Accepted Manuscript

A multi-agent conversational system with heterogeneous data sources access

Eduardo M. Eisman, María Navarro, Juan Luis Castro

PII: S0957-4174(16)00056-7
DOI: [10.1016/j.eswa.2016.01.033](https://doi.org/10.1016/j.eswa.2016.01.033)
Reference: ESWA 10491

To appear in: *Expert Systems With Applications*

Received date: 2 June 2015
Revised date: 20 January 2016
Accepted date: 21 January 2016

Please cite this article as: Eduardo M. Eisman, María Navarro, Juan Luis Castro, A multi-agent conversational system with heterogeneous data sources access, *Expert Systems With Applications* (2016), doi: [10.1016/j.eswa.2016.01.033](https://doi.org/10.1016/j.eswa.2016.01.033)

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Highlights

- We present a multi-agent conversational architecture for heterogeneous data sources.
- Expert agents are specialized in accessing different knowledge sources.
- Decision agents coordinate expert agents to provide a coherent final answer to users.
- This generic architecture is used to make a SmartSeller for a bookstore.
- A comparative analysis demonstrates several improvements regarding existing systems.

A multi-agent conversational system with heterogeneous data sources access

Eduardo M. Eisman^{a,*}, María Navarro^b, Juan Luis Castro^a

^aDepartment of Computer Science and Artificial Intelligence, University of Granada, Spain

^bCEMSE Division, King Abdullah University of Science and Technology (KAUST), Thuwal, Kingdom of Saudi Arabia

Abstract

In many of the problems that can be found nowadays, information is scattered across different heterogeneous data sources. Most of the natural language interfaces just focus on a very specific part of the problem (e.g. an interface to a relational database, or an interface to an ontology). However, from the point of view of users, it does not matter where the information is stored, they just want to get the knowledge in an integrated, transparent, efficient, effective, and pleasant way. To solve this problem, this article proposes a generic multi-agent conversational architecture that follows the divide and conquer philosophy and considers two different types of agents. Expert agents are specialized in accessing different knowledge sources, and decision agents coordinate them to provide a coherent final answer to the user. This architecture has been used to design and implement SmartSeller, a specific system which includes a Virtual Assistant to answer general questions and a Bookseller to query a book database. A deep analysis regarding other relevant systems has demonstrated that our proposal provides several improvements at some key features presented along the paper.

Keywords: Natural language interfaces, Virtual assistants, Embodied conversational agents, Multi-agent systems, Semantic grammars, Ontologies

1. Introduction

As many studies reveal (Chai et al., 2001a,b; Kaufmann and Bernstein, 2010; Zhou et al., 2012), there is a clear preference of users for full natural language query interfaces rather than keywords, formal query languages, or menu driven interaction. In addition, the interest of users in a particular site decreases exponentially with the increase in the number of mouse clicks (Huberman et al., 1998). This fact is emphasized even more if we talk about mobile devices, where traditional input interfaces are very limited. Natural language systems are able to improve the perceived usefulness, ease-of-use, and efficiency, which in turn account for positive attitude and intention to use those systems.

From an economical point of view, the increase of e-commerce spending supposes a great opportunity for natural language interfaces. According to comScore¹ (one of the most important Internet marketing research companies), Q1 2014 saw desktop-based U.S. retail e-commerce spending rise 12% year-over-year to \$56.1 billion, marking the eighteenth consecutive quarter of positive year-over-year growth. M-commerce spending on smartphones and tablets added \$7.3 billion for the quarter, up 23% vs. year ago, for a digital commerce spending total of \$63.4 billion in the first quarter. In addition,

some events like Alibaba's² Singles' Day sales confirm that m-commerce is more and more important. During that day, the world's biggest online retail sales day, sales exceeded predictions at \$9.3 billion, shipping 278 million orders, 43% of which were placed on mobile devices³. This change in consumer behavior reflects the necessity of defining more intelligent ways of interacting with websites and product databases rather than traditional keyword search.

This paper focuses on the problem of accessing heterogeneous data sources using natural language dialog. As we will see in Section 2, most of the natural language interfaces (NLIs) that can be found in literature are usually oriented to solve very specific problems in which the knowledge is stored in just one type of source (e.g. NLIs to relational databases, NLIs to ontologies...). However, the reality is usually quite different.

As can be seen in Figure 1, when users interact with an NLI, they have a single objective in mind: getting the information they want at that moment. From their point of view, it does not matter if it is an NLI to a database, an ontology, or an XML file, nor if it is a virtual assistant that can instruct them in using a website or answer general questions. What is really important is that, if we want these systems to be applied to many different fields and be extensively used by the general public and not only by expert users or database administrators, they must put all that heterogeneous information at the service of users in an integrated, transparent, efficient, effective, and pleasant way.

As we will see in Section 3, the architectures used by this type of systems present some advantages and certain disadvantages.

*Corresponding author. C/ Daniel Saucedo Aranda s/n, 18071, Granada, Spain. Tel: +34 958 246163. Fax: +34 958 243317.

Email addresses: eisman@decsai.ugr.es (Eduardo M. Eisman), mariaisabel.navarrojimenez@kaust.edu.sa (María Navarro), castro@decsai.ugr.es (Juan Luis Castro)

¹<https://www.comscore.com/Insights/Press-Releases/2014/5/comScore-Reports-56-1-Billion-in-Q1-2014-Desktop-Based-US-Retail-ECommerce-Spending-Up-12-Percent-vs-Year-Ago>

²<http://www.alibaba.com/>

³<http://www.bbc.com/news/business-29999289>

Figure 1: Natural Language Interface for general public

tages. And although many of these systems have been more or less usable in practice for different application domains, they present several problems. Many of them are not interactive or their interactivity is very limited. However, users often do not know where to find the information, or their requests are vague and they need to complete them little by little with additional information, which could be provided in a dialog. For this reason, the lack of memory and a dialog management module to keep track of the conversation and use the context in an appropriate way is a big problem. Other issues regarding these systems are that most are not ready to work with heterogeneous data sources at the same time and in a transparent way. Moreover, the majority cannot handle fuzzy concepts such as “cheap” or “recent”, or temporal queries involving relative dates, which is essential. On the other hand, they do not use 3D characters that can emotionally engage in conversation with users, making the interaction process friendlier. Finally, sometimes they are simple prototypes that are far from being a powerful and easy to use system that can be used in a real environment with a large number of users of any type.

