   (create termdyn ^op ?op ^arg (?arg ?targ) ^sem ?val ^lex (*rest) ))
(rule dynentryins
   ruleset lex_entries
   priority 1
   control forever
   (termdyins ^op ?op ^sem ?con )
   ->
   (?semf := (obtain-semfeatures '(con) ?op))
   (?catfi := '(pngi '(?semf)))
   (?lexdin := (create-name ?con ?op ))
   (?termf := (create-object ?lexdin 'pngi))
   (?val := '(instance (?con)))
   (?tfslotg := (add-slots  ?lexdin '((type 'dyn)(cat ?catfi)(lex ?val))))
   (?adcl := (add-to-value 'activelex 'value ?lexdin))
   (delete 1))
```

Figure 6.28: Rules of the ruleset lex_entries

The statements in the action part of the rule **lexentry**, the rule **dynentry** and the rule **dynentryins** generate an LO instance encoding all necessary information about the operation argument. The category of these linguistic instances is augmented with syntactic and semantic features.

These rules generate a linguistic instance for each possible realization of the operation parameter. These instances represent the interface lexical entries.

6.3 FOLLOWING AN EXAMPLE

A simplified example of the performance of the basic set of control rules is described below.

As discussed earlier, the grammars generated by the two basic sets of rules represent queries and descriptions about the application concepts represented in the CO. For example, if **ARCHITECT** is a CO concept, several ways to state the existence of a particular architect are supported by the grammar generated by GISE. Clauses such as *Existe un arquitecto* (*there is an architect*) and *<Nombre Propio> es un arquitecto (<Proper Noun> is an architect*) can be introduced to express the creation of an instance of the concept **ARCHITECT**. In the first case, the system automatically gives a new name to the instance, while in the second case, *<Proper Noun>* will be the name of the instance generated.

The process of generating the grammar rules and lexical entries necessary for accepting the expression of this operation in Spanish is described next.

The concept **ARCHITECT**, belonging to the CO in the SIREDOJ application, is shown in Figure 6.29. Three facets describe the concept: **isa**, **interface_entity** and **lex**. The facet **isa** indicates that all members in the class belong to the upper class **PERSON**. The facet **interface_entity** and its value **yes** indicate that the concept is expressed during communication between the user and the interface. The facet **lex** contains a list of pointers to the linguistic realizations of the concept contained in the application terms set. In this example, the only pointer is **arquitecto**.

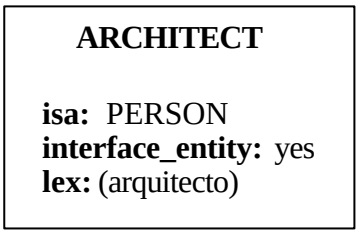

*Figure 6.29: The concept **ARCHITECT***

The process for obtaining the linguistic resources necessary to express the operation to create an instance of the concept **ARCHITECT** is divided into the three steps described bellow.

Step1

The process performed in this step is depicted in Figure 6.30.

The process starts when the basic set of rules described in the previous section is activated.

First, the function to initiate the process is called up. As described above, there is one different function for each type of interface considered. The function **inim** is called up to generate interfaces that only accept the incorporation of new information to enrich the CO, preventing its consultation. This is the case of the interface to SIREDOJ. The initialization function activates the ruleset TOP, shown in Figure 6.15. As described in Section 6.3.2, this ruleset checks the initial conditions indicating the type of interface that must be generated, and activates the appropriate ruleset. When the initialization function is **inim**, the only ruleset activated is the ruleset **creating_instance**, described in Figure 6.16.

In this first step, the existing concepts in the CO are related to the operations performing the creation of concept instances in the case level of the CO. These operations are the **CREATE_INSTANCE_WITH_NO_NAME_O**, performing the creation of instances whose name is generated automatically and the **CREATE_INSTANCE WITH_NAME_O**, creating instances whose name is specified in the operation call. Each of these two operations is represented as a CO object. The operation parameters, as well as the conditions governing its execution, are represented as facets. The facets describing these two operations are **ins**, representing the name of the instance, **con**, representing the concept identifier and **pcc**, representing the conditions governing the conceptual instance creation.

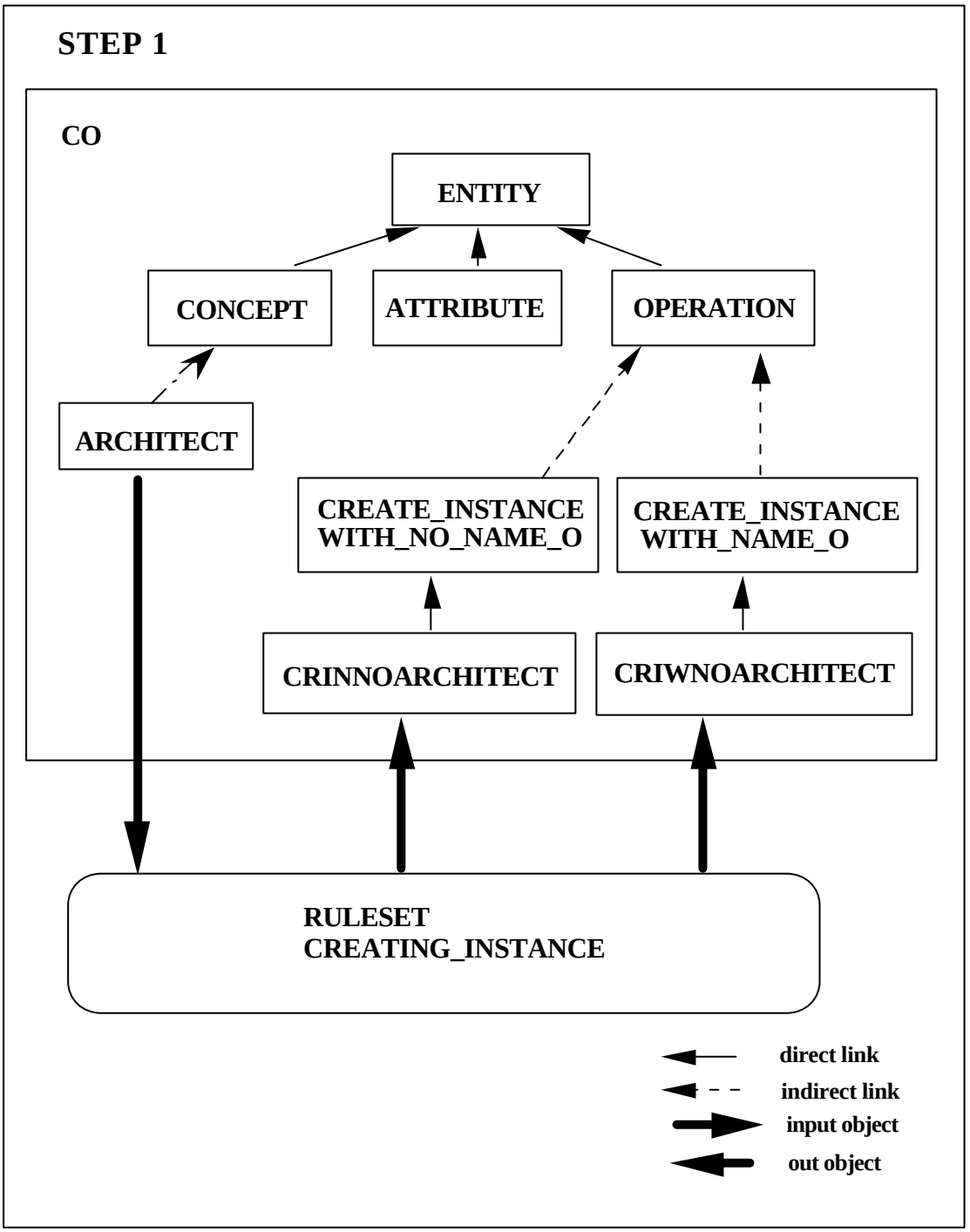

*Figure 6.30: The performance of the first step of the process for the concept **ARCHITECT***

The rule **cio**, belonging to the ruleset **creating_instance** generates the corresponding instances of the two operations mentioned above. If the concept **ARCHITECT** exists alone in the CO, then only one instance of these two operations will be generated[4].

[4]Because it is more efficient to access objects in the WM than objects in the ontology, when a rule generates ontology instances that are accessed by the following rules, it also creates the WM objects representing the instances. Initially, a set of metarules is responsible for the representation of the CO concepts as WM objects.

Figure 6.31 shows the instance of the **CREATE_INSTANCE_WITH_NO_NAME_O** operation generated for the concept **ARCHITECT**. This operation, **CRINNOARCHITECT**, creates an instance of the concept **ARCHITECT** and generates a new name that identifies the instance.

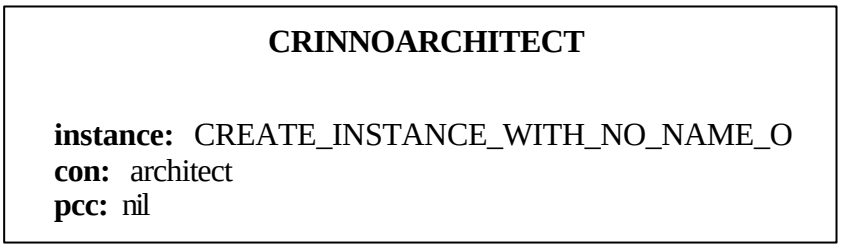

*Figure 6.31: The instance **CRINNOARCHITECT**, for creating an instance of the concept **ARCHITECT** without giving its name*

Figure 6.32 shows the operation **CRIWNOARCHITECT,** the instance of the **CREATE_INSTANCE_WITH_NAME_O** generated for the concept **ARCHITECT**.

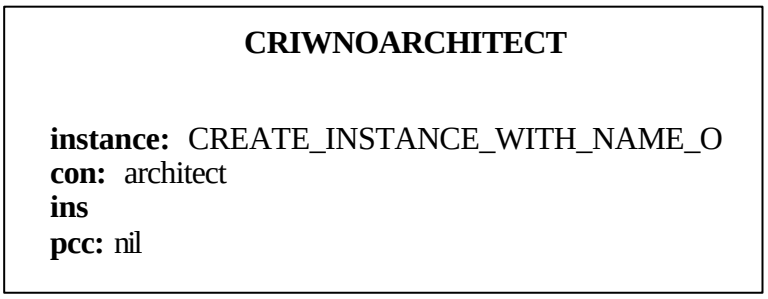

*Figure 6.32: The instance **CRIWNOARCHITECT**, for creating an instance of the concept **ARCHITECT** giving its name*

Of course, in a real situation, many concepts exist and thus the performance of the rule will iterate for each of these concepts, leading to the construction of other instances for the two operations. In the same way, other rules belonging to the ruleset **filling_attribute** will be fired in order to create the permitted operations for modifying the instance attributes. In this example, the concept **ARCHITECT** has no attributes to be filled by the user

Step 2

Figure 6.33 describes the second step of the process to obtaining the linguistic structures for creating instances of the concept **ARCHITECT**.

The second step of the process consists of mapping the instances of operations generated in the first step onto the corresponding LO objects. The first ruleset applied in this step is the ruleset **grammar**, described in Figures 6.20-6.22. This ruleset is responsible for obtaining the linguistic objects necessary to express the operations created in the first step.

Following the example, the identifiers of the LO objects representing the realization of the two operations creating conceptual instances are stored in the object **activerules**, indicating that they must be included in the grammar. These LO classes are the **EXISTENTIAL_CLAUSE_CREATE_INSTANCE_WITH_NO_NAME** class, shown in Figure 5.6 and the class **ATTRIBUTIVE_CLAUSE_CREATING_INSTANCE WITH_NAME**, shown in Figure 5.7.

Obtaining the linguistic structures for more complex operations, such as operations filling and consulting conceptual attributes, is not so direct. New linguistic instances must be generated according to the classes of the attributes to fill.

Once the ruleset **grammar** has been applied, the ruleset **arguments** and the ruleset **lexical_entries** are activated. These two rulesets are responsible for creating the linguistic instances expressing the parameters of the operations generated. First, the rules in the ruleset **arguments** are applied to the operation instances generated in the first step of the process. These rules create WM objects describing information associated with the parameters of the operations generated in Step 1. Then, the rules in the ruleset **lexical_entries** use these WM objects to create the LO instances supporting the expression of the parameters. The LO instances generated will be incorporated into the interface lexicon.

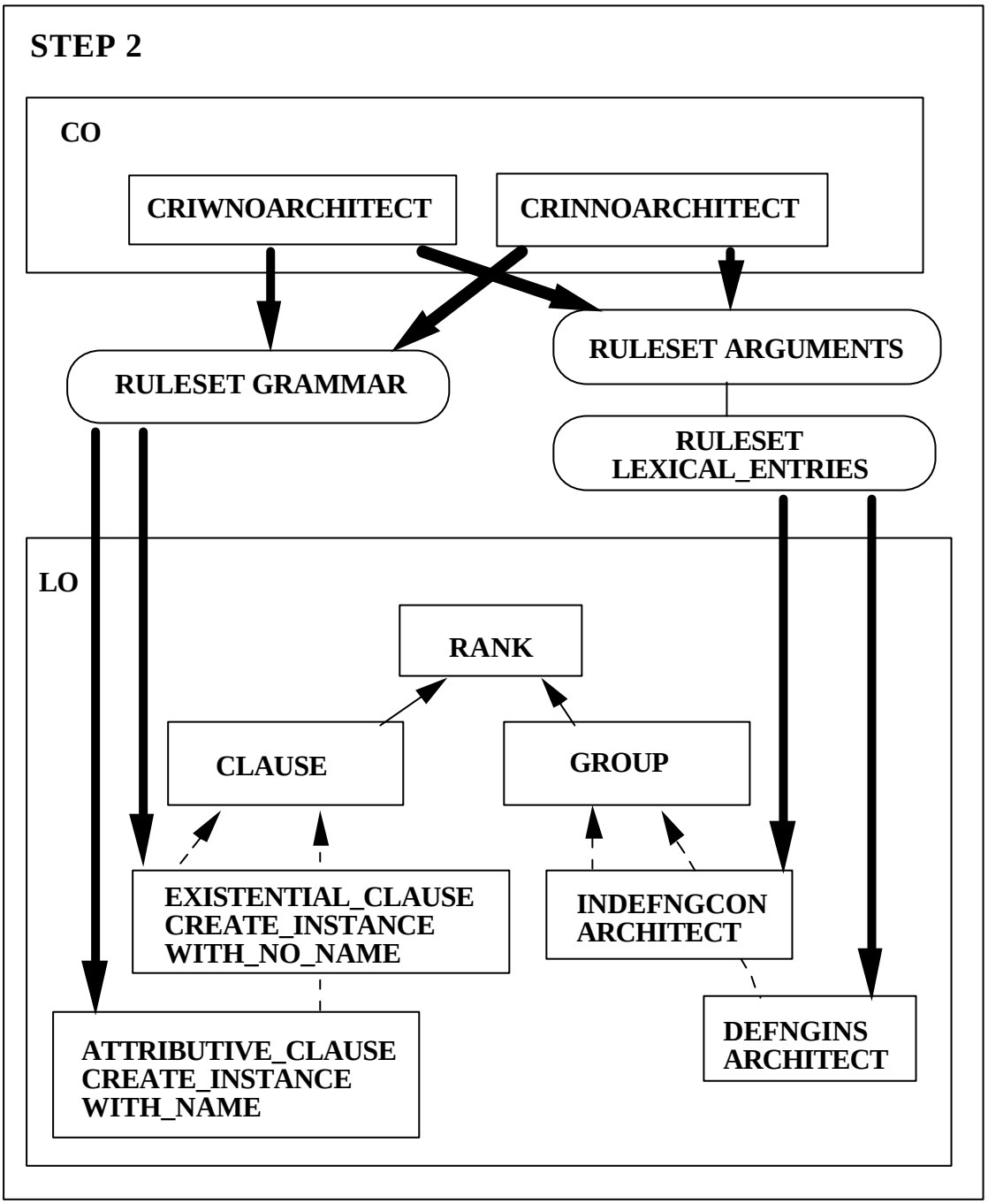

Figure 6.33: The performance of the second step of the process for the concept ***ARCHITECT***

The rules in the ruleset **arguments** create WM objects representing the syntactic and semantic information associated with the parameters of the operations generated. These parameters correspond to CO concepts, attributes and values. The syntactic information associated with these parameters is obtained from the facet **lex**, containing the pointers to the entries in the set of application terms. Each of these CO objects is related to one or more entries in this set. The syntactic information describing each application term consists

of category, superficial presentation and syntactic features.

The semantic information associated with the operation parameters consists of the semantic interpretation and the semantic features. This information is obtained from the object representing the operation.

The generated objects representing the operation parameters will be classified as lexical terms (**termlex**), dynamic terms (**termdyn**) and instance terms (**termdyins**). The lexical terms represent the arguments expressed by a word (or sequence of words) included in the lexicon generated for an application. The dynamic terms correspond to values that will be requested from the user during communication. An instance term represents all the instances of a specific concept existing in the CO case level at any state of the communication process.

The preconditions associated with these concepts will be incorporated into the object **precobject**. This object represents all preconditions associated with the concepts involved in the communication. These preconditions together with the preconditions associated with the operations will be incorporated into the grammar generated as predicates to be executed at run-time.

In our example, only three rules of the ruleset **argument** are applied. These rules are: **crcon_ng** , **coni_ng** and **con_ng**, described in Figure 6.25. The rule **crcon_ng** is applied to the instances of the operation creating conceptual instances whose name must be generated by the system. For each operation instance, this rule creates a WM object describing the information associated with the parameter **con**. This parameter represents the identifier of the concept (i.e. **architect**). The WM object created for this parameter corresponds to lexical terms, that is, this WM object will be used to generate the lexical entries with its superficial realizations. This WM object represents the syntactic information obtained from the application term's description and the concept and operation identifiers.

Following the example, the rule **crcon_ng** is applied to the operation **CRINNOARCHITECT** (for creating an instance with no name for the concept **ARCHITECT**). As a result, a WM object representing the parameter **con** will be generated. This WM object is described in Figure 6.34.

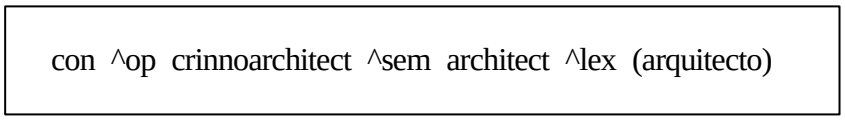

*Figure 6.34: The WM object representing the information associated with the parameter **con** of the operation **CRINNOARCHITECT***

As shown in this figure, the name of the WM object created is **con**. The attributes describing this object are **op**, **sem** and **lex**. The value of the attribute **op** represents the name of the operation, **crinnoarchitect**. The attribute **sem** represents the semantic interpretation of the parameter **architect**. Finally, the attribute **lex** represents the pointers to the entries in the set of application terms. In this example, there is only onepointer: **arquitecto**.

The rule **con_ng** is applied to the WM objects created by the **crcon_ng**. For all pointers associated with a conceptual description, this rule creates a WM object. The attributes describing the resulting object are **op**, **sem**, **arg** and **lex**. The value of the attribute **arg** consists of a list of two words representing the type of parameter. One word represents the name of the parameter (**con**, **ins**, **attr** or **val**). The other word represents the name of the parameter and, in the case of it being an attribute or value, the basic class to which the attribute belongs (i.e. **attris**, **valdoes**). In this example, the value of the attribute **arg** is **(con con)**. The attribute **lex** in the resulting object represents all syntactic information associated with one of the pointers included in the concept description. This information consists of all possible realizations related with an entry in the set of application terms and the corresponding syntactic category. In this example, there are two different realizations of the concept: one represents the definite nominal group, *el arquitecto* (*the architect*); the other represents the indefinite nominal group, *un arquitecto* (*an architect*).

The rule **coni_ng** is applied to the operations generated for creating instances with a given name. The two parameters of these operations are: **con**, representing the name of the concept and **ins**, representing the name of the instance. The value of the parameter **con** is the same as that of the parameter **con** in the operations creating a conceptual instance, whose name is generated by the system. Because the rule **crcon_ng** is performed over the parameter **con**, the rule **coni_ng** only considers the parameter **ins**. For all instances of the **CREATE_INSTANCE_WITH_NAME_O**, this rule creates a WM object describing the information associated with the parameter **ins**. The objects created correspond to dynamic terms; they represent the conceptual instance name introduced by the user at run-time. The

name of the WM objects created for this rule is **termdyn**. These objects are described by the same attributes describing the lexical terms: **op**, **arg**, **sem** and **lex**. In this case, the value of the attribute **sem** is the function that will request the user to introduce the name of the instance at run-time.

In our example, the rule **coni_ng** will be applied on the operation **CRIWNOARCHITECT** (for creating an instance of the concept **ARCHITECT** giving it a specific name). The WM object generated to represent the parameter **ins** of this operation is shown in Figure 6.35.

The next ruleset applied is **lexical_entries**, responsible for generating the linguistic instances in the LO representing the parameters of the operations generated in the first step. This ruleset, described in Figure 6.28, is applied to the WM objects created by the ruleset **arguments**.

```
termdyn ^op criwnoarchitect ^arg (ins ins) ^val name
        ^lex ((defng el_aquitecto (syn (gen m) (num s)))
              (indefng un_arquitecto (syn (gen m) (num s))))
```

*Figure 6.35: The WM object representing syntactic and semantic information associated with the parameter **ins** of the operation **CRIWNOARCHITECT***

As shown in Figure 6.28, this ruleset contains three rules: the rule **lexentry**, the rule **dynentry** and the rule **dynentryins**. The rule **lexentry** performs the generation of instances representing operation arguments whsoe surface realization is obtained from the description of the application terms. The rule **dynentry** is in charge of generating the instances representing arguments associated with a dynamic function. The rule **dynentryins** controls the generation of the instances for arguments representing all existing instances of a specific concept at run-time.

As can be seen in Figure 6.28, the WM objects representing all semantic and syntactic information associated with an operation argument are used to create the corresponding linguistic instances. The rule **lexentry** is applied to WM objects representing lexical terms, the rule **dynentry** to WM objects representing dynamic terms and the rule **dynentryins** to WM objects describing conceptual instances.

All LO instances representing a word (or a sequence of words) are described by the facets **cat** (category), **lex** (superficial presentation), **sem** and **type**. The facet **cat** represents the category associated with linguistic objects and is augmented with syntactic and semantic features. The facet **lex** is a word (or words) representing the superficial form. The semantic interpretation is represented in the facet **sem**. The attribute **type** indicates whether the entry represents a lexical or dynamic entry. Dynamic instances, those associated with a dynamic function, are described by the facets **cat**, lex (representing the function identifier) and **type**.

Following the example, the rule **lexentry** is applied to the WM object representing the parameter architect of the operation for creating an instance with no name of the concept **ARCHITECT**. This WM object is used to create two instances representing the expression of the parameter in Spanish: one instance for the indefinite nominal group realized as *un arquitecto* (*an architect*), and the other for the definite nominal group *el arquitecto* (*the architect*). The first instance is described in Figure 6.36 and the second in Figure 6.37. These two instances correspond to the two realizations associated with the concept.

INDEFNGCONARCHITECT

instance: INDEFINITE_NOMINAL_GROUP_CONCEPT
type: lex
cat: indefngcon(syn(gen(m), num(s)), sem(con(architect)))
lex: un_arquitecto
sem: architect

Figure 6.36: The indefinite nominal group representing the concept ARCHITECT

DEFNGCONARCHITECT

instance: DEFINITE_NOMINAL_GROUP_CONCEPT
type: lex
cat: defngcon(syn(gen(m), num(s)), sem(con(architect)))
lex: el_arquitecto
sem: architect

Figure 6.37: The definite nominal group representing the concept ARCHITECT

All syntactic information (category, string and syntactic features) is obtained from the attribute **lex** of the WM object. In this example, both entries are associated with the syntactic feature **gen** (gender), whose value is **m** (masculine) and the feature **num** (number), whose value is **s** (singular). The semantic information associated with these two instances consists of the semantic feature **con** and the semantic interpretation, both representing the concept identifier **architect**.

When applying the rule **dynentry** to the WM object shown in Figure 6.35, two LO instances are created. They represent the name introduced by the user to identify the instances of the concept **ARCHITECT** generated. This name will be asked from the user at run-time. One of these two instances corresponds to the indefinite nominal group and the other instance represents the definite nominal group. The linguistic instance representing the definite nominal group is shown in Figure 6.38.

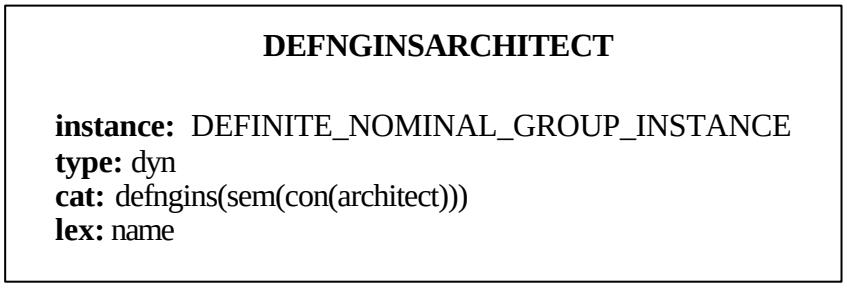

Figure 6.38: The definite nominal group representing an instance of the concept ARCHITECT

The identifiers of all the instances representing a word (or a group of words) created for an application are stored in the object **activentry**. This object is used in the next step to generate the application lexicon.

The third step

In the third step of the process, the grammar is generated. Grammar rules are obtained from the linguistic objects marked as active rules (those having its identified stored in the object **activerules**). The lexical entries are obtained from the instances marked as active entries (those having its identified stored in the object **activentries**). Once the grammar rules and lexical entries are obtained, a compiler translates them to their final form, checking for errors such as the existence of non-accessing categories or repeated information.

Following the example, two grammar rules and six lexical entries are obtained for expressing the creation of instances for the concept **ARCHITECT** in Spanish. They are described in the next chapter, where the formalism of the grammar and lexicon are detailed.

CHAPTER 7

THE NATURAL LANGUAGE INTERFACE

This chapter describes the architecture and functionality of the interface designed to control the NL communication between users and application. This interface has been designed as an independent module to be integrated into the application and the CO representing it. More than one possible architecture to integrate the interface into the application is allowed. The linguistic resources used by the interface are obtained by adapting the CO, the LO and the control rules to the application, as described in previous chapters.

This chapter has been organized in six sections. The first section gives a general description of the NLIs used by GISE. The components of this interface are described in the following sections.

7.1 AN OVERVIEW OF THE NLI

The interface designed for controlling the NL communication between the user and the application consists of two modules: the NLI and the communication manager (CM).

The NLI guides the user to introduce correct NL sentences, and once those sentences have been analyzed, it passes the resulting interpretation to the CM. The NLI module consists of all the components involved in the NL communication between the user and the

application. These components are the grammar and lexicon, the parser, the menu system and the dialogue component (DC). Figure 7.1 shows the components of the NLI, described in the following Sections 7.2-7.5.

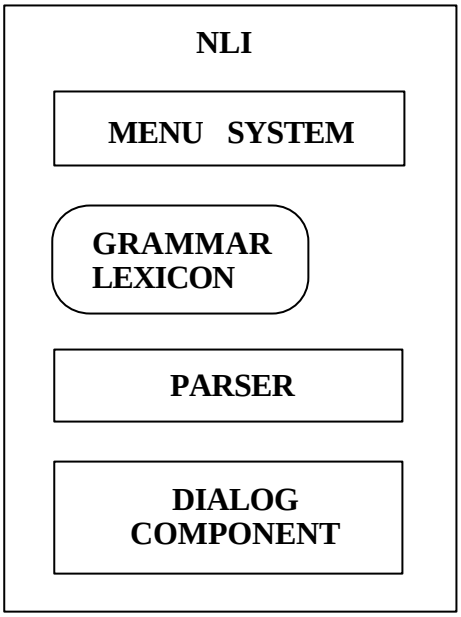

Figure 7.1: The NLI module

Figure 7.2 shows the flow of information through the different components in the interface. The menu system displays all possible NL options on screen. Once the user has select an option, the menu system passes it to the parser. Subsequently, the parser analyzes it and passes the set of next possible options to the menu system. Once a whole sentence has been recognized and interpreted by the parser, it is passed to the DC. If necessary, the DC completes the resulting interpretation and passes it to the CM. The CM controls the information exchange between the NLI and the application. The information arriving at the CM consists of one or more operations. The CM is in charge of executing these over the CO. When the application needs particular information from the user, it first consults the CO. If the necessary information is not found in the CO, it is requested directly from the user. The CM also allows for other forms of communication than NL, as described in Section 7.6.

All the components in the NLIs have been implemented in standard Prolog. The menu system, uses PC Arity Prolog predicates to display the menus containing the NL options the user can choose.

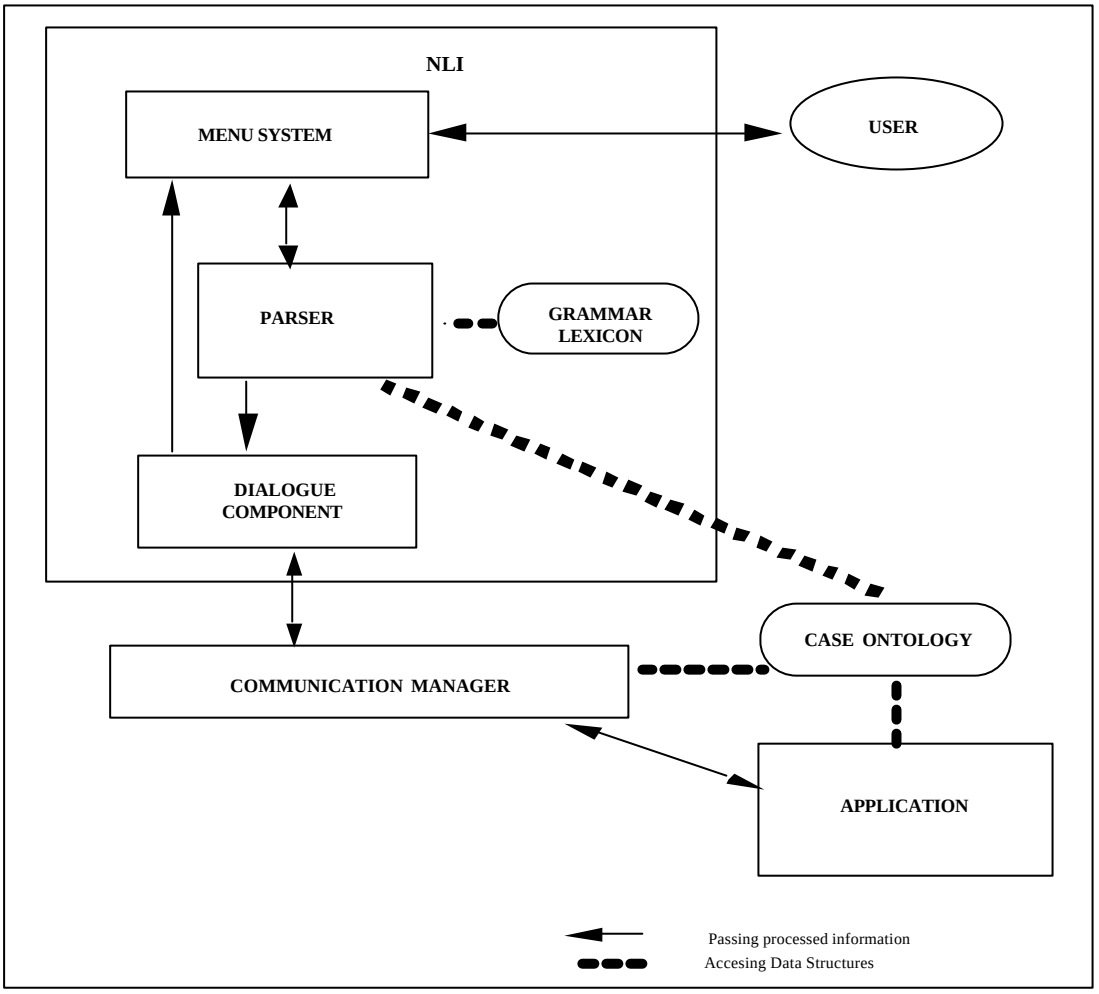

Figure 7.2: The processing of a user intervention

7.2 THE GRAMMAR AND THE LEXICON

As detailed in previous chapters, the process of generating the application-restricted grammar consists of adapting the LO to the communication tasks required for an application. When obtaining the appropriate linguistic structures for an application from its conceptual representation, instances of the linguistic classes in the LO domain level are generated. These instances are represented as definite-clause grammar (DCG) rules and lexical entries.

The DCG formalism has been chosen because:

- It is more expressive than conventional CFG.
- In a limited domain, DCGs can be efficiently parsed (using Prolog).
- The grammar being automatically created by the system, it is not necessary to use a more friendly formalism (such as Patr-II).

7.2.1 The resulting grammar

The grammar and lexicon are obtained from the LO instances created in the process of relating the LO to the tasks of communication necessary for an application. The LO instances described by more than one component correspond to grammar rules. In these LO objects, all allowed sequences of constituents appear in the facet **pattern**. One grammar rule is obtained from each presentation allowed for the constituents. The left-hand part of the grammar rule corresponds to the category associated with the linguistic object, represented in the facet **category**. The right-hand part of the rule is the sequence of categories representing the constituents of the instance. Each of these categories corresponds to an object in the LO.

For example, Figure 7.3 shows the DCG rule representing a possible expression in Spanish of the operation to create an instance without giving its name. It has been obtained from the LO domain level class **EXISTENTIAL_CLAUSE_CREATING_INSTANCE WITH_NO_NAME**, described in Figure 4.6.

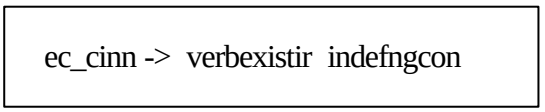

Figure 7.3: A grammar rule for expressing the operation to create an instance without giving its name

The left-hand part of the rule is the category of the linguistic class, **ec_cinn**. The right-hand part of the rule consists of two categories representing the two constituents of the linguistic class: the category **indefngcon**, representing the **subject** and the category **verbexistir** representing the **verb**.

An expression of the operation to create an instance giving its name is represented in the LO class **ATTRIBUTIVE_CLAUSE_CREATING_INSTANCE_WITH_NAME**, described in Figure 5.7. The grammar rule representing this class is shown in Figure 7.4.

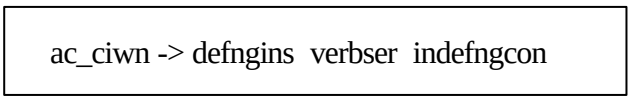

Figure 7.4: A grammar rule for expressing the operation to create an instance giving its name

Additional information describing the linguistic class is also represented in the grammar rules. This information consists of the syntactic and semantic features augmenting the categories, the semantic interpretation and the preconditions associated with each rule. The information incorporated into the rules is described below.

7.2.2 The semantic interpretation

The semantic interpretation associated with the linguistic resources generated for an application is based on lambda calculus. Semantic information is associated with each rule to indicate the order of interpretation of its constituents. This information consists of a list of the numbers representing the constituents. This semantic list is defined recursively as a list of two elements. Each element in the list can be either a number or a list of two elements, that in turn can be either a number or a list of two elements. In the particular case of the rule having only one constituent, the semantic list associated with the rule is empty. The semantic interpretation associated with each lexical entry consists of a lambda function or a lambda value.

At run-time, once the parser has recognized all constituents of the rule, it interprets them according to the semantic list. The result of evaluating the first element of the list is applied to the result of evaluating the second element in the list. If the element is a number, the result of its evaluation is the semantic interpretation associated with the constituent represented by the number. If the element is a list, then the result is obtained by applying the result of evaluating the first element to the result of evaluating the second.

Additionally, if the grammar rule represents an operation, the identifier of the operation is incorporated into the rule description. This information improves the processing of user interventions. The result of the semantic interpretation process is a list of words representing CO operations and their parameters (which correspond to concepts, attributes and values).

Incorporating the semantic interpretation into the grammar rules shown in Figure 7.3 and Figure 7.4, expressing the creation of conceptual instances will result in the two grammar rules shown in Figure 7.5. The semantic information associated with the first rule is the list **(2 1)**. This indicates that the semantic interpretation of the second constituent recognized by the rule (that associated with the category **verbexistir**) has to be applied to the semantic interpretation of the first one (that associated with the category **indefngcon**). The identifier

of the operation expressed by the rule, **CREATE_INSTANCE_WITH_NO_NAME_O** (abbreviated **crinno**) is also incorporated into the rule description.

```
ec_cinn -> verbexistir indefngcon   (1  2)   crinno

ac_ciwn -> defngins  verbser  indefngcon  ((2 3) 1)  criwno
```

Figure 7.5: Two grammar rules for creating an instance together with their semantic interpretation

The semantic information associated with the second rule is list **((2 3) 1)**. The first element in the list is a sublist indicating that the semantic interpretation of the second constituent recognized by the rule (that associated with the category **verbser**) has to be applied to the semantic interpretation of the third one (associated with the category **indefngcon**). The resulting lambda function has to be applied to the semantic interpretation of the first constituent recognized by the rule (that associated with the category **defngins**). The semantic information also includes the identifier of the operation expressed, **CREATE_INSTANCE_WITH_ NAME_O** (abbreviated **criwno**).

7.2.3 The syntactic and semantic features associated with the categories

If the communication tasks in an application were restricted to a limited number of operations over few entities representing the domain, then the process of generating the grammar and lexicon required would be much simpler. In that case, each concept, attribute and value in the CO could be represented by a different category. The grammatical categories could be directly related to each object in the CO. The existence of different grammar rules representing the same operation performed over different concepts and attributes is not a problem when the number of concepts and operations to represent is not high. This strategy simplifies the process of generating the grammar and lexicon, and the processing of user interventions.

However, when the complexity of the communication tasks and concepts increases and the language required to express the meanings is larger, the strategy described above is not appropriate. In complex applications, general syntactic information must be represented in the grammar to cover the different linguistic phenomena appearing in the communication tasks. Syntactic and semantic features are incorporated into the categories to improve efficiency in the processing of user interventions. These features indicate the concordance

between the syntactic and semantic characteristics of the constituents represented in the rule.

The syntactic features incorporated into the categories of the grammars generated by GISE are limited to giving information about the gender, number and tense of the word (or words) represented by the categories. The specific conceptual information representing the identifiers of the CO concepts and attributes is incorporated into the semantic features associated with the categories.

The features are obtained from the description of the LO object. The attributes **synres** and **semres** in the LO objects describe the agreement between the features associated with the categories representing the constituents.

Augmenting the categories in the rule with features representing existential clauses creating conceptual instances, shown in Figure 7.5, results in the rule shown in Figure 7.6. The category in the left-hand part of the rule, **ec_cinn** is augmented with the semantic feature **con**, representing the concept. The value of this feature must be the same as the value of the feature **con** associated with the category **indefngcon**. The number agreement between the subject (category **indefngcon**) and the verb (category **verbexistir**) is controlled by the syntactic feature **num** associated with both categories. The category representing the verb **verbexistir** is also augmented with the syntactic feature **tense**. The category **indefngcon** is also augmented with the syntactic feature **gen**.

```
ec_cinn(sem(con(C))) ->  verbexistir(syn(num(N),tense(T)))
                         indefngcon(syn(gen(G),num(N)),sem(con(C)))    (1 2)    crinno
```

Figure 7.6: Representing the syntactic and semantic features in the grammar rule for creating an instance without name

Figure 7.7 shows the second grammar rule in Figure 7.5 when augmenting its categories with their corresponding syntactic and semantic features.

The incorporation of this syntactic and semantic information into the grammar categories restricts the number of available options at each state of the analysis, as well as simplifying the interpretation of the constituents recognized by a rule.

```
ac_ciwn(sem(con(C))) ->  defngins(sem(con(C)))   verbser(syn(num(N),tense(T)))
                         indefngcon(syn(gen(G),num(N)),sem(con(C)))    ((2 3) 1)     criwno
```

Figure 7.7: Representing the syntactic and semantic features in the grammar rule for creating an instance giving its name

7.2.4 The resulting lexicon

The lexical entries are obtained from the LO instances corresponding to words or sequences of words. These instances represent the parameters of the application operations as well as the general words belonging to closed syntactic classes, such as auxiliary verbs and articles. The general words are represented in the LO and are reused for all applications. The semantic interpretation associated with these general instances consists of lambda functions. The semantic interpretation associated with the instances expressing operation parameters generated for each application consists of either a lambda function or value. The fields describing the lexical entries are obtained from the instance facets. The lexical entries consist of three fields: category, semantic interpretation and string (or linguistic realization)

Figure 7.8 shows the lexical entries representing the two forms of the verb *ser* (*be*) in the present tense. These two lexical entries belong to the LO. As shown in this figure, the two entries in Figure 7.8 are associated with the category **verbser**, augmented with the syntactic features **num**, representing the number and **tense**, representing the tense of the verbal form.

category	semantic interpretation	string
verbser(syn(num(s),tense(p)))	(((l,X),(l,Y)),(X,Y))	< es >
verbser(syn(num(p),tense(p)))	(((l,X),(l,Y)),(X,Y))	< son >

Figure 7.8: Lexical categories representing the verb ser

The semantic interpretation of the lexical entries in Figure 7.8 is a lambda function represented by the list: **((l, X), (l, Y), (X,Y))**. The first sublist indicates that the function has two lambda arguments, represented by the variables **X** and **Y**. The second sublist establishes that the function returns a list containing the value of the two variables. This semantic interpretation indicates that the verb *ser* (*be*) acts as a link between two semantic values.

Examples of the lexical entries generated for a specific application are shown in Figure 7.9.

These two lexical entries are recognized by the rules described above for creating conceptual instances. The categories associated with these entries exemplify how semantic and syntactic information is encoded in the categories representing CO objects. For example, the syntactic features associated with the category **indefngcon** are **syn(gen(G),num(N))**, indicating the lexical entry gender (**m** for masculine and **f** for feminine) and number (**s** for singular and **p** for plural). The semantic feature **(sem(con(C)))** represents the concept identifier[1] (i.e. **architect**, **owner**).

category	semantic	string
indefngcon(syn(gen(m),num(p)),sem(con(architect)))	architect	<unos_arquitectos>
indefngcon(syn(gen(m),num(s)),sem(con(owner)))	owner	<un_propietario>

Figure 7.9: Examples of lexical entries representing concepts of the application SIREDOJ

7.2.5 Providing the grammar and lexicon of dynamic mechanisms

Dynamic mechanisms were incorporated into the grammar and lexicon generated in order to use the contextual information available at run-time to reduce the grammar rules and lexical entries that need to be considered. Dynamic mechanisms consist of preconditions attached to the grammar rules and the dynamic entries whose value is set during the communication process. These dynamic mechanisms are independent of the grammar formalism, although they are especially well suited to unification based grammars.

The dynamic entries

Dynamic entries are incorporated into the application-restricted grammar to improve the efficiency and friendliness of the NL communication. Their superficial representation, as well as the semantic interpretation associated with them, is set at run-time, while the string and interpretation in the remaining (non-dynamic) entries is set during the generation phase. The use of dynamic entries reduces the number of lexical entries to be considered at run-time and allows the user to introduce new values.

[1] Notice the semantic feature **con** represents the identifier of the concept (**architect**), not its linguistic realization. In the examples appearing in this chapter, the concepts and attributes identifiers are in English while their corresponding linguistic realizations are in Spanish.

Dynamic entries consist of two fields: the linguistic category and the dynamic function. The categories of these entries are augmented with semantic features. Syntactic features are not associated with the categories of dynamic entries, because of their superficial form, and thus their syntactic features are unknown in the generation phase.

The dynamic function associated with these entries is a function requesting the user to introduce specific values during the communication process. The value introduced by the user will be set as the semantic interpretation associated with the category. Dynamic functions are written as Prolog predicates.

There are three different types of dynamic entries:

- Those entries representing instances of concepts
- Those entries representing a proper noun or a number that will be requested to the user at run-time
- Those entries associated with a menu (or window) that will be displayed on the screen at run-time

Dynamic entries representing instances of concepts are associated with the operation that obtains all identifiers for existing instances of a specific CO concept in the very moment of the communication, and displays them. Examples of this type of dynamic entry are represented in Figure 7.10. The function associated with these entries is the Prolog predicate **instance** having the identifier of a concept as parameter. When the user selects these entries, all instances of the concept in the case level of the CO are displayed on screen.

category	dynamic fuction
pngi(sem(con(person)))	instance(person)
pngi(sem(con(building)))	instance(building)
defngvalwho_subject(sem(con(buildingcontract_parts),attr(subject1)))	instance(person)

Figure 7.10: Examples of lexical entries representing instances of concepts

The two first entries in the figure represent instances of concepts. The third entry represents the value of the attribute **subject1** in an instance of the concept **ASSIGNMENT_PARTS** (representing the parts involved in a contract). The attribute **subject1** represents the subject that assigns. Its value must be an instance of the concept **PERSON**.

The function associated with the second type of dynamic entries is a function requesting

the user to introduce an open value, that is, a proper noun or a number. Three examples of these entries are represented in Figure 7.11. The first lexical entry in the figure is associated with the category **defngins**. It represents the name given by the user when creating a conceptual instance. The function asks the user to introduce the name of the new instance created at run-time.

category	dynamic function
defngins(sem(con(person)))	name
defngvalof_name(sem(con(train_station), attr(city)))	name
nadjvalof_quantity(sem(con(person), attr(age)))	number(years)

Figure 7.11: Examples of lexical entries representing names and quantities

The second dynamic entry in Figure 7.11 is associated with the category **defngvalof_name**, representing the name that corresponds to the value of an attribute belonging to the class **OF_NAME**. This second entry in the figure corresponds to the proper name filling the attribute **city** of the instances of the concept **TRAIN_STATION**, described in Figure 4.2. It is associated with the function **name**.

The third dynamic entry represents the value of the attribute **age** of instances for the concept **PERSON**, described in Figure 4.3. Its category is **nadjvalof_quantity**, corresponding to the numeral adjectives representing the value of adjectives in the class **OF_QUANTITY**. It is associated with the function **number**, requesting the user to introduce a quantity at run-time. This function has an argument representing the unit of measurement.

Finally, the third type of dynamic entries are those associated with functions responsible for displaying a specific menu with a set of fixed values at run-time. The user has to choose only one or more of the values in the menu. Two examples of these entries are represented in Figure 7.12. The two lexical entries in the figure correspond to two different instantiations of the same category, **defngvalof_cause**. This category corresponds to the value of an attribute in the class **OF_CAUSE**. The first entry represents the value of the attribute **reasonotbuilt** for the concept **BUILDING_REQUIREMENT**, described in Figure 4.7. When the user selects this option, a menu with all possible reasons why the requirement to build has not been fulfilled are displayed on screen.

category	dynamic function
defngvalof_cause(sem(con(building_requirement), attr(reasonotbuilt)))	menu(reasonotbuilt)
defngvalof_cause(sem(con(payment_requirement), attr(reasonotpaid)))	menu(reasonotpaid)

Figure 7.12: Examples of lexical entries representing menus

The second lexical entry represents the value of the attribute **reasonotpaid** for the concept **PAYMENT_REQUIREMENT**. The range of this attribute is a set of predefined options describing all possible reasons why a party to a building contract has refused to fulfill the duty to pay.
A new example of grammar rule generated to support the expression of a CO operation is given in next section.

7.2.6 A new example: The grammar rules generated for the operation filling the attributes in the class IS

As described before, the process of obtaining the appropriate grammar and lexicon for an application consists of obtaining the linguistic resources necessary to express all operations allowed for an application. This process mainly consists of mapping the object representing an operation onto the corresponding linguistic objects in the LO. In this process, both the class of the operation and the class of the conceptual attributes involved in the operation are considered.

Two grammar rules obtained from the operation filling one attribute belonging to the class **IS** are described immediately below. These two rules express the operation of filling a concept attribute belonging to the class **IS** and having the closed set **yes/no** as possible values. These two rules represent the two general forms to express this operation in Spanish:

<concept name> <verb be> <attribute name>

<concept name> <no> <verb be> <attribute name>

The resulting grammar rules are shown in Figure 7.13. There are also several other possible combinations of these constituents not represented in the grammar rules shown in this figure.

```
ac_fais  -->  defngcon  verbestar  dadjattris   ((2 1) 3)   fais

ac_fais  -->  defngcon  no  verbestar  dadjattris  ((3 1) (2 4))   fais
```

*Figure 7.13: The grammar rules expressing the operation filling an attribute belonging to the class **IS***

The left-hand part of the two rules is represented by the category **ac_fais**. This category corresponds to the category of the attributive clause expressing the filling of a conceptual attribute belonging to the class **IS**. This clause is represented in the LO application level as a subclass of the **ATTRIBUTIVE_CLAUSE** class, shown in Figure 5.5.

The right-hand of the two rules is represented by the categories associated with the constituents of the clause. These categories are **defngcon**, representing the definite noun expressing the concept, **verbestar**, **no** and **dadjattris**, representing the connotative adjective expressing the attribute.

The categories in the rules are augmented with syntactic and semantic features. The resulting grammar rules are shown in Figure 7.14. The category **defngcon** is augmented with the syntactic features **gen** and **num**. It is also augmented with the semantic feature **con**. Its semantic interpretation is a lambda value representing the concept identifier. The string associated with this category corresponds to the superficial representation of the concept.

```
ac_fais(sem(con(C))) -->
   defngcon(syn(gen(G),num(N)),sem(con(C)))
   verbestar(syn(num(N),tense(T)))
   dadjattris(syn(gen(G),num(N)),sem(con(C),attr(A)))     ((2 1) 3)    fais


ac_fais(sem(con(C))) -->
   defngcon(syn(gen(G),num(N)),sem(con(C)))
   no verbestar(syn(num(N),tense(T)))
   dadjattris(syn(gen(G),num(N)),sem(con(C),attr(A)))    ((3 1) (2 4))   fais
```

Figure 7.14: Two grammar rules for filling attributes in the class IS, augmented with syntactic and semantic features

The category **dadjattris** is augmented with the same syntactic features: **gen** and **num**. It is, additionally, augmented with the semantic features **con**, representing the concept and **attr**, representing the attribute. Its semantic interpretation is a lambda value representing the identifier of the attribute. The attribute realization is represented in the field **string**.

category	interpretation	string
defngcon(syn(gen(f),num(s)), sem(con(delivery_requirement)))	(delivery_requirement)	<la_obligacion_de_entrega>
defngcon(syn(gen(f),num(s)), sem(con(payment_requirement)))	(payment_requirement)	<la_obligacion_de_pago>

Figure 7.15: Lexical entries representing concepts of the application SIREDOJ

Examples of Spanish lexical entries recognized by these two grammar rules are shown in the Figures 7.15-7.18. The lexical entries shown in Figure 7.15 and Figure 7.16 are generated for the application SIREDOJ. These entries are associated with the categories **defngcon** and **dadjattris**, appearing in the grammar rules in Figure 7.14.

category	inter.	string
dadjattris(syn(gen(f),num(s)),sem(con(delivery_requirement),attr(dfulfilled)))	(dfulfilled)	<cumplida>
dadjattris(syn(gen(f),num(s)),sem(con(payment_requirement),attr(pfulfilled)))	(pfulfilled)	<cumplida>

Figure 7.16: Lexical entries representing attributes of the application SIREDOJ

Examples of lexical entries associated with the same categories for the railway communication system are shown in Figure 7.17 and Figure 7.18. These lexical entries are used to express the filling of the attribute **full** of the concepts **INTERCITY1** and **EUROMED1**, belonging to the conceptual class **TRAIN**. Examples of clauses expressing this operation are *El Intercity está completo* (*The Intercity is full*) and *El Euromed no está completo* (*The Euromed is full*).

category	interpretation	string
defngcon(syn(gen(f),num(s)), sem(con(intercity1)))	(intercity1)	<el_intercity>
defngcon(syn(gen(f),num(s)), sem(con(euromed1)))	(euromed1)	<el_euromed>

Figure 7.17: Lexical entries representing concepts of the railway communication application

The incorporation of semantic information into the categories restricts the number of entries the analyzer must consider at run-time. For example, if the grammar rules described above were the only active, then if the user introduces the string *el_intercity*, associated with the category **defngcon(syn(gen(m),num(s)),sem(con(intercity1)))**, and, following

this, introduces the string *esta*, associated with the category **verbestar(syn(num(s),tense(p)))**, only the entries whose category matches the next category in the rule, **dadjattris(syn(gen(m),num(s)),sem(con(intercity1),attr(A))),** will be accepted as a result. The accepted entries will be those corresponding to the attributes describing the concept **intercity1**, belonging to the class **IS** and having the associated preconditions satisfied. If the only possible lexical entries were those in Figures 7.17 and Figure 7.18, then the only accepted string will be *completo.*

category	interpretation	string
dadjattris(syn(gen(m),num(s)), sem(con(intercity1), attr(full)))	(full)	<completo>
dadjattris(syn(gen(m),num(s)), sem(con(euromed1), attr(full)))	(full)	<completo>

Figure 7.18: Lexical entries representing attributes of the train consulting application

The semantic list associated with the first rule in Figure 7.14 indicates that the lambda function associated with the constituent recognized by the category **verbestar** must first be applied to the lambda value associated with that recognized by the category **defngcon**. Next, the resulting function must be applied to the lambda value associated with the constituent recognized by the category **dadjattris**.

The identifier of the operation expressed by the rules, **fais**, is also included in the rules description. It is incorporated in the list resulting of the lambda calculus described above.

For example, considering the lexical entries shown in the Figure 7.15 and Figure 7.16, if the clause introduced is *La obligación de entrega está cumplida* the resulting interpretation will be: (**fais, delivery_requirement**, **dfulfilled**). If the clause introduced is *La obligación de pago está cumplida* the result will be: (**fais, payment_requirement**, **pfulfilled**). The different phases in the interpretation process are detailed in next section.

7.2.7 Preconditions attached to the grammar rules

Preconditions are incorporated into the grammar rules to dynamically adapt the linguistic resources to the application requirements. Thus, at each stage of the communication, only the entries that the application can process are accepted by the NLI.

As described in Chapter 4 in Section 4.6, there are preconditions associated with the operations and preconditions associated with concepts. For example, the performance of all simple operations filling the attributes of the conceptual instances is governed by the

existence of the instances. The preconditions associated with concepts are case conditions (that must be evaluated at run-time) governing the creation of instances of concepts and the filling of their attributes. They condition the creation of instances of a specific concept to the previous existence of instances of other concepts. They may also condition the value of an instance attribute to the values of other attributes. Examples of these conditions are those associated with the concept **BUILDING_REQUIREMENT**, shown in Figure 4.7.

When adapting the general structures to express the specific operations required for an application, the preconditions associated with the operations and concepts are represented as facets of the corresponding LO instances. Such preconditions are attached to the grammar rules representing these LO instances.

Figure 7.19 shows the preconditions associated with the rule representing the existential clauses that the creation of conceptual instances without giving their name.

```
pcg([pcon(C)])   ec_cinn(sem(con(C))) ->
                      verbexistir(syn(num(N),tense(T)))
                      indefngcon(syn(gen(G),num(N)),sem(con(C)))  (1 2)  crinno
```

Figure 7.19: A rule for creating a conceptual instance without giving its name together with its associated preconditions

As shown in the figure, the predicate **pcg ([pcon(C)])** is attached in front of the left-hand part of the rule. It is represented as a Prolog predicate. The argument of the predicate **pcg** is a list containing all preconditions. In this example, there is only one element in the list, the predicate **pcon(C)**, representing all preconditions associated with concept **C**. This precondition indicates that a conceptual instance can be created if all preconditions associated with the concept are satisfied.

Only the lexical entries associated with the concepts and attributes satisfying the grammar rule conditions at run-time will be accepted by the interface. The user will know the correct options at each stage of the communication because a system of menus has been integrated into the interface. All possible entries that the user can choose to build the sentence are displayed in menus.

The preconditions attached to a grammar rule representing the operation to fill attributes in the class **IS** are shown in Figure 7.20. There are two preconditions attached to the left-hand part of the grammar rules. These preconditions are obtained from those governing the operation **FILL_ATTRIBUTE_IS_O**. The case preconditions associated with this operation are inherited from the **FILL_ ATTRIBUTE_O**, described in Figure 4.5. These

preconditions are: **((instance _ins _con) (pcon _con _attr))**

These two preconditions establish that the operation filling an attribute of a conceptual instance can only be performed if the instance exists, and if it satisfies the case precond tions included in the concept description governing the filling of the attribute. These preconditions are attached to the grammar rules recognizing the expression of the operation as a list of two Prolog predicates.

The resulting Prolog list is:

[instance(I,C), pcon(C, A)][2]

The predicate arguments are variables whose value is set at run-time using the Prolog unification mechanism. These two Prolog predicates ensure that only the correct entries will be accepted at run-time. The accepted entries will be those representing the existing conceptual instances and those representing the attributes of these instances satisfying the preconditions defined in the concept.

The first precondition attached to the grammar rules shown in Figure 7.20 is represented by the Prolog predicate **instance(X, C)**. This predicate will be satisfied if and only if in the CO there is at least one instance of the concept represented in the variable **C**. This associated precondition prevents the description of a conceptual instance before its creation (i.e., only if there is an instance of the concept **DELIVERY_REQUIREMENT**, can its attribute **dfulfilled** be filled).

The second precondition represents the case preconditions included in the conceptual definition governing the filling of the attribute. They are represented by the predicate **pcon(X,A)**, which will be satisfied if the conceptual case preconditions governing the filling of the attribute represented in the variable **A** are satisfied for the instance represented in the variable **X**. If no preconditions governing the filling of the attribute are associated with the concept, then this predicate is always satisfied. For example, as there is no preconditions associated with the concept **INTERCITY**, this second predicate has no effect when building the clause: *el intercity está completo* (*the intercity is full*).

As shown in Figure 7.20, the list containing the preconditions associated with the rules is represented as a term of the Prolog predicate **pcg**. This predicate is attached in front of the left-hand part of the rule.

[2] The name of each variable in the precondition is represented as an underscore followed by the name of the argument. In this example, the name of the variable associated with the Prolog predicates representing the preconditions is indicated by a capital letter

```
pcg ( [instance (X, C), pcon ( X, A)] )
  ac_fais(sem(con(C))) -->
    defngcon(syn(gen(G),num(N)),sem(con(C)))
    verbestar(syn(num(N),tense(T)))
    cadjattris(syn(gen(G),num(N)),sem(con(C),attr(A)))     ((2 1) 3)   fais
```

Figure 7.20: A grammar rule for filling the attribute IS and its preconditions

New preconditions for adapting the generated grammar to other considerations not depending on the CO, such as screen size, can also be attached to the grammar rules during the generation phase. Examples of these preconditions are those conditions limiting the number of active instances at a given time.

7.3 THE PARSER

The parser used in the NLIs is a left-corner unification based chart parse. The implementation follows the modified left-corner algorithm described in [Ross82]. The central points of this parser are described in Section 7.3.1. The modifications of this basic algorithm are detailed in Section 7.3.2

7.3.1 The left-corner algorithm

The NBT (Non-selective Bottom to Top) algorithm applied to a CFG proposed by Griffiths and Petrick in 1965 is known as the **left-corner** algorithm. Since its efficiency in NL processing was demonstrated by Slocum in [Slocum81], several versions of this algorithm have been implemented for NLIs. These versions include that used in the NLMENU ([Thomson86]), a head-corner parser developed for the OVIS system, a Dutch spoken dialogue system [VanNoord97] and a probabilistic version described in [Manning97].

The main advantage of this algorithm is it satisfies the condition on the prefix correctness. This conditions guaranties that from a correct prefix there is always a correct choice to continue. This condition is especially important when the sentences to parse are introduced incrementally, as is the case in the interfaces incorporating a system-menu that guides the

user to introduce one by one the word (or sequence of words) in a sentence.

Ross proposed an improved modification to transform the recognition algorithm proposed by Griffiths and Petrick into a parsing algorithm. This modified algorithm is given below.

The algorithm uses three push-down stacks called **alpha, beta** and **gamma**. The **alpha** stack contains the next symbols (categories) to be recognized. The **beta** stack contains the symbols of the rules that have been selected in previous analysis steps and have subsequently to be recognized. The **gamma** stack contains the symbols that have already been recognized.

Initially, the terminals (or symbols) to be parsed followed by an END are pushed on stack **alpha**. The non-terminal that is to be the root (or bottom symbol) followed by END is put on the **beta** stack. The gamma stack only contains END. The ultimate goal is constructing a tree that has as the root node the non-terminal symbol on the bottom of the **beta** stack as a single element.

Three rules of the follow the form can then be applied:

[A,B,C] -> [D,E,F] if Conditions,

A, B, C, D, E and F being arbitrary terminal and non-terminal symbols (or categories). This general form must be interpreted as: if A is on the top of **alpha**, B is on top of **beta**, C is on top of **gamma** and Conditions are satisfied, then replace A by D, B by E and C by F. The three rules are:

Rule 1

[V1,X,Y] -> [? ,V_2...V_ntX, V_1AY] if A ->V_1V_2...V_n is a production of the grammar, X is in the set of nonterminal symbols and Y is anything

This rule states that if there is a grammar rule having as left-corner the symbol to be recognized, then the symbol is popped from **alpha,** the rest of the symbols in the right-hand part of the rule are pushed into **beta** and the symbol in the left-hand part is pushed into **gamma**. The **t** is pushed into stack **beta** to mark the end of the grammar rule symbols. The symbol recognized is also stored in **gamma**, attached beneath the symbol in the left-hand part or the rule (resulting in V_1AY in the top of the **gamma** stack).

To increase the efficiency of the algorithm, a condition eliminating bad parse paths before trying them is added to this rule. This condition states that the symbol X may reach down to the symbol A. X may reach A if A is a left-corner and if there is at least one rule having as right-hand part X and as a left-corner either A, or a symbol that may reach down to the symbol A. A *reachability* matrix may be used to facilitate the evaluation of this task.

Rule 2

[X,t,A] -> [AX,? ,?] if A is in the set of nonterminals

The **t** in top of the **beta** stack indicates that all the symbols of the last grammar rule selected have been recognized. In that case, the sub-tree just built is popped from **gamma** and pushed into **alpha**.

In this situation, the semantic analysis of the symbols recognized can proceed. To perform the semantic interpretation, however, the symbols representing the words introduced must be associated with their semantic interpretation. Once the semantic interpretation is performed, the result is then attached to the root of the syntactic sub-tree just built and pushed into **gamma**.

Rule 3

[B,B,Y]->[? ,? ,Y] if B is in the set of nonterminals or terminals

This rule states that if the symbol in the top of **alpha** is the same as that on top of **beta,** the symbol has been recognized. In that case this symbol is popped from **alpha** and **beta** and attached as the right daughter of the top symbol on **gamma** (resulting BY on top of **gamma**).

The repeated application of the three rules described above always lead to the final state. The final state is reached when either none of three rules can be applied or a parse is found. When END is on top of each stack, the symbol has been recognized. The resulting parse is one possible parse for the sentence. If none of the three rules can be applied and END is not on top of each stack, a bad path has been followed.

In the particular case the algorithm is used to parse a word at a time, a final state is also reached when the top of **alfa** is END, indicating that the word has already been recognized. In this situation, the stacks **beta** and **gamma** resulting from the parse, along with the next word introduced, are passed to the analyzer.

There are two situations in which the performance of the algorithm is not deterministic. The first situation is that in which both Rule 1 and Rule 3 can be applied. That is, the top of alpha is the same symbol on the top of **beta** and there is also a grammar rule having this symbol as left-corner. The second non-deterministic situation is when searching for a new grammar rule to apply Rule 1, and there is more than one possible choice. That is, there is more than one grammar rule having as left-corner the symbol on top of **alpha**.

Each of the parse paths resulting from an application of a different rule and from the selection of a different rule could result in a valid parse. Thus, all these paths must be followed to completion. The strategy followed in these non-deterministic situations is to continue parsing in a depth-oriented manner. That is, apply one rule (or select one rule),

get a new state, and then apply one of the applicable rules to that new state. This continues until the final state is reached. To assure that all alternatives are pursued, once the final state is reached, backtracking to the last choice-point must be done to pick another applicable rule (or grammar rule).

The parser could easily be adapted to analyze the input in a breath-oriented manner. However, when the algorithm is used to parse a word at a time, as with GISE, the NL-MENU system and in other NLIs, a depth-oriented manner is more appropriate.

The Ross enhance version of the algorithm incorporates the ability to predict the set of all possible nth words of a sentence, given the first n-1 and this is the main reason of choosing this algorithm. The possible words to continue the sentence are those words reachable from the top of the beta stack, which represents the next category to be recognized in the rule selected. These words can easily be obtained using the *reachability* matrix mentioned above. Displaying all the next NL options on screen is very useful for guiding the user to build the NL sentences acceptable to the system.

The Ross version also includes a modification for dealing with grammar rules that employ conventions of abbreviations that are often used when writing grammars. This modification, however, has not been incorporated into the version implemented for GISE. This new ability does not significantly increase efficiency when using the grammars generated by GISE, given the preconditions attached to the grammar rules.

The modifications introduced when adapting this algorithm to the GISE NLI are described below.

7.3.2 The GISE parser

The parser used in the NLIs in GISE is based on the left-corner algorithm described above. Few modifications have been required to adapt it to the system functionality. The GISE parser has been adapted to DCG where categories are augmented with syntactic and semantic features. The original algorithm was designed for CFG. GISE parser can deal with the more complex phenomena that arise when using DCG, such as the feature agreement between two or more categories. The evaluation of the agreement of the features associated with the categories has been implemented using the Prolog unification mechanism.

An also important aspect of the modified algorithm is that it can deal efficiently with grammar rules having preconditions attached. As described earlier, preconditions are incorporated into the grammar rules to restrict user entries to those the application can

process at each state of the communication. Thus, when selecting the next reachable rules from a given state of analysis, only the rules satisfying the preconditions must be considered. To perform this process efficiently, additionally to the tree push down stacks described above, a queue containing all grammar preconditions that are satisfied at each state is used. The grammar preconditions are on the case level of the CO. They are evaluated once a change to the ontology has been performed; that is, when either a sentence introduced by the user or an application statement has been executed.

The parser has also been adapted to deal with more than one possible lexical entry associated with the string introduced by the user. That produces a new situation in which the performance of the parser in not deterministic. If there is more than one entry associated with the word to recognize, then all alternatives are pursued.

The GISE parser, as well as the Ross modified version, can be used to parse a whole sentence and to parse a word at a time. When incrementally parsing, once a word has been recognized, the next possible words are obtained. No final marks are used in the GISE parser (such as END and t used in the original algorithm described above). When using the parsing a word at a time, the process ends each time the stack **alpha** is empty. When parsing a whole sentence at a time, the process ends when either the three stacks are empty or when no rule can be applied.

Semantic analysis

In GISE, the parser performs the syntactic and semantic analyses in parallel. Once the categories in a rule have been recognized, they are interpreted. To perform the semantic analysis, semantic information is associated with the grammar rules and the symbols (or categories) representing the words introduced by the user. This semantic information is also stored in the **alpha** and **beta** stacks described above.

The elements stored in **alpha** are named **item**. Each item consists of a list of three elements. The first element is the syntactic category to be recognized. The second element is the superficial representation associated with a lexical entry, or the last syntactic sub-tree built. The third element is semantic interpretation. This information corresponds to the interpretation associated with either the lexical entry, or with the sub-tree that has been built. The information in the grammar rules selected **(**both their left-hand part and their semantic interpretation) are stored in **gamma**. This stack also contains the lexical entries that have been recognized by the categories of the rules that are being processed.

As described earlier, the semantic analysis used is based on lambda calculus. Lambda calculus allows a simple and efficient interpretation process. Once all constituents in the

rule have been recognized, they are analyzed semantically following the order indicated in the semantic list associated with the rule. The semantic interpretation associated with each category consists of a lambda function or value. The semantic analysis consists of applying the lambda functions over the lambda values following the order indicated in the rule. The result of the semantic analysis consists of a list of all possible semantic interpretations for the input.

In the resulting NLI, each user intervention consists of a sentence expressing one or more operations over the CO. Each possible semantic interpretation of this sentence contains the information needed to execute those operations. For each operation expressed, the semantic interpretation consists of the name of the operation, the parameters expressed in the sentence and additional information to deduce those parameters not expressed. This additional information consists of the left-hand category of the grammar rule recognizing the sentence and the name of the concept over which the operation is applied.

The semantic interpretation built by the parser will subsequently be passed to the dialogue component. This component will complete the information in order to perform the corresponding operations.

7.3.3 Following an example

An example of the process of analysis is detailed below. The analysis of the sentence *existe un arquitecto* (*there isan architect*), expressing the creation of an instance of the concept **ARCHITECT** is described.

The grammar generated for creating an instance of a CO concept contains the two rules described in the previous section. As discussed earlier, preconditions and the name of the operation are included in the rule description. If the root category is **c**, two new rules have to be added to the grammar in order to reach the rules mentioned. The resulting grammar is shown in Figure 7.21.

```
pcg ([])  c  ->   ec_cinn(sem(con(C)))  ( ) ( )

pcg ([])  c  ->   ac_ciwn(sem(con(C))) ( ) ( )

pcg ([pcon(C)])   ec_cinn(sem(con(C))) ->
        verbexistir(syn(num(N),tense(T)))
        indefngcon(syn(gen(G),num(N)),sem(con(C)))      (1 2) (crinno)

pcg ([pcon(C)])  ac_ciwn(sem(con(C))) ->
      defngins(sem(con(C)))    verbser(syn(num(N),tense(T)))
      indefngcon(syn(gen(G),num(N)),sem(con(C)))     ((2 3) 1) (criwno)
```

Figure 7.21: A grammar for generating conceptual instances

The lexicon required to express the creation of instances of the concept **ARCHITECT** is shown in Figure 7.22.

category	semantic interpretation	string
verbser(syn(num(s), tense(p)))	(((l, X), (l, Y)), (X,Y))	<es>
verbser(syn(num(p), tense(p)))	(((l, X), (l, Y)), (X,Y))	<son>
verbexistir(syn(num(s), tense(p)))	(((l, X)), (X))	<existe>
verbexistir(syn(num(p), tense(p)))	(((l, X)), (X))	<existen>
indefngcon(syn(gen(m),num(s)), sem(con(architect)))	(architect)	<un_arquitecto>
defngins (sem(con(architect)))		name

*Figure 7.22: The lexicon necessary for creating instances of the concept **ARCHITECT***

In the initial state, the **alpha** and **gamma** stacks are empty. The **beta** stack contains the root category, in this case **c**. The list of satisfied preconditions will contain all the preconditions attached to the grammar rules that are satisfied in the initial state. In this grammar, the only precondition attached is **pcon(C)**. This precondition, as explained above, indicates that a conceptual instance can be created if all preconditions associated with the concept are satisfied. Most concepts, such as the concept **ARCHITECT**, do not have associated preconditions. For these concepts, the precondition **pcon(C)** is always satisfied. The precondition **pcon(architect)** will be the only element in the list of satisfied preconditions in the example described.

First, the parser will obtain the correct options to start. That is, all NL options that can be reach from the root category, **c**, stored in stack **beta**. In this example, the only possible options will be the word *existe* and the function **name** that ask for the name of the instance. The menu system will display these options on screen and will pass the value introduced

by the user to the parser. If the user chooses a word displayed in the menu, then the parser selects the first lexical entry associated with this word (it can be more than one) and pushes it into **alpha**. Each lexical entry contains three values: category, string and semantic interpretation. If the user is asked to introduce a value by a dynamic function, then a list containing the syntactic category, the symbol **ins** and the value introduced by the user, is passed to the parser.

If the user introduces the proper name *Juan*, the list

[indefng(sem(con(architect))),[Juan],ins]

is passed to the parser and pushed into **alpha**.

The three rules of the algorithm are then evaluated. Rule 1 is the only one that can be executed. The category of the item on top of **alpha** is **indefng(sem(con(architect)))**, the top of **beta** is **c** and there is one grammar rule reachable from category **c**, having **indefng(sem(con(architect)))** as left-corner category and its preconditions satisfied.

The only one grammar rule that can be selected in this example is:

```
pcg ([pcon(C)])  ac_ciwn(sem(con(C))) ->
      defngins(sem(con(C)))   verbser(syn(num(N),tense(T)))
      indefngcon(syn(gen(G),num(N)),sem(con(C)))      ((2 3) 1)  (criwno)
```

As a result of executing Rule 1, the three stacks change their values. The left-hand part of this rule as well as its semantic information and the lexical entry are pushed into **gamma**. The item recognized is popped from **alpha**. The rest of the categories in the right-hand part of the rule selected are pushed into **beta**.

In the example, the resulting stacks once Rule 1 is executed are:

```
stack alpha = ( )

stack beta =
   ((verbser(syn(num(N),tense(T))),
      indefngcon(syn(gen(G),num(N)),sem(con(architect)))) , c)

stack gamma =
   ((((defngins(sem(con(architect))),(Juan),ins)),
     ((2, 3), 1),ac_ciwn(sem(con(architect))),criwno))

precondition list = (pcon(architect))
```

An empty **alpha** stack indicates that a word has been parsed. Subsequently, the next acceptable words are obtained and displayed on screen by the menu system. In this case, the only possible word to continue the input sentence is *es*. Once the user selects **it**, the corresponding lexical entry is obtained. The entry for the word *es* is:

```
(verbser(syn(num(s),tense(p))), (((l,X),(l,Y)),(X,Y)) , es)
```

This entry is pushed into the **alpha** stack. The resulting stacks are:

```
stack alpha =
    ((verbser(syn(num(s),tense(p))),(((l,X),(l,Y)),(X,Y)),es))

stack beta =
   ((verbser(syn(num(N),tense(T))),
      indefngcon(syn(gen(G),num(N)),sem(con(architect)))) ,c)

stack gamma =
   ((((defngins(sem(con(architect))),(Juan),ins)),
     ((2, 3),1),ac_ciwn(sem(con(architect))),criwno))

precondition list = (pcon(architect))
```

The three rules are evaluated again. At this step, Rule 3 is the only one that can be executed. It can be executed because the category of the item on top of **alpha** and that on

top of **beta** is the same. This rule pops the item recognized from the **alpha** stack and pushes it into the **gamma** stack. It also pops the top of the **beta** stack, containing the last category recognized.

The resulting stacks are:

```
stack alpha = ( )

stack beta =
   ((indefngcon(syn(gen(G),num(s)),sem(con(architect)))),c)

stack gamma =
   (((verbser(syn(num(s), tense(p))),(((l,X),(l,Y)),(X,Y)),es)      ,
      (defngins(sem(con(architect))),(Juan),ins)),
    ((2, 3), 1),ac_ciwn(sem(con(architect))),criwno))

precondition list = (pcon(architect))
```

The **alpha** stack is empty again and thus, the next acceptable words are obtained and displayed on screen. The only possible option to continue the input sentence is *un_arquitecto.* Once the user selects it, the corresponding lexical entry is obtained and pushed into stack **alpha**. The entry for these two words is:

```
(indefngcon(syn(gen(m),num(s)),sem(con(architect))), (architect), un_arquitecto)
```

The resulting stacks are:

```
stack alpha =
   ((indefngcon(syn(gen(m),num(s)),sem(con(architect))),(architect),un_arquitecto)

stack beta =
   ((indefngcon(syn(gen(G),num(s)),sem(con(architect)))) ,c)

stack gamma =
   (((verbser(syn(num(s),tense(p))),(((l,X),(l,Y)),(X,Y)),es)      ,
      (defngins(sem(con(architect))),(Juan),ins)),
    ((2, 3), 1),ac_ciwn(sem(con(architect))),criwno))

precondition list = (pcon(architect))
```

The only rule that can be executed next is Rule 3. This rule pops the lexical entry from the top of **alpha** and pushes it into **gamma**. It also pops to the top of **beta**, the category **indefngcon(syn(gen(G),num(s)),sem(con(architect)))** that has matched the category **indefngcon(syn(gen(m),num(s)),sem(con(architect)))**, associated with the recognized item. The resulting stacks are:

```
stack alpha =  ( )

stack beta =  (( ) ,c)

stack gamma =
 (((indefngcon(syn(gen(m),num(s)),sem(con(architect))),(architect),un_arquitecto),
    (verbser(syn(num(s),tense(p))),(((l,X),(l,Y)),(X,Y)),es)      ,
    (defngins(sem(con(architect))),(Juan),ins)),
  ((2, 3), 1), ac_ciwn(sem(con(architect))),criwno))

precondition list = (pcon(architect))
```

The top of the **beta** stack is an empty list, indicating a rule has been recognized. At this step, Rule 2 is the only one that can be executed. Information stored in **gamma** is used to interpret the lexical entries recognized by the last rule.

Following the semantic list in the rule (**((2, 3), 1)**), the semantic interpretation of the item recognized by the second category is applied to the semantic interpretation of the item recognized by the third category. That is, the lambda function associated with the category **verbser(syn(num(s),tense(p))** is applied to the lambda value associated with the category **indefngcon(syn(gen(m),num(s)),sem(con(architect)))**. The result of applying the function **((l, X), (l, Y), (X, Y))** to the value **architect** is the function **((l, Y), (architect, Y))**. This resulting function is then applied to the lambda value associated with the category **indefng(sem(con(architect)))**. As a result, the list **(architect, Juan)** is obtained.

Next, additional information necessary for executing the operation expressed by the sentence is incorporated into the semantic interpretation. This information consists of the left-hand category of the grammar rule (**ac_ciwn**), the name of the operation associated with the rule (**criwno**) and the name of the concept over which the operation is performed (**architect**). The resulting semantic interpretation is:

```
( ac_ciwn, criwno, architect, architect, Juan)
```

This semantic interpretation indicates that the sentence expresses the operation **criwno**, for

creating a conceptual instance giving its name. The two parameters of this operation are the name of the concept, **architect** and the name of the instance **Juan**.

The execution of Rule 2 will also pop up a list containing the left-hand category, the semantic interpretation and a sublist with the categories recognized and their corresponding strings into **alpha**. The resulting stacks will be:

```
stack alpha =
  ((ac_ciwn(sem(con(architect))),(ac_ciwn,criwno,architect,architect,Juan),
    ((verbser(syn(num(s),tense(p))),
     indefngcon(syn(gen(m),num(s)),sem(con(architect))),
     defngins(sem(con(architect)))), (es ,el_arquitecto,ins))))

stack beta =  ( c )

stack gamma =  ( )

precondition list = (pcon(architect))
```

At this step, the only rule to execute is Rule 1, once more. A new rule to reach the category **ac_ciwn(sem(con(architect)))** from the top category on **beta**, **c**, must be selected. The only rule that can be applied is:

```
pcg ([])  c  ->   ac_ciwn(sem(con(C))) ( ) ( )
```

The left-hand category of this rule (**c**) and its semantic information (two empty lists) is pushed into **gamma**. The semantic interpretation of the rules having only one category in the right-hand part, such as this rule, consists of an empty list. The second list indicates that the rule does not express any operation. The item containing the category recognized is also pushed into **gamma**. Once Rule 1 is executed, the information stored in the stacks is:

```
stack alpha =  ( )

stack beta =  (( ) , c)

stack gamma =
  (((ac_ciwn(sem(con(architect))),(ac_ciwn,criwno,architect,architect,Juan),
    ((verbser(syn(num(s),tense(p))),
     indefngcon(syn(gen(m),num(s)),sem(con(architect))),
     defngins(sem(con(architect)))), (es ,el_arquitecto,ins)))),( ),c,( ))

precondition list = (pcon(architect))
```

Again the top in **beta** is an empty list, indicating that all categories in a rule have been recognized. Rule 2 must be applied. The categories recognized by the rule as well as the information stored about the rule are popped from **gamma**. The semantic analysis must be undertaken. When there is only one category in the rule, the resulting semantic interpretation corresponds to the semantic interpretation of the item recognized by the rule. A list containing the left-hand category of the last rule processed, the categories recognized, together with the resulting semantic interpretation is popped into **alpha**.

The information contained in the stacks at this stage is:

```
stack alpha =
((c,(ac_ciwn,criwno,architect,architect,Juan),
    ((verbser(syn(num(s),tense(p))),
     indefngcon(syn(gen(m),num(s)),sem(con(architect))),
     defngins(sem(con(architect)))), (es ,el_arquitecto,ins))))

stack beta =  ( c )

stack gamma =  ( )

precondition list = (pcon(architect))
```

Rule 3 is applied next. The only category to recognize (the category of the only element in **alpha**) is the initial category (the only element in **beta**) and it corresponds to the **root** category, **c**. The top of **alpha** and **beta** are popped. As a result, all three stacks are empty. Thus, the final state is reached. The element popped from **alpha** is stored as the final result. It contains three elements: the root category, the syntactic tree, and the final semantic interpretation.

7.3.4 The parser data structures

Previously to the analysis process, the global data structures containing information about the grammar and lexicon are built. These data structures are represented as Prolog predicates and stored in the program database.

These data structures are described below:

sintactica. This data structure represents each grammar rule. Rules are represented following the format:

sintactica(Pcc, Num, [Left, Sem, Mom, Right], Op)

where **Pcc** represents the preconditions attached to the rule; **Num** is the number identifying the rule; **Left** is the left-corner category; **Sem** contains the interpretation order of the constituents of the rule; **Mom** is the category of the left-hand part of the rule; **Right** is the list of the categories in the right-hand part of the rule without the left corner and **Op** is the operation expressed by the rule (if the rule does not express any operation it is an empty list).

lexic. This data structure represents all lexical entries. Each lexical entry is represented following the format:

lexic(String, [Cat, Sem, String])

where **String** is the realization, **Cat** is the category and **Sem** is the semantic interpretation.

accessible. This data structure represents all reachable categories from the left-corner categories. Because a left-corner parser is used, information about the categories reachable from each left-category is frequently required. Compiling this information improves the performance of the parser. The categories reachable from each left-corner category are represented following the format:

accessible(Left-corner, Reachable)

where **Left-corner** is the left-corner category of a grammar rule and **Reachable** is a list containing all categories that can be reached from the **Left-corner**.

sfinals. This data structure represents the categories in the left-hand part of the rules expressing operations. It also includes the root category. This information is consulted when building semantic interpretation. The category of the left-hand part of a rule is incorporated (without its associated features) into the list resulting from the semantic analysis corresponding to the rule. This global variable is represented as follows:

sfinals(ListFinalCategories)

where **ListFinalCategories** contains all categories in the left-hand part of the rules expressing operations.

The compiler also builds global data structures to adapt the parsing to the menu-guided

NLI. These data structures facilitate the process of displaying all possible NL options on screen that the user can choose when building a sentence. The data structures contain information about the terminal categories and their distribution in menus (or windows). These data structures are:

terminals. This structure represents all terminal categories in the grammar. The information is represented in the format

terminals(ListTerminalCategories)

grup. This structure represents the distribution of the lexical entries in menus. Lexical entries having the same category are grouped and represented following the format:

grup(Cat, Entries, Menu)

where **Cat** is a category, **Entries** is the list of superficial presentations associated with this category and **Menu** is the name of the menu where they will be displayed.

All lexical entries that can be accepted will be displayed on screen distributed in menus regarding their **upper category**, that is, the general class they belong. These menus are represented as a global data structure to improve the NLI performance. Although there is a basic configuration of menus, it can be modified when desired.

There is also conceptual information represented in this module. There are global variables containing information about the CO application level, where the application is modeled. There is a global variable representing the values of the conceptual attributes of menu type. The set of possible values of each of these attributes is displayed as a menu (or window) during communication. This global variable is called **menu**.

The data structures representing the set of the possible values for these attributes follow the form:

menu (Menu, Attribute, Value)

where **Menu** represents the name of the menu, **Attribute** represents the attribute identifier and **Value** represents the list of all possible values for the attribute.

Preconditions associated with the CO concepts are also represented as data structures in the database to improve the performance of the analysis. This information is represented in the data structure **pcon**.

The preconditions attached to the concepts are represented following the format:

pcon (Con) :- Preconditions

where **Con** represents the concept and **Preconditions** represents the set of predicates that must be executed to create an instance of the concept.

The preconditions governing the filling of an attribute of a conceptual instance are represented following the format:

pcon (Con, Attribute) :- Preconditions

where **Attribute** represents the attribute to fill.

Verifying the grammar

The possible errors in the grammar and lexicon are detected before building the parser data structures, described above.

The grammar and lexicon generated by GISE for a specific application are obviously error free but they can be modified manually by the user, if desired. There is also the possibility of incorporating a grammar completely developed by the user in the NLI. Different types of errors could be introduced in the case of the grammar having been modified manually or having been generated by control rules defined by the user.

In the case of the grammar having been generated by one of the basic set of control rules described in Chapter 6, then the only possible corrections consist of eliminating non-necessary information. This information basically consists of repeated grammar rules and general lexical entries that are not required in the specific application.

The compiler detects errors in the form, consistence, accessibility, redundancy and cycles. It eliminates all grammar rules and lexical entries not correctly expressed (not following the form described above) as well as those that are redundant. The compiler also detects the rules that are not accessible from the root category and those that belong to a cycle. Finally, consistence between the grammar and lexicon is also checked: all terminal categories of the grammar (those not appearing in the left-hand part of any rule) must appear in the lexicon and the categories in the lexicon must be terminal grammar categories.