A Web Information Extraction System to DB Prototyping*

P. Moreda, R. Muñoz, P. Martínez-Barco, C. Cachero and M. Palomar

Grupo de investigación del Procesamiento del Lenguaje y Sistemas de Información.
Departamento de Lenguajes y Sistemas Informáticos. Universidad de Alicante.
Alicante, Spain
{moreda,rafael,patricio,ccachero,mpalomar}@dlsi.ua.es

Abstract. Database prototyping is a technique widely used both to validate user requirements and to verify certain application functionality. These tasks usually require the population of the underlying data structures with sampling data that, additionally, may need to stick to certain restrictions. Although some existing approaches have already automated this population task by means of random data generation, the lack of semantic meaning of the resulting structures may interfere both in the user validation and in the designer verification task.

In order to solve this problem and improve the intuitiveness of the resulting prototypes, this paper presents a population system that, departing from the information contained in a UML-compliant Domain Conceptual Model, applies Information Extraction techniques to compile meaningful information sets from texts available through Internet. The system is based on the semantic information extracted from the EWN lexical resource and includes, among other features, a named entity recognition system and an ontology that speed up the prototyping process and improve the quality of the sampling data.

1 Introduction

Nowadays, both practitioners and researchers avow the necessity of conceptual models to design and deploy high-quality non-trivial applications. Following this trend, well known Software Engineering practices, usually based on the *de facto* standard in industry UML [14], are being adopted. However, the yet insufficient number of support for automatic tasks that ease the implementation and testing of the corresponding software artifacts is diminishing the potential benefits of such models. We agree with [1] in that Advanced Software Production Environments that include Model Based Code Generation and Testing Techniques that partially automate the development process are crucial for organizations to maximize the results of applying conceptual modeling concepts.

* This paper has been supported by the Spanish government, projects TIC2000-0664-C02-01/02 and TIC2001-3530-C02-01/02

In this sense, efforts made towards such automation have already showed some interesting results. Most current software development processes already include semi-automatic translation of concepts among models, thus promoting the reuse of design specifications and avoiding inconsistencies. Closer to our objective, efforts in the Requirement Analysis field are being made to generate each time more refined conceptual models out of textual descriptions. In fact, we believe that textual analysis is bound to pay an ever increasing role in this automated process, as will be explained in Section 3. This paper presents one of its application possibilities: the population of application prototypes with sampling data that provide meaningful content on which to test and get stakeholder feedback.

2 Class Diagram definition and constraints

A Class Diagram is a graphic view of the static structural model of the system [14]. Being a milestone in the conceptual modeling process, it includes domain-related textual information in the form of descriptors (text strings) associated to each of its elements. Due to its nature, the Class Diagram constitutes an invaluable source for domain knowledge: the UML notation helps not only in the conceptual understanding of the domain, but also provides an ontology over the domain vocabulary.

In order to illustrate the whole sampling data extraction process, a small example is going to be employed all along the paper: a *Hotel Reservation System*. The UML class diagram corresponding to such system can be observed in Fig. 1.

In this system, and as a basic explanation (for reasons of brevity) let's assume each hotel has a set of *Rooms (Habitación)* which can be of different *Types (TipoHabitación)*. Rooms can be booked by *Clients (Cliente)*. For each *Booking (Reserva)* the system keeps track of the *Services (Servicio)* provided (laundry, drinks, etc), in order to *Charge (Cargo)* them to the client. On client departure, an *Invoice (Factura)*, which may include more than one *Booking*, is generated and the method of *Payment (MedioPago)* is registered. In this diagram, all classes revolve around a class decorated with a standard *singleton* UML stereotype. UML Stereotypes are a very powerful standard extension mechanism that provide the modeling constructs with additional semantics, in this case the existence of a single instance of the *Hotel* class. This class constitutes the Domain Class, that is, the class that establishes the domain context, whose utility will be shown in section 4.2[1].

2.1 Nomenclature constrainst

In order to facilitate the text analysis, it is necessary to define a set of rules for class, attribute and relationship names that complement the general UML

[1] A whole explanation of Class Diagram design rules is out of the scope of this article. Interested readers are referred to [14].

notation rules. These rules make easy the task of name reconstruction as will be explained in 4.1.

Several features must be considered for each component of the class diagram (classes, attributes and relationships).

Class names. The rules to take into account for the definition of a class name are the following:

1. A simple noun beginning with a capital letter is a class name –i.e. Factura (*Bill*)–.
2. An usual abbreviation of language beginning with a capital letter is a class name –i.e. Tlfn (*Tel*) is the abbreviation of *Telephone number*–.
3. An acronym is a class name –i.e. CP (*Zip*) is the acronym of *Postal Code*–.
4. A complex name made up of any element above described is a class name –i.e. TipoHabitación (*RoomType*)–.

Attribute names. The way of defining attribute names follows rules similar to those applied to class names, its main difference lying in that the attribute name begins with a small letter. These rules are the following:

1. A simple noun beginning with a small letter is an attrbute name –i.e. categoría (*category*)–.
2. An usual abbreviation of language beginning with a small letter is an attribute name –i.e. tlfn (*tel*) is the abbreviation of *telephone number*–.
3. An acronym is an attribute name –i.e. NSS is the acronym of *Social Security Number*–.
4. A complex name made up of any element above described is an attribute name –i.e. numFactura (*billNumber*)–. In order to make easy the name reconstruction task, the attribute names will be split into several words by capital letters.

Relationship names. Last, relationship names must follow the following rules:

1. A verb in its infinitive form beginning with a capital letter is a relationship name –i.e. Reservar (*to book*)–.
2. A verb in its infinitive form beginning with a capital letter plus a class name is a relationship name –i.e. HacerCargo (*to make charge*)–.

3 Prototyping

All the rules presented above simplify the definition of a search activity to find meaningful data that could be part of the system population. Even in automated environments this 'sampling population task' has traditionally been (up to our

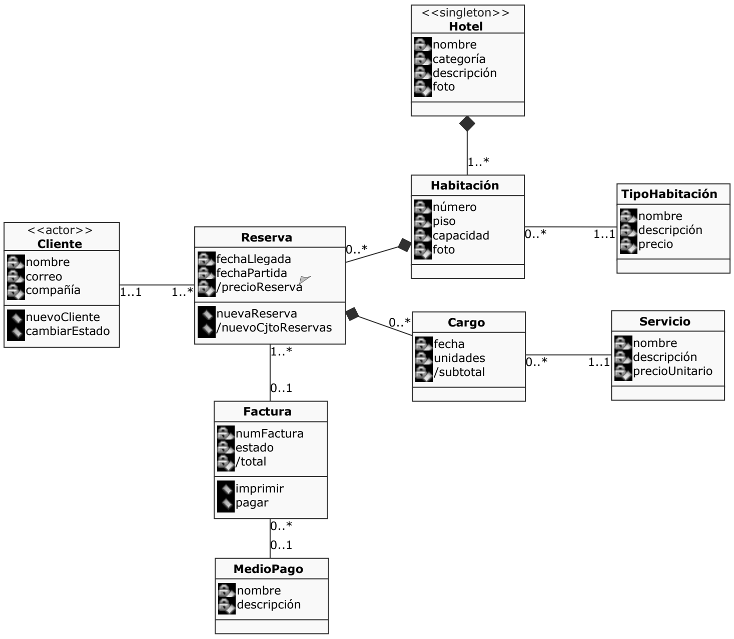

Fig. 1. *Example of a class diagram*

knowledge) left out to the designer, who, either manually or by random generation routines had to create and maintain the necessary 'testing sets'. Although these simple techniques have sufficed for a long time, both the increasingly important role of stakeholders in the software development process and some empirical observations suggest that meaningful sampling data helps to improve not only the stakeholder perception of the application under development, but also facilitates the verification and validation process. Another useful characteristic of the use of meaningful data during the development process is that it may help to refine design aspects. Perhaps one of the most straightforward ones is the evolution of the models to deal with data formats not previously considered. Furthermore, we may be interested in maintaining certain restrictions on the sampling data that permit the verification and/or validation of certain parts of the application. Letting the designer manually assure that these restrictions are met may become cumbersome. All these reasons justify, from our point of view, the use of natural language processing activities such as the ones presented next.

4 Natural Language Processing

As we have stated above, the goal of this paper is to identify some information called relevant in real texts in order to fill the underlying data structures. For this task, Information Extraction (IE) techniques, an application of Natural

Language Processing, are used. Information extraction is the name given to any process that structures and combines data which is found, explicitely stated or implied, in one or more texts. It is important to stress that we do not aim at developing an information extraction system, but at taking advantage of this technique to select data that provides content to the domain models.

In the following subsections we will show the Information Extraction techniques that have been used in this system, as well as the constraints that must be fulfilled in the Class Diagram to help the automation of the sampling data.

4.1 Name Reconstruction

In order to obtain the word sense through WordNet it is necessary to make a construction of class, attribute and relationship names from the Class Diagram. In this construction, a dictionary of acronyms and abbreviations is used. This dictionary returns their full names. Each entry in the dictionary relates the acronym or the abbreviation to its full name. Moreover, this dictionary could be extended with new user-defined entries.

Finally, this dictionary is used together with an appropriate set of heuristic rules (e.g. abbreviation + Capital word → Full word + "of" + word, –*numBill (number of bill)*–) to accomplish the reconstruction of the names. This information will be used in following steps.

Next, an overview of this process is presented.

Class and attribute name reconstruction process

1. If the name is a composed name, then it is split into single words separated by blank spaces building a composed term (e.g. *numBill* → *num bill*).
2. Each single word is looked for in the dictionary of acronyms and abbreviations. If found, the abbreviate word in the composed term is updated with the full word. (e.g. *num* → *number*).
3. The term is looked for in WordNet. If found, the process finish.
4. Blank spaces in the term are replaced with underline chars. (e.g. *number bill* → *number_bill*).
5. The term is looked for in WordNet. If found, the process finish.
6. The preposition 'de' '*of*' is inserted between each pair of consecutive names in the term (with blank spaces). (e.g. *number bill* → *number of bill*).
7. The term is looked for in WordNet. If found, the process finish.
8. Blank spaces in the term are replaced again with underline chars.(e.g. *number of bill* → *number_of_bill*).
9. The term is looked for in WordNet.

Relationship name reconstruction process

1. If the name is a composed name, then it is split into the verb (left part) and the remainder words separated by blank spaces building a composed term. (e.g. *ChangeNumBill* → *change numBill*).

2. The verb is looked for in WordNet.

The remainder words are reconstructed with the class name reconstruction process (e.g.*numBill* → *number bill*). This process will be used if it is necessary the disambiguation of the verb.

4.2 Domain Recognition

One of the main tasks that must be accomplished in any kind of web searching is to avoid the problem with word sense ambiguity. In this sense, the part-of-speech of this word is an important clue to detect ambiguities. For instance, the word '*plant*' has a different meaning when it acts as a verb from a noun.

Moreover, the same word with the same part-of-speech could have several senses when it is defined in different domains. For instance, the noun '*mouse*' has a different meaning if it is defined into a computer-science domain or in an animal one.

Furthermore, also the same word with the same part-of-speech inside the same domain could have different senses. In this case, the clue is the belonging to a defined subdomain. This is, for example, the different sense of the word '*charge*' in a hotel domain that could mean the price charged to the custom for the service, or the task that has been assigned to a person in the hotel.

As above shown, our proposal is based on the use of the names of attributes to find possible values to them. Unfortunately, the name of an attribute is not free from a possible ambiguity, as well as the information that is going to be extracted from the World Wide Web. So, ambiguity must be solved in two different fields, on one hand, in the attribute name, and on the other hand, in the information to be extracted from the WWW.

In order to solve the ambiguity in the attribute name, our proposal makes use of the following knowledge:

- **Part-of-speech (POS)**. POS information is known in a class diagram. In this way, attribute or class names are usually nouns. However, the name of a relationship is a verb.
- **Domain definition**. Domain is one of the clues to word sense disambiguation. In order to establish the domain in which the class diagram has been defined, every class diagram is forced to have a DOMAIN CLASS in which information about the general domain of the diagram is user-defined. This domain must be selected from a domain ontology. The definition of this ontology is based on the domain classification used by web searchers as we will further explain.
- **Subdomain recognition**. In those cases in which domain is not enough to word sense disambiguation, a subdomain recognition must be performed. In our proposal a **subdomain recognition module** for class attribute names has been developed. This module is based on heuristic rules that relate attributes with their classes and the relationships between their classes and other classes. For instance, '*name*' is a very usual word as as attribute

name. However, it has no sense in its own. To identify this sense we must know information about the class which it belongs to. In this case, if the name class is '*Person*' the sense of this '*name*' is, without doubt, inferred.

Once all the information to solve the ambiguity is obtained, then an unique word sense is obtained through the **WordNet consulting module**. This module performs a consult in the lexical database WordNet, and filters the result with POS, domain, and subdomain values.

Other problem to solve is the ambiguity in the information that is going to be extracted from WWW texts. Again, some additional information about words must be included in order to define their concrete sense. In this case our system is equipped with some tools providing this kind of information:

– **POS tagger**. This tool provides the information about part-of-speech for every word in the text that is being analyzed.
– **Domain web searcher**. Our system takes advantage of the domain classification developed by any web search engine. By means of this kind of web search engines, the searching scope is reduced to web pages included in this domain. To be more precise, we will reduce the scope to the domain defined in the DOMAIN CLASS of the class diagram. That is the reason why the ontology used to define these DOMAIN CLASSES is based on the web searchers domain classification.
  In this work we have used the Google web search engine (`www.google.com`). Consequently, the ontology used to define the DOMAIN CLASS has been extracted from the one define in Google. The top level of this ontology is shown in Fig.2.

Fig. 2. *Top level of the Google ontology*

– **Word sense disambiguator (WSD)**. Finally, a WSD system is used to disambiguate those cases in which POS and domain are not enough to extract a single sense. In this way, the WSD system developed by [9] is being used. This tools provides a single WordNet label for each word.

Once we have clarified the way of obtaining a single sense for both attribute names and the information that is going to be extracted, then a compatibility feature must be identified between them. This feature determines if the web information fits the attribute name, so it is useful to fill this structure.

However, this feature not only needs to perform a direct comparison between the WordNet senses, but also needs to apply some semantic relationships as defined in WordNet namely the synonym, hyperonym and hyponym relationships between those senses to relax this compatibility feature.

This task provides all needed information to the Named-Entity recognition module that we will explain in the next subsection.

4.3 Named-Entity Recognition

The main technique used to identify real information is the named-entity recognition technique. According to MUC[2] [11], we can distinguish different kinds of categories of named entities: person, organization, location, dates, currency, etc. Named entity recognition involves processing a text and identifying certain occurrences of words or expressions belonging to a category of Named Entity. Different approaches for Named Entity recognition have been developed at MUC competition. Most of them use gazetteers and list of names to help to recognize the entities [13, 10, 4]. Obviously, we will need huge lists to guarantee a high score. However, Cucchiarelli et al. in [2] report that one of the bottlenecks in designing Name Entity recognition systems is the limited availability of large gazetteers. Mikheev et al. in [8] develop a Name Entity recognition system which combines rule-based grammars with statistical models (maximum entropy).

Our Named Entity recognition system can be split in the following steps:

1. **Name identification**. The first problem to be solved is the identification of the boundaries of a named entity. Entities can be complex, consisting of several words. This is specially common when conjunctions, prepositions and definite articles are involved. In Spanish, it is common to find full names with prepositions and definite articles (Joaquín de la Cierva) or conjunction (Santiago Ramón y Cajal). Moreover, in Spanish like in English, the name of companies can be made up of any kind of word (bare nouns, adjectives, numbers, conjunctions, etc.). According to McDonald [7], we can use internal and external evidence to recognize a named entity and to classify it as belonging to the adequate category. The following rules are applied in order to establish the starting and end point of an named entity:

[2] Conferences organized to define the different task accomplished by IE system and to evaluate them using a specific domain.

(a) Looking for a trigger. Internal evidences notify the presence of named entities. These internal evidence can be made up of several specific words called triggers. Every kind of category has specific triggers. The names of companies frequently use corporate designators such as S.A., S.L., Co., Cia., Ltd., Inc., etc. The names of people frequently use person titles, such as Sr., Sra., Mr., Mrs., etc. Also, the names of locations may use words such as Ciudad (City), Calle (Street), Avda. (Ave.). In figure 3, a short sample set of rules is shown.

```
PERSON

Title(Sr., Sra.,) + CapWord                                   Sr. Indurain
Title(D., Dña.,) + CapWord + CapWord                    D. Cristobal Colon
Title(D., Dña.,) + CapWord + CapWord + CapWord          D. José Luis Pérez

ORGANIZATION

Designator(S.A., S.L.,) + {CapWord}                        S.A. El Águila
{CapWord} + Designator (S.A., S.L.,)                          Repsol S.A.

LOCATION

Trigger(C/, Avda., Plza.) + {CapWord}                     Avda. Cataluña
```

Fig. 3. *Example of rules to check the entity triggers*

(b) Grouping all capital words. However, these previous triggers do not always appear with the named entity. In this case, other internal evidence is needed to identify the named entity, such as the use of partial orders of the composing words. Examples of this fact are shown in figure 4.

```
PERSON

ChristianName + CapWord                                  Miguel Indurain
ChristianName + CapLetter + CapWord                    Juan A. Samaranch

LOCATION

{CapWord}+ "," + CountryName                            Alicante, España
```

Fig. 4. *Example of rules to check the entity without triggers*

In addition to internal evidence, external evidence can be used to identify some entities. The external evidence is made up of a set of rules taking advantage of some words that appear next to entities. These special words depend on the domain work. For this reason, we use the name classes and attributes from the class diagram to identify some entities together with all the words semantically related to the appropriate sense provided by the WSD. Moreover, we also use EuroWordNet to find semantic names related to the name classes and attributes, as we have previously described. Figure 5 shows an example of this kind of rules in the context of our domain example.

```
PERSON

"director" + ChristianName + CapWord          el director Miguel Indurain

ORGANIZATION

"director" + "of" ChristianName + CapWord            el director of Melia
LOCATION

"located in" + {CapWord}+ "," + CountryName           located in Alicante
```

Fig. 5. *Example of rules using external evidence*

2. **Name entity Disambiguation**. Once the named entity is identified, several problems can be found. This task aims at solving the following problems:
   - Ambiguity capitalized words. Sometimes, the first word of the sentence (capital word) appearing next to the entity can be identified as a part of the entity name ([The Melia] offers... vs. The [Melia] offers...). This problem is solved by matching previous appearances of the entity. If it is the first appearance, both structures are considered, and the removal of one of them is delayed until later steps.
   - Semantic ambiguity. The structure of different named identities (person and location, person and organization, etc.) can be similar. Typical examples are extensively shown in literature, (e.g. John F. Kennedy, Philip Morris). Our named entity recognition system uses the semantic information associated to the verb of the sentence and the top ontology of WordNet in order to solve such problem.
   - Structural ambiguity. Different types of problems can be classified within structural ambiguity. All structural problems are caused by the use of prepositions and conjunctions. This type of words cause the problem of the limitation of the entity.
     - Use of Conjunctions. The appearance of a conjunction between capital words causes a doubt related to whether to add both capital words to the same entity ([Construcciones y Contratas S.A.] construye la autopistas... –[Construcciones y Contratas S.A.] builds the motorway...–) or to split them in two different entities ([Real Madrid y P. Vieira] acuerdan reunirse de nuevo... –[Real Madrid and P. Vieira] agreed to

meet again...–). The use of verb's number (singular, plural) can help to take a decision about to add or to split the words. According to [12], the study of triggers can also to help us to decide the formation of the entity.

- Use of Prepositions. The use of preposition as part of the named entity is mainly used in Organization and Location named entities ([Center for Scientific Researchs]...). This fact can cause some problems when two entities are related using this type of word (notificación del [COI] para [Miguel Indurain]... – notification of [COI] for [Miguel Indurain]). This kind of problem is the most difficult to solve. Our system uses a list of words, such as center, that usually appears next to a preposition and a set of capital words making up an unique entity.

4.4 No named-entity recognition

Frequently, attributes from class diagram requiere values made up of bare nouns instead of proper nouns. For example, an attribute as *Position (Categoría)* from *Employer (Empleado)* can be fill with *Recepcionist (Recepcionista)* and *Waiter (Camarero)*.

The name reconstruction of attributes and classes, the WSD and the external evidence play an important role to recognize them from the text. The external evidence use the name reconstructed to match into the text all occurrences of the name and its semantic related word.

5 NLP Algorithm

Next, a general view of the NLP algorithm that has been used is shown:

1. The class diagram defined according a set of nomenclature constraints is translated into a textual XML structure showing attribute, class, and relationship names.
2. These names are reconstructed in order to obtain their equivalents in natural language.
3. Using the Domain Class, the main domain of the Class Diagram is recognized.
4. For each attribute name, its sub-domain is recognized by means of the class in which it is included. The attribute names together with their domains and sub-domains are used to access WordNet and extract their semantic related words (synonyms, hyperonyms, and hyponyms).
5. Using the Google web search engine we obtain the documents containing any of these attribute and class names, or their semantic related words. To guarantee the correct domain of the document, this web search is performed through the Google directory that includes the Domain Class defined in the Class Diagram.

6. The sentences where the goal words appear are parsed by means of a POS tagger and a WSD to solve the ambiguity, discarding incorrect semantics senses.
7. Once the texts were relevant information can be founded have been selected, then the Named-Entity recognition is applied to recognize proper nouns and bare nouns associate to goal words.
8. Finally, the extracted information is provided to a Filling module using XML tags.

6 Filling the Schema

The last step in the sampling data extraction task is the load of such data in the underlying data structures. Due to the fact that this process differs depending on the underlying storage system, our system provides a middle step in which an XML implementation-independent description of the data found during the information extraction process is provided. Such description does not need to be complete. Some attributes can belong to domains such as *datatime* or *number* and for them random data can be generated. In this random data generation the load module must take into account invariants associated to the Class Diagram constructs. For example, imagine we have associated an OCL expression to the class *TipoHabitación* (see Fig. 1) that establishes that any room price must be between 80 and 200 Euros (*precio¡200 and precio¿80*). Of course, the random numbers generated for such attribute must stick to such restriction to provide sound information.

Once this XML template has been modified with such random data, the actual load process of the system must be performed. In the actual stage of development, we have restricted ourselves to underlying relational structures which are directly derived from the modeled Class Diagram. This load module receives, in addition to the XML data file, another XML document that, by means of rules, gathers general constraints that must be fulfilled by the system population for a given purpose. As a further example, imagine we need at least one unpaid invoice (*Factura.estado=pendiente*, see Fig. 1) in order to be able to verify and validate the functionality associated to such situation. This fact would be modeled by means of the following rule:

¡TSampling¿ ... ¡rule context="Invoice"¿ ... ¡condition name="AtLeastOneInvoice" mandatory="yes" minOcc="1" maxOcc="" value="estado=pendiente"/¿ ... ¡/rule¿ ... ¡/TSampling¿

Inside this rule, we can observe how the mandatory character of the rule determines whether the NLP algorithm can finish or, on the contrary, must continue looking for new sampling data. The data set degree of compliance to the non mandatory rules determines its quality for the system purpose.

7 Conclusions

Prototyping is widely used to validate and verify the functionality of Information System applications. However, nowadays prototyping tools are more focused on designing than data filling task. In this paper we have presented an automatic system that is able to avoid the hard manual task that developers of prototypes must perform in order to obtain test data. This system looks for real data in web documents to accomplish this goal. Taking advantage of Information Extraction techniques, our system uses a named entity recognizer to obtain the specific information to be related to classes and attributes of the DC. This DC must fulfill a minor set of constraints in order to aid the automation.

As a result, the prototype will be filled with real and understandable information that helps in the verification and evaluation process of the final application.

Despite this system has been successfully tested, several issues must be improved in further versions of our system. These issues are related to the following ideas:

- In our proposal a small ontology based on the Google domains has been used. However, a complete ontology must be defined to allow its use on whatever kind of domain.
- The semantic fine-grained of WordNet produces errors in the WSD module. The WSD module is forced to choose among different senses (very close related) despite all of them belongs to the same domain. To solve this problem, future work will apply the Magnini domain WordNet [6, 5] that clusters close related senses.

The main contributions of this work are:

- Entity recognition system based on UML Class Diagrams
- Ontology
- Data insertion strategy

References

1. R. Bell. Code Generation from Object Models. *Embedded Systems Programming*, 3:1–9, 1998.
2. A. Cucchiarelli, D. Luzy, and P. Velardi. Automatic semantic tagging of unknown proper names. In ACL, editor, *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL'98)*, pages 286–292, Canada, 1998.
3. Morgan Kaufman, editor. *Sixth Message Understanding Conference (MUC-6)*, Los Altos, Ca, November 1995.
4. G. R. Krupka. Description of the SRA system as used for MUC-6. In Kaufman [3], pages 71–86.
5. B. Magnini and G. Cavaglia. Integrating subject field codes into WordNet. In *Proceedings of the LREC-2000*, 2000.
6. B. Magnini and C. Strapparava. Experiments in Word Domain Disambiguation for Parallel Texts. In *Proceedings of the ACL Workshop on Word Senses and Multilinguality*, 2000.
7. D. McDonald. Internal and external evidence in the identification and semantic categorization of proper names, 1996.

8. A. Mikheev, M. Moens, and C. Grover. Named Entity Recognition withou Gazetteers. In ACL, editor, *Proceedings of the 11th European Chapter of the Association for Computational Linguistics (EACL)*, pages 1–8, Norway, 1999.
9. A. Montoyo, A. Suárez, and M. Palomar. Combining Supervised-Unsupervised Methods for Word Sense Disambiguation. In Alexander Gelbukh, editor, *Proceedings of 3nd International conference on Intelligent Text Processing and Computational Linguistics (CICLing-2002)*, volume 2276 of *Lecture Notes in Computer Science*, pages 156–164, Mexico City, 2002. Springer-Verlag.
10. R. Morgan, R. Garigliano, P. Callaghan, S. Poria, M. Smith, A. Urbanowicz, R. Collingham, M. Costantino, and C. Cooper. Description of the LOLITA system as used for MUC-6. In Kaufman [3], pages 71–86.
11. MUC-7. Saic:science aplications international corporation. `http://www.itl.nist.gov/iaui/894.02`, 2001.
12. R. Muñoz, A. Montoyo, F. Llopis, and A. Suárez. Reconocimiento de entidades en el sistema EXIT. *Procesamiento del Lenguaje Natural*, 23:47–53, 1998.
13. B. Sundheim. Overview of results of the MUC-6. In Kaufman [3], pages 13–32.
14. OMG Unified Modelling Language Specification. http://www.rational.com/, 2001.