Our proposal to address these problems and improve what other systems provide is to use a multi-agent approach that allows applying a divide and conquer philosophy. In this way, a set of expert agents specialized in concrete domains facilitates the access to different knowledge sources, and a series of decision agents interacts with them to coordinate them and provide a unique final answer to the user. This proposal will be analyzed in depth in Section 4, and it will include the design and development of a specific conversational system for a bookstore. This system will be used in Section 5 to make a comparative analysis in order to confirm that our proposal provides several improvements in the aforementioned features. Section 6 will present some details about the implementation and usage of the system. Finally, Section 7 will include some general conclusions and directions for future work.

2. Related work

The problem of Natural Language Interfaces (NLIs) started more than four decades ago. NLI systems are divided into three main categories, depending on how knowledge is structured:

- NLIs to structured data: usually closed-domain, they can work either on relational databases translating the query to SQL (the so called Natural Language Interfaces to Databases, or NLIDBs), or on ontologies translating the query to SPARQL, for example.
- NLIs to non-structured or semi-structured data: usually open-domain, they process huge amounts of documents to find the answer to a question.
- interactive NLIs: used in dialog systems, they have memory to remember previous questions.

In this paper we will focus on NLIs to structured data and interactive NLIs. In order to be able to better understand the benefits of our proposal, we will show the main features of some NLIDB systems that can be found in literature, from classic implementations to state-of-the-art prototypes.

Since their appearance at the end of the sixties, many different systems have been proposed. During the first years, all the systems were domain specific. They were tied to a particular database and they were difficult to port. They used to be based on pattern matching techniques and syntactic trees. During the seventies and eighties, this kind of systems were improved with the use of new techniques such as semantic grammars or intermediate representation languages, which were independent from the database. However, it was not until the end of that period that some commercial systems started to appear (Sijtsma and Zweekhorst, 1993) (e.g. Intellect, Natural Language, Q&A, LanguageAccess by IBM, English Query by Microsoft, or English Wizard by Linguistic Technology Corporation). In spite of their initial popularity, these systems gradually disappeared due to several problems: bad performance, significant effort required to develop specialized systems for individual databases that could not be easily adapted to work with different domains, language coverage not obvious to users, ambiguity of natural language, and so on. Nevertheless, after some years in the oblivion, they started to live a second youth with the arrival of the new century, and nowadays they are mature enough as Apple Siri⁴, Google Now⁵, Microsoft Cortana⁶, or Amazon Echo⁷ show, although these systems are much more complex than a simple NLIDB.

We will talk more about the different types of architectures of these systems in Section 3. But first, we will briefly describe some specific examples. Although in Section 5 we will make a thorough analysis considering some of these systems, more details on any of them can be found in the corresponding paper.

LUNAR (Woods et al., 1972) is one of the most well known NLIDB systems. It presented a natural language interface to a database containing chemical analyses of Apollo-11 moon rocks. LUNAR used an Augmented Transition Network (ATN) parser, Woods’ procedural semantics, and two databases for chemical analysis and literature references. Although it was

⁴<https://www.apple.com/ios/siri/>

⁵<https://www.google.com/landing/now/>

⁶<http://www.windowsphone.com/en-US/how-to/wp8/cortana/>

⁷<http://www.amazon.com/oc/echo>

not broadly used, its performance was impressive for that time, with an accuracy of 78%, or 90% if dictionary errors were corrected.

RENDEZVOUS (Codd, 1974) was intended to engage users in free (relatively unrestricted) dialogs to help formulate queries over relational databases. It placed special emphasis on query paraphrasing and engaging users in clarification dialogs. The objective was to make sure that the intended meaning of the user's question had been correctly captured before routing the formal query to the relational DBMS.

LADDER (Hendrix et al., 1978) was designed to access information about US Navy ships. It could be used with large distributed databases and different database management systems. It used semantic grammars (a different grammar had to be defined for each application), parsing the user's question to build a semantic tree, which was then mapped to a database query. It also included facilities such as spelling correction and elliptic reasoning.

PLANES (Programmed LANGUAGE-based Enquiry System) (Waltz, 1978) was able to answer questions using a large relational database of aircraft flight and maintenance data. It carried out clarifying dialogs with the user and was able to answer vague or poorly defined questions. It used an ATN based parser and a semantic case frame analysis to understand questions.

CHAT-80 (Warren and Pereira, 1982) is another of the most popular classical systems. It transformed English questions about world geography into Prolog expressions, which were evaluated against the Prolog database. It used a small set of English language vocabulary, enough for querying the database. It formed the basis of many other systems such as MASQUE/SQL (Androutsopoulos et al., 1993).

ASK (Thompson and Thompson, 1983) allowed users to teach the system new words and concepts during the interaction. It was a complete information management system which interacted with multiple external databases, electronic mail programs and other computer applications. In this way, user's queries generated requests to the appropriate underlying systems.

TEAM (Grosz et al., 1987) focused on the design of portable systems which could be easily configured by database administrators with no knowledge of NLDBs, as opposite to the high configuration costs of other systems. A database expert engaged in an acquisition dialog with TEAM to supply the information needed to adapt the system to a new database or to expand its capabilities in answering questions.

MASQUE/SQL (Androutsopoulos et al., 1993) was a modified version of MASQUE (Modular Answering System for Queries in English) (Auxerre and Inder, 1986), a portable natural language front-end for Prolog databases. It could be used as a front-end to any commercial database supporting SQL, although it had to be configured for each different domain and database.

JUPITER (Zue et al., 2000) was a conversational interface that allowed users to obtain weather forecast information for over 500 cities worldwide over the telephone using spoken dialog. Like many other systems from the Massachusetts Institute of Technology (e.g. MERCURY for flight reservations (Seneff and

Polifroni, 2000) or ORION for off-line task delegation (Seneff et al., 2000)), JUPITER made use of the GALAXY (Goddeau et al., 1994) conversational system architecture. It had a separate geography table organized hierarchically that provided a means of summarizing results that were too lengthy to present fully. An end-to-end Japanese version was developed (MOKU-SEI (Nakano et al., 2001)).

MERCURY (Seneff and Polifroni, 2000) was an over-the-telephone NLI built within the GALAXY-II (Seneff et al., 1998) architecture. It provided flight schedule information and pricing potential itineraries involving the 150 busiest airports worldwide. This domain entailed additional challenges such as time zone differences or geographical proximity of airports. The system included a fuzzy matching heuristic, and users were permitted to specify and update constraints at any point in time.

gNarLI (Shankar and Yung, 2000) was a rule-based pattern matcher written in Perl. Each domain (e.g. movies, or classes at Harvard) had different rules that could be matched to a portion of the input question by a regular expression pattern and had some bearing on the SQL query to be generated. It was not suited to handle huge statistical datasets (e.g. NBA statistics) with many columns; calculate averages and sums, compare, and sort in different ways.

HappyAssistant (and the later enhanced versions Natural Language Shopping Assistant –NLSA– and Natural Language Assistant –NLA–) (Chai et al., 2002) was an interactive NLI that helped users access e-commerce sites to find relevant information about products and services. It used a rule system with weights and ranks to implement business rules that were able to display the web page that satisfied the user's requests or initiate a dialog to ask for additional information. Moreover, when a product was recommended to the user, it generated an explanation automatically. The NLA version included a statistical parsing that allowed scaling to multiple languages and geographies with minimal reconfiguration.

PRECISE (Popescu et al., 2003) introduced the idea of semantically tractable sentences, which could be translated into a unique semantic interpretation by analyzing some lexicons and semantic constraints. It matched keywords in a sentence to the corresponding database elements. First, it narrowed the possibilities using the max-flow algorithm, and then, it analyzed the syntactic structure of the sentence. The evaluation was performed on two databases: ATIS (questions about flights) and GeoQuery (U.S. Geography). PRECISE was able to achieve high accuracy in semantically tractable questions, although it compensated for the gain in accuracy at the cost of recall.

WASP (Word Alignment-based Semantic Parsing) (Wong and Mooney, 2006) was designed to construct a complete, formal, symbolic, meaningful representation of a natural language sentence. It learned to build a semantic parser given a corpus of natural language sentences annotated with their correct formal query languages. The whole learning process was done using statistical machine translation techniques with minimal supervision, so it was not necessary to manually develop a grammar in different domains. WASP was evaluated on the GeoQuery domain and on a variety of natural languages (English, Spanish, Japanese and Turkish).

Let's Go (Raux et al., 2006) was a telephone-based bus schedule information system used in Pittsburgh, USA. It was based on the RavenClaw dialog manager (Bohus and Rudnicky, 2003), which provided a set of domain independent dialog strategies for handling non-understandings. It tried to keep system requests as specific as possible since long prompts were not well received by users.

ORAKEL (Cimiano et al., 2008) focused on minimizing the effort of adapting the system to a given domain. It used an ontology to guide the lexicon construction process and an inference engine to provide an answer, even if it was not explicitly contained in the knowledge base but can be inferred from it. For this reason, questions had to be translated into logical form.

C-Phrase (CatchPhrase) (Minock, 2010) was an NLI system that could be configured using a web-based GUI reducing the necessary time and expertise. It modeled queries in an extended version of Codd's tuple calculus and used synchronous context-free grammars with lambda-expressions to represent semantic grammars. The given grammar rules might be used in the reverse direction to achieve paraphrases of logical queries. The result was automatically mapped to SQL. The system was evaluated on the GeoQuery 250 corpus.

FREyA (Feedback, Refinement and Extended vocabularY Aggregation) (Damljanovic et al., 2012) was an interactive natural language interface for querying ontologies and the Linked Open Data (LOD)⁸. It used the Stanford Parser (Klein and Manning, 2002), an ontology-based lookup, and usability methods such as feedback and clarification dialogs for training the system and improve its performance over time. It was tested on GeoQuery and the DBpedia (Auer et al., 2007).

NaLIR (Li and Jagadish, 2014) was a generic interactive NLI for querying relational databases. It used the Stanford Parser and generated a SQL query that might include aggregation, nesting, and various types of joins. An interactive communicator explained how each query was processed and solved ambiguous interpretations generating multiple choice selection panels. It was applied to different domains: Microsoft Academic Search⁹, Yahoo! Movies¹⁰, and the DBLP bibliography (Ley, 2002)¹¹.

Regarding commercial systems, one of the most famous examples is Anna, IKEA's Automated Online Assistant¹². Like most conversational agents, she has a memory and is able to answer a quite broad set of general questions (e.g. personal information, or hobbies). But, if we focus on the IKEA-related knowledge, although she answers questions about products (e.g. *"I need a black table"*), opening hours, payment methods, or delivery charges successfully, she fails answering a little bit more complex questions like *"I need a cheap black table"*, *"I need a black table under \$ 100"*, or *"What are the best selling black tables of 2015?"*. In both cases, Anna's answer is: *"We have many options for black table in our catalogue. You can*

click on your favorite on the webpage I have opened to you."

An extensive list of other commercial systems applied to different domains and languages can be found on chatbots.org.

In next section, we will analyze how NLIDBs are usually classified regarding their architecture, and we will show the advantages and disadvantages of each one of them.

3. Technologies and architectures

Although there exist other ways of classification, Natural Language Interfaces to Databases systems (NLIDBs) are usually classified according to their architecture (Androutsopoulos et al., 1995; Pazos R. et al., 2013). In this way, we can find four types of systems: pattern matching, syntax-based, semantic grammars, and intermediate representation languages. In this section, we will analyze all these architectures, showing their advantages and disadvantages.

3.1. Pattern matching systems

Pattern matching is the typical architecture used in first NLIDBs. It basically uses two main concepts, patterns and expressions. On the one hand, a pattern is a simple template which can be used to match the user's input in order to identify the type of question. It can be a little bit more complex if, for example, words are reduced to their roots or synonyms are used. On the other hand, an expression is a query, usually written in SQL language, associated to the kind of question and used to get the desired information from the database.

In order to better understand this approach, we will briefly describe two examples of patterns and expressions that can be found in Androutsopoulos et al. (1995). If we consider the pattern "... capital ... <Country>", it could have the associated expression "SELECT capital FROM country WHERE country = <Country>". This would retrieve the capital of a specific country (e.g. *"What's the capital of Spain?"*). On the other hand, the pattern "... capital ... country ..." could have the associated expression "SELECT country, capital FROM country". This second pattern would answer questions like *"List the capital of every country."*

The main advantage of a pattern matching system is its simplicity. It is really easy to implement, add, and remove features from the system, and there is no need for syntactic parsing. In addition, sometimes this kind of systems can provide a reasonable answer when the user asks a question that is out of the range of sentences the patterns were designed to handle. For example, in the face of the question *"Is Barcelona the capital of Spain?"*, the system might not provide a direct *"no"* answer to the question but a list containing the actual capital of Spain, Madrid. This is not the ideal answer, but it is okay.

However, this kind of systems presents a huge disadvantage, its superficiality. When somebody talks with one of these systems, he or she suddenly realizes that many mistakes are made, and the coverage is limited by the number of patterns. For these reasons, they are not enough for making a good natural language interface to databases system.

⁸<http://linkeddata.org/>

⁹<http://academic.research.microsoft.com/>

¹⁰<http://movies.yahoo.com/>

¹¹<http://dblp.uni-trier.de/>

¹²<http://www.ikea.com/us/en/>

$S \rightarrow WQ VP$
 $WQ \rightarrow WH \textit{river} \mid WH \textit{state}$
 $WH \rightarrow \textit{what} \mid \textit{which}$
 $VP \rightarrow V DP$
 $V \rightarrow \textit{borders} \mid \textit{passes through}$
 $DP \rightarrow \textit{Illinois} \mid \textit{Indiana} \mid \textit{Missouri}$
 (a) Syntactic tree

$S \rightarrow \textit{RiverQuestion FlowThrough State}$
 $\textit{RiverQuestion} \rightarrow (\textit{what} \mid \textit{which}) \textit{river}$
 $\textit{FlowThrough} \rightarrow \textit{passes through}$
 $\textit{State} \rightarrow \textit{Illinois} \mid \textit{Indiana} \mid \textit{Missouri}$
 (b) Semantic tree

Figure 2: Trees for question “Which river passes through Illinois?”

An example of pattern matching system is gNarLI (Shankar and Yung, 2000).

3.2. Syntax based systems

A syntax based system analyzes the sentence to make a syntactic (parse) tree which is translated into predicate logic to get the database query. These systems use grammar rules and lexicons. A lexicon is essentially a catalog containing the words of the language, whereas grammar rules combine those words into meaningful sentences in that language.

Taking the GeoBase¹³ database and considering the question “Which river passes through Illinois?” adapted from Pazos R. et al. (2013), using a lexicon and some grammar rules, the system could make the syntactic tree shown in Figure 2a. This parse tree would be mapped to the logical query $\text{?(river}(x) \cap \text{flow_through}(x, \text{Illinois}))$.

The main advantage of this kind of systems is that they provide a detailed structure of the sentence (part-of-speech tagging: verb, noun, adjective, preposition, adverb...), and that syntactic tree can be easily mapped to the database query.

Nevertheless, these systems presents some drawbacks such as semantic ambiguity (a column belonging to several tables), several interpretations (syntactic trees) for a unique query, bad portability (rooted to database), and it is not always clear when a node should add semantic information.

The typical example of syntax based system is LUNAR (Woods et al., 1972), although more recent systems such as FREyA (Damljanovic et al., 2012) and NaLIR (Li and Jagadish, 2014) also make use of syntactic parsing and analysis.

3.3. Semantic grammar systems

Semantic grammar systems are very similar to syntax based systems. In this case, they simplify the parse tree to the minimum removing or combining the nodes that are not needed for making a semantic interpretation of the query. In this way, they better reflect the semantic representation without having complex tree structures.

Since semantic grammar categories are selected to enforce semantic constraints, a grammar or production rule here might not correspond to general syntactic concepts. For example, if we consider the same question as before, “Which river passes through Illinois?”, it could be parsed into the semantic tree that appears in Figure 2b (Pazos R. et al., 2013). As we can see, this semantic tree is shallower and easier to understand and process than the corresponding syntactic tree of Figure 2a.

The main advantage of this kind of systems is its great performance. There is no need for complex syntactic trees. In addition, the semantic information is assigned to tree nodes, so it reduces the elliptical problems of user queries (when a user omits some words of the sentence needed for a correct grammatical reconstruction but not for understanding its meaning).

However, these systems are difficult to port. New semantic grammars are needed for new domains, although some systems try to make these rules automatically from a corpus or interacting with the user.

Some examples of semantic grammar systems are LADDER (Hendrix et al., 1978), PLANES (Waltz, 1978), ASK (Thompson and Thompson, 1983), JUPITER (Zue et al., 2000), MERCURY (Seneff and Polifroni, 2000), NLA (Chai et al., 2002), PRECISE (Popescu et al., 2003), WASP (Wong and Mooney, 2006), and Let’s Go (Raux et al., 2006).

3.4. Intermediate representation languages

Together with semantic grammars, this is one of the most used techniques. Instead of directly translating the user’s query into SQL using a syntax-based approach, they use an intermediate language of logic queries (predicate logic), which is then translated into a specific database query language.

The main advantages are the independence of the database management system (DBMS), and the possibility of including reasoning modules between the semantic analyzer and the database query generator.

Nevertheless, these systems have a limited application scope, since most of them use deductive databases, much less used than relational databases.

Some examples of intermediate representation languages systems are RENDEZVOUS (Codd, 1974), CHAT-80 (Warren and Pereira, 1982), TEAM (Grosz et al., 1987), MASQUE/SQL (Androutopoulos et al., 1993), ORAKEL (Cimiano et al., 2008), and C-Phrase (Minock, 2010).

4. SmartSeller, our proposal

Some authors (Pazos R. et al., 2013) state that a good natural language interface to databases (NLIDB) system should: be easy to configure and use; include tools for modifying the knowledge; make its capabilities and limitations evident to users; offer recommendations sufficiently justified; be robust in case of possible failure; answer quickly and with accuracy; answer deductive, temporal and fuzzy queries; be multimodal; be independent from the domain, the database management system, the language, the hardware and the software; handle linguistic phenomena (e.g. anaphora, ellipsis, ambiguity, or incomplete search values). Moreover, if we focus on the context

¹³<http://www.geobase.ca/>

Figure 3: Generic Multi-Agent Architecture Overview

of that broader sense of NLI described in Section 1 and depicted in Figure 1, we think that not only the aforementioned aspects are necessary, but features like interactivity (including a memory and a dialog manager to engage in conversation with users in order to modify or revise requests), versatility (ability to work with heterogeneous data sources at the same time), or emotions (representation using a 3D character able to emotionally react during the interaction) are also important.

As can be seen in Figure 3, our system has been designed following a client-server multi-agent architecture. Two types of agents have been defined: expert agents and decision agents. Expert agents are responsible for providing information about specific domains, and the access to them is carried out by means of decision agents, which manage the answers provided by the different expert agents to provide a unique coherent final answer to the user. Decision agents could be organized hierarchically to make the integration process easier.

This architecture entails many advantages over other NLIDB systems. Its multi-agent nature makes it easily scalable and allows following a divide and conquer approach. Several expert agents are aimed to different purposes. Each agent is an expert doing its own work, and it does not have to know anything about all the other expert agents. They are able to access, process and abstract the information of heterogeneous data sources and architectures to work with reusable high-level semantic interpretations independent from the database management system.

This generic architecture could be applied to any domain. In this paper we present a specific design for a bookstore. We include two expert agents and one decision agent, as can be seen in Figure 4.

The first expert agent is a Virtual Assistant similar to Elvira,

Figure 4: SmartSeller Specific Architecture Overview

the Virtual Assistant of the University of Granada (Eisman et al., 2012). It is able to answer general questions (e.g. “What’s the time?”, “What’s the weather like?”), personal questions about the Virtual Assistant (e.g. “What’s your name?”, “Do you like singing?”), and specific questions about the shop (e.g. “What’s the opening hours?”, “What’s your telephone number?”, “How could I pay the book?”). For the sake of modularity, we have built a hierarchy of ontologies to store the knowledge associated to these three types of questions. We will talk more about the Virtual Assistant in Section 4.1.

The second expert agent is the Bookseller strictly speaking. It provides specific information about a database containing more than two hundred thousand books (titles, authors, publishers, categories, prices, dates...). It uses a semantic grammar and a set of indexes to improve the performance of the system. We will talk more about the Bookseller in Section 4.2.

On the other hand, the Decision Agent coordinates both expert agents to provide a unique coherent final answer to the user. We will talk more about the Decision Agent in Section 4.3.

Regarding the languages and technologies, the core of the system has been implemented in Java. The knowledge of the Virtual Assistant is stored in ontologies which are edited using Protégé¹⁴ (Noy et al., 2001) and handled by the system using the Apache Jena¹⁵ library. The emotional state of the avatar is controlled using a fuzzy rule-based system (Eisman et al., 2009). The Bookseller manages a set of indexes over the book database using Apache Lucene¹⁶. Finally, the answers provided by the system are transformed into spoken dialog using Nuance Loquendo TTS¹⁷.

¹⁴<http://protege.stanford.edu/>

¹⁵<https://jena.apache.org/>

¹⁶<http://lucene.apache.org/>

¹⁷<http://www.nuance.com/for-business/customer-service-solutions/loquendo-small-business-bundle/index.htm>

Figure 5: Architecture of the Virtual Assistant

4.1. The Virtual Assistant

The first expert agent is the Virtual Assistant, which answers general domain questions (e.g. “What day is it today?”), personal questions about the Virtual Assistant (e.g. “How old are you?”), and general questions about the bookstore (e.g. “Where is the bookstore?”). As can be seen in Figure 5, it is composed of four main modules. The Natural Language Understanding processes the question to generate a list containing all the Information Units (IUs) to which it refers (this concept will be explained in Section 4.1.1). The Dialog Manager uses a rule-based system to filter that list and decide about which IU the Virtual Assistant must talk. The Emotional State Controller updates its emotional state. Finally, the Communication Generator retrieves and adapts the answer for that IU. This answer is made up of several elements: the sentence, the URL of a web page with additional information, a list of recommendations about related topics that can be used to continue the conversation, and so on. Here, we will briefly explain these modules of the Virtual Assistant since they are not the main contribution of this paper. More details on them can be found in Eisman et al. (2012).

4.1.1. Knowledge representation

The Virtual Assistant uses two types of knowledge: several plain text files containing thousands of regular expressions to recognize the questions, and a hierarchy of ontologies to store all the information related to the answers provided.

An ontology (Gruber, 1993; Staab and Studer, 2009) is essentially a formal representation of a set of concepts within a particular domain and some relationship established between those concepts. Our ontology is based on entities that we have called Information Units (IUs), which are pieces of information about specific concepts. They include the definition of the concept, the name, a URL with additional information, and so on. There are different types of IU, which have been explained in Eisman et al. (2012): objects, properties, specific subjects, general subjects, subject lists, and guided tours. All these IUs are not isolated in the ontology, but they can be connected with each other. In this way, we can see our ontology as a kind of graph in which the IUs are the nodes and the connections between them are the edges of the graph.

The other type of knowledge is all the information needed for the recognition of the different questions. This data is stored us-

Bookstore : *compan(ies|y) | books?(|-)?(shop|store)s?*

Address : *address(es)? | addr? | a?venues? | directions? | loca(liza)?tions? | placements? | streets?*

Address : *how * (access | arrive | find | get to | locate)*

Address : *(what | where | which) * (a?venues? | directions? | streets?)*

Telephone : *cellulars? | (smart)?phone | tel(ephone)s?*

Telephone : *(contact | information) numbers?*

Telephone : *numbers?(of)? (contact | information)*

Figure 6: An excerpt from a token file

ing plain text files which contain tokens associated to the IUs. These tokens represent patterns or regular expressions that capture all the different ways that people use to refer to them, and the initial set is specified by hand for each language. Figure 6 shows some very simple examples. Although they are less precise than other types of recognition techniques like semantic grammars, they entail an easy way to capture knowledge.

This could become a very tedious work, but it can be simplified using wild cards (written as ***). This is a greedy operator that can represent any set of words, so the number of possibilities to be written is reduced. However, it must be used carefully to avoid capturing wrong sentences. Another simplifying mechanism is the use of replacements files including regular expressions about certain misspellings that can appear in the question, synonyms, or complex IU names. This reduces the number of words of the question making the processing of the token files faster, increases the reliability of the understanding process since it reduces the ambiguity of questions, and simplifies the maintenance of the system. On the other hand, there are some tools that have been designed to reduce the effort devoted to this task. For example, Moreo et al. (2013) proposed a semiautomatic method to reduce the problem of creating templates to that of validate, and possibly modify, a list of proposed templates. In this way, a better trade-off between reliability—the system is still monitored by an expert—and cost is achieved. In addition, updating templates after domain changes becomes easier, human mistakes are reduced, and portability is increased.

4.1.2. The Natural Language Understander

The Natural Language Understander identifies all the Information Units (IUs) referred by the question. First, it reduces the language to a specific vocabulary so that it can be managed more easily. Some misspellings are corrected, some words are replaced with synonyms, some complex names with abbreviations, and so on. Next, it uses the token files to generate a list with all the IUs referred by the replaced query.

For example, let us consider the question “*What is the address of the bookstore?*”. First, some expressions would be replaced giving as a result the query “*address the bookstore*”. This replaced query would be used to look for the IUs in the

token files. In this case, the object *Bookstore* and the property name *Address* would be identified and passed to the Dialog Manager, which would generate an appropriate answer. In this case, the answer would be the property *Bookstore_Address*.

4.1.3. The Dialog Manager

The Dialog Manager uses the list of Information Units that matches the user’s question to make an abstract answer that is later transformed into a specific answer by the Communication Generator (CG). The process consists of two main steps, IU selection and filtering. Selection is a rule-based process that allows changing the provided answer depending on if it contains a certain IU, if the Virtual Assistant is in a certain emotional state, etc. Filtering removes ambiguity using the contextual information that can be extracted from the ontologies.

The remaining IUs (at least one) are used to generate the abstract answer that is used by the CG to create a specific answer, as will be described in Section 4.1.5.

4.1.4. The Emotional State Controller

The Emotional State Controller manages the emotional state of the Virtual Assistant using a dynamic probabilistic fuzzy rule-based system (Eisman et al., 2009). It uses two essential concepts: emotional state and personality. Emotional state is built on the basis of eight basic emotions that correspond with the following emotional attributes: joy, disdain, anger, fear, worry, surprise, sadness, and embarrassment. The value of each attribute is a real number between 0 (total absence) and 1 (total presence). On the other hand, personality is a set of static features that allows carrying out a subjective assessment of the questions asked by the user, and that makes the emotional state tend to an equilibrium. Six different personalities have been defined: anguished, depressive, hypochondriac, maniac, phobic, and normal.

An example of rule is shown in Figure 7. It states that IF the selected IU for answering the question is *ComplimentGoodLooking*, AND this IU is not already present in the memory of the Virtual Assistant (it is the first time the user makes a compliment), THEN a low positive variation of the embarrassment emotional attribute is produced.

4.1.5. The Communication Generator

The Communication Generator (CG) instantiates the abstract answer provided by the Dialog Manager in a concrete answer written in a specific language. The core of the answer is a sentence associated to the IU that can change depending on a set of constraints like the time or the date. It is also transformed into spoken dialog using the Loquendo text-to-speech. In addition, it can include a web page where the user can find additional information, and a recommendation list with some other IUs related to the answer, which might be very interesting for the user.

4.2. The Bookseller

The Bookseller is a domain specific expert agent that answers questions about a database containing thousands of books. It

AntecedentAnswer[Answer == ComplimentGoodLooking] $\wedge$ AntecedentMemory[ComplimentGoodLooking == Zero]
 $\Rightarrow$ ConsequentEmotionalState[EmbarrassmentVariation = Low_Positive]

Figure 7: Example of fuzzy rule to control the emotional state variation

differs from the Virtual Assistant in many aspects. In general, although a natural language understanding based on tokens is quite simple and works more or less well in some circumstances (even providing reasonable answers for some out-of-domain questions), it does not help the system to really understand the meaning of the question asked by the user, and this superficiality leads it to make many mistakes. However, taking advantage of the structure and the semantics of the question, the system can better handle ambiguity, provide more precise answers, and improve the quality of the recognition. For this reason, as can be seen in Figure 8, the Bookseller employs a dynamic semantic grammar which allows making conditions of different classes, and a set of indexes which makes easier the access to the information of the database. In addition, the grammar generates an intermediate representation language that makes the system independent of the database management system (DBMS). Next, we will describe the whole process:

1. The Bookseller gets a new query from the Decision Agent.
2. The Natural Language Understander (NLU) clones the original base grammar, which does not contain any specific rule about titles, authors, or any other database attribute. These specific rules are added afterwards.
3. The NLU filters the production rules of the cloned grammar removing the final tokens that do not match the question. This speeds up the recognition process.
4. The NLU uses the question to look for titles, authors, categories, publishers, and descriptions in the indexes, and labels in the concepts hierarchy. For the retrieved values, specific rules are generated and dynamically added to the grammar.
5. The NLU carries out a fast pass over the grammar to filter out all the production rules that are not able to generate any part of the question.
6. The NLU processes the grammar to get all the possible restrictions specified by the question.
7. The NLU filters the restriction list using a set of predefined files in order to remove noisy restrictions.
8. The Dialog Manager (DM) processes the restriction list to generate all possible valid interpretations to the question.
9. The DM classifies these interpretations using scores.
10. The DM bets for the best interpretation considering the type of restrictions.
11. The DM determines if compatible restrictions from the memory can be reused.
12. The DM adapts the chosen interpretation to always retrieve at least one record.
13. The Communication Generator (CG) makes a specific answer for the interpretation selected by the Dialog Manager and passes it to the Decision Agent.

Next, we will analyze all these details of the Bookseller.

4.2.1. Knowledge representation

All the information that the Bookseller is going to provide is stored in a database of a real bookstore containing more than two hundred thousand books. Title, author, publisher, category, date, and price are just a few examples of all the information that we can get for each book. A set of indexes makes the access to the database easier, and different operators define how these indexes are created. In addition, a labels hierarchy and a dynamic semantic grammar recognize and capture all the valuable information to provide a precise answer.

The Indexes. In order to facilitate the portability of the system, we access the database through a view that contains all the needed fields. Once defined, the Natural Language Understander module accesses it using a set of indexes handled by the Index Manager. In this way, the accuracy and performance of the system are increased since we can define operators for preprocessing the information obtained through the view. This allows making partial or incomplete searches (e.g. using only few words of the title, or the initials of the author), and it resolves some consistency problems that data might have.

The Dynamic Semantic Grammar. It allows creating a structure of restrictions and interpretations that are ultimately translated into the particular query language of the database. It is modified at runtime to remove the unnecessary grammar rules for a given query and add specific rules using the indexes and labels.

The initial set of grammar rules is specified by hand for each language, and it is fixed and improved with the use of the system. At this moment, our grammar consists of approximately one hundred and fifty semantic rules organized in different categories: restrictive rules that impose restrictions over specific attributes of the database, sorting rules that order the retrieved records by a specific attribute, top rules that get top or bottom records once they have been sorted according to some criterion, and delete rules that remove previously imposed restrictions. Next, we will show some simplified examples of each category.

Restrictive rules. They allow searching by TITLE (exact or partial matching), AUTHOR (exact, partial, or using initials), PUBLISHER, CATEGORY, LABEL, publication DATE (exact year, interval, fuzzy categories such as recent or very recent), PRICE (exact price, interval, cheap, expensive), ISBN, and EAN.

For example, if we asked the question “books written by Shakespeare”, and considering the start symbol $\sim$ CONDITION, the three grammar rules of Figure 9 would be able to produce the fragment “written by shakespeare” and impose the restriction AUTHOR = “shakespeare”. The first rule states that a $\sim$ CONDITION could be formed by an optional $\sim$ BY_AUTHOR (note the question mark after the symbol) followed by a mandatory $\sim$ AUTHOR_VALUE. The second rule states that $\sim$ BY_AUTHOR represents any sentence with the same meaning as “written

Figure 8: Architecture of the Bookseller

by". Finally, $\sim$ AUTHOR_VALUE produces any of the authors contained in the database. Although for the sake of clarity we have only included the author Shakespeare in this third rule, in Section 4.2.2 we will see how this type of rules are dynamically generated depending on the concrete question made by the user.

Another interesting feature is the possibility of associating statements to any production rule. Figure 9 shows three types of statements. The first one is associated to the $\sim$ AUTHOR_VALUE rule and it has the form $\sim$ author = #value#;. It allows defining variables to capture knowledge from users. In this case, it states that the complete value captured or produced by this rule is stored in a variable named author, which is used to generate conditions later on. The second statement is associated to the $\sim$ BY_AUTHOR rule and it is $\%$ CAPTURE_TEXT = false;. This means that the text produced by this rule is irrelevant for the condition that is generated. In contrast, the text produced by $\sim$ AUTHOR_VALUE is meaningful since it is the

name of the author and it should be shown to the user to justify why certain restrictions have been imposed over the data. Finally, the last statement is associated to the $\sim$ CONDITION rule and it is $\text{@smartseller.condition.String(AUTHOR, EQUALS, } \sim\text{author)}$. It means that, if this rule produces some non-empty fragment of the question, then it creates a String condition over the AUTHOR attribute using the value captured by the author variable in the $\sim$ AUTHOR_VALUE rule. Although this only shows the use of a String condition, many other data types have been implemented and can be used: Array, Date, Double, Integer, and so on.

It is important to emphasize that production rules for the start symbol $\sim$ CONDITION do not need to produce the whole question but only fragments of it. This is essential to make a robust system which should do its best from the question although it may not be grammatically perfect. Nevertheless, the more text is produced by the grammar, the better for the recognition


```

~CONDITION → ~BY_AUTHOR? ~AUTHOR_VALUE;
    @smartseller.condition.String(AUTHOR, EQUALS, ~author);
~BY_AUTHOR → (by|of)( the)?( (author|poet|writer)s)? || somebody (named|whose name is) ||
    ((that|which) )?( (ha(s|ve)( been)?|w(as|ere)) )?(ma(de|ke)|writ(e|o|ten))
    ( by)?( the)?( (author|poet|writer)s)? ||
    (the |which )?(author|poet|writer)s?( (are|is|was|were))?;
    @%CAPTURE_TEXT = false;
~AUTHOR_VALUE → shakespear;
    ~author = #value#;

```

Figure 9: Restrictive rule for AUTHOR: "shakespear"

```

~CONDITION → ((in|of) )?(th(e|is) )?(current|
    present) year ||
    ((in|of) )?this year;
    @smartseller.condition.Date(false, DATE,
    GREATER_OR_EQUALS,
    #time_this_year_first_day#,
    LESS_OR_EQUALS,
    #time_this_year_last_day#, null);

```

Figure 10: Restrictive rule for DATE: "this year"

```

~CONDITION → ~LAST ~NUMBER ~MONTH;
    @smartseller.condition.Date(false, DATE,
    GREATER_OR_EQUALS, ~-number, null,
    null, MONTH);
~LAST → last;
~NUMBER → \d+;
    @~number = java.lang.Integer(#value#);
~MONTH → months?;

```

Figure 11: Restrictive rule for DATE: "last months"

process since less ambiguity is generated and more accuracy is gained.

Figure 10 includes another interesting example of grammar rule which shows the use of temporal questions. They allow users to refer to relative dates, so the generated restrictions depend on the time when questions are asked. In this way, the same question could generate different restrictions in different moments. For example, the question "*books published in this year*" would impose a DATE restriction with an interval between the first day and the last day of the current year. This values, specified in the rule by #time_this_year_first_day# and #time_this_year_last_day#, are instantiated at runtime.

Another example of rule covering temporal questions is the one shown in Figure 11. We can see how a ~NUMBER rule stores a number of months in a variable with the same name. This value is later used to create a DateCondition for the last ~number months (note the minus sign before this variable in the ~CONDITION rule). In the same way, additional rules can cover common time expressions such as "*last decade*".

On the other hand, as Figure 12 shows, other restrictive

```

~CONDITION → ((that|which) )?((is|are) )?
    (brand|clearly|extremely|
    frankly|really|super|very)
    (current|modern|new|
    recent(ly)? ) ||
    late(st)? releases?;
    @smartseller.condition.Date(false, DATE,
    GREATER_OR_EQUALS,
    @grammar.date-very-recent, null, null,
    DAY);

```

Figure 12: Restrictive rule for DATE: "very recent"

```

~CONDITION → ~PRICE ~SORTED_BY ||
    ~SORTED_BY ~PRICE;
    @smartseller.condition.OrderBy(PRICE, ASC);
~SORTED_BY → ~BY || ~SORTED ~BY?;
~BY → according to( (its?|the(irs?)) )? ||
    by( (its?|the(irs?)) )?;
~SORTED → order(ed|ing|s)? ||
    sort(ed|ing|s)?;
~PRICE → amounts? || prices? || values?;

```

Figure 13: Sorting rule for PRICE

rules include parameters instantiated at runtime that can be customized by the administrator to modify the behavior of the system. For example, when a book is considered recent, cheap, small (reduced dimensions or number of pages below a threshold), and so on. In this way, if we set the grammar.date-very-recent parameter to -183 days, the question "*brand new computers books*" would retrieve the computers books published within last six months.

Sorting rules. They retrieve records sorted by specific fields such as date or price. The rules shown in Figure 13 state that records are sorted by price in ascending order (ASC) by default. For example, the question "*poetry books sorted by price*" would impose the restrictions CATEGORY = "poetry" and ORDER BY PRICE ASC. In addition to these default rules, others explicitly sort in both ascending and descending order.

```

~CONDITION → ~DETERMINED_ARTICLE? ~MOST_RECENT ~NUMBER ~BOOK? ~PUBLISHED? ||
~DETERMINED_ARTICLE? ~NUMBER ~BOOK? ~MOST_RECENT ~PUBLISHED? ||
~DETERMINED_ARTICLE? ~NUMBER ~MOST_RECENT ~BOOK? ~PUBLISHED?;
@smartseller.condition.Limit(~number);
@smartseller.condition.OrderBy(DATE, DESC);
~DETERMINED_ARTICLE → the;
~MOST_RECENT → la(st|test) || least (ancient|old) || most (current|modern|new|recent)(ly)? ||
newest || posthumous;
~NUMBER → three;
@~number = java.lang.Integer(3);
~BOOK → books? || comics? || editions? || novels? || releases? || stor(ies|y) ||
tales? || texts? || versions? || works?;
@%CAPTURE_TEXT = false;
~PUBLISHED → ((that|which) )?(w(as|ere) )?(edited|published|written);

```

Figure 14: Top rule for DATE

```

~CONDITION → ~ANY_OTHER ~AUTHOR;
@smartseller.condition.Delete(AUTHOR);
~ANY_OTHER → ((by|of|with) )?any( others?)? ||
others?;
~AUTHOR → (author|poet|writer)s?;

```

Figure 15: Delete rule for AUTHOR

QUESTION	super ?heroé?s?
CONCEPTS	avengers batman captain america hulk iron man marvel spider man spiderman superman superwoman thor wolverine x men
WHERE	3 (TITLE & DESCRIPTION)

Table 1: Definition of the “superheroes” label

Top rules. They retrieve top or bottom records once they have been sorted by a certain criterion. For example, for the question “the three most recent books written by Shakespeare”, the rules shown in Figure 14 would impose the restrictions AUTHOR = “shakespeare”, ORDER BY DATE DESC and LIMIT 3.

Delete rules. They remove previously imposed restrictions. For example, using the rule shown in Figure 15, the question “any other author” would delete any restriction imposed by some restrictive rule over the AUTHOR attribute.

Labels. They are features that we associate to a database record without modifying it. They can be defined manually or automatically using a thesaurus. This enriches the data and fills the knowledge gaps that might exist in the database. In this way, labels are a great value since they allow users to make queries containing information that is not even present in the database.

Labels are defined using a hierarchy of concepts. In the future this will allow the Bookseller to recommend related books and give advice when a user is not sure about what item to buy.

A perfect example to illustrate the use of labels is the query “superheroes”. As can be seen in Table 1, each label is determined by three parameters: in the presence of which questions we want the label to fire, which concepts we want to associate to the label, and a mask indicating where the Bookseller should look for those concepts (e.g. 1 for searching through the title, 2 for the description, 3 for both the title and the description, 4 to associate a concrete book using its identifier...).

So, even though the database does not contain any information about when some character is a superhero, the Bookseller

knows that Batman and Spider-man are superheroes, and any book including these words in its title or description would be a good candidate to be retrieved if the user asked “superheroes”.

4.2.2. The Natural Language Understander

The Natural Language Understander identifies the restrictions that are specified by the question. The whole process consists of six main steps: grammar cloning, rule filtering, searching, grammar pruning, processing, and restriction filtering.

Cloning. The original grammar is copied to be modified.

Rule filtering. Final tokens not matching the query are deleted. For example, all the right parts of the ~ANY_OTHER and ~AUTHOR rules shown in Figure 15 would be deleted for the question “To kill a mockingbird” since this would not match the words *any*, *other*, *author*, *poet*, and *writer*.

Searching. The question is searched using the indexes and labels, and each result is preprocessed to get different combinations which generate new grammar rules. This preprocessing is different for each attribute. For example, Figure 16 shows the rules that would be made for the author William Shakespeare permuting his names and initials. Although there could be many combinations, in practice only those appearing in the question are added. What is important is to be flexible enough.

For the title attribute, the preprocessing is a little bit more complex. First, it tries generating combinations for an (almost) exact matching relaxing stopwords (using a predefined list) and

```

~AUTHOR_VALUE → william shakespeare;
~AUTHOR_VALUE → shakespeare william;
~AUTHOR_VALUE → william s;
~AUTHOR_VALUE → s william;
~AUTHOR_VALUE → w shakespeare;
~AUTHOR_VALUE → shakespeare w;
~AUTHOR_VALUE → william;
~AUTHOR_VALUE → shakespeare;

```

Figure 16: Dynamic grammar rules for AUTHOR: "William Shakespeare"

even omitting probably optional content. For example, it would remove the content in brackets for a book titled "*Around the World in 80 Days (Illustrated Edition)*". If this matching is not successful, it tries to the other side, that is, instead of searching the title returned by the search result through the question asked by the user, it tries an exact matching of the question over the title. This is useful to resolve questions like "*Snow White*" or "*The Seven Dwarfs*", which would be an exact matching over a book titled "*Snow White and the Seven Dwarfs*". Again, stop-words are relaxed for better results. Even so, if any kind of exact matching is not successful, partial matching is tried instead. In a first step, it starts omitting few words in the title. In a second step, it tries matching low frequency isolated words. Figure 17 shows some examples of rules that would be generated for the title "*Around the World in 80 Days (Illustrated Edition)*".

Again, only those rules that produce some fragment of the question are generated. In addition, we could use a stemmer to reduce words to their roots or a thesaurus to include synonyms. This would increase the overall performance and accuracy.

Pruning. Once specific rules have been added to the grammar dynamically, this is pruned to remove those production rules which we know for sure that are not going to match any fragment of the question. This speeds up the grammar processing. For example, if no ~AUTHOR_VALUE rules were generated, the ~CONDITION rule shown in Figure 9 would be pruned.

Processing. All the remaining production rules with a left part equals the start symbol of the grammar ~CONDITION are processed to extract all possible restrictions from the question.

Restriction filtering. Any noisy restriction that might generate undesirable ambiguity is removed. For example, if a title restriction were formed by only one very common word (e.g. "it"). Nevertheless, it would be kept if it were formed by more words and the meaning were perfectly clear (e.g. "*the book It*").

4.2.3. The Dialog Manager

The Dialog Manager combines restrictions and generates candidate interpretations that are evaluated to select the best answer. The process consists of five steps: candidate generation, scoring, filtering, memory management and adaptation.

In this section, we will consider the question "*I am looking for a version of Mr. and Mrs. Smith that costs less than ten euros*", and we will see how it is handled

throughout the whole process. We will begin with the assumption that the Natural Language Understander has identified several restrictions for the question: VOID = "I am looking for a version of", TITLE = "Mr. and Mrs. Smith", AUTHOR = "Smith", and PRICE < 10 euros (justified by "*costs less than ten euros*"), where VOID is a special variable that captures specific text fragments (e.g. common expressions, requests for something...) that might generate undesirable ambiguity.

Candidate generation. Restrictions are combined to generate all possible maximal interpretations to the question. At this step, the Dialog Manager does not care about whether the generated interpretations retrieve any result from the database, it only makes sure that the restrictions are compatible when they make up a new interpretation. Two restrictions are incompatible if they are produced by two fragments of the question that overlap with each other, or if the fulfillment of one implies the non-fulfillment of the other, and vice versa.

Considering the aforementioned question, since the TITLE and AUTHOR restrictions are not compatible with each other (that author did not write that book), just two maximal interpretations would be generated: VOID & TITLE & PRICE, and VOID & AUTHOR & PRICE.

Scoring. A score is calculated for each maximal interpretation by adding the weights of the words of the question that it cannot produce. The higher this score is, the more penalized the interpretation will be. Three word lists with different weights are used: a generic stopword list (0.2), a domain word list (0.5), and a high frequency word list (0.5). Any other word falling out of these lists has a penalization of 1.0.

In our example, the score of the VOID & TITLE & PRICE interpretation would be equal to 0.2, the weight of "*that*", and the one of VOID & AUTHOR & PRICE would be 1.4, the sum of the weights of the words "*Mr. and Mrs.*" (0.5, 0.2 and 0.5) and "*that*".

Filtering. To choose the best interpretation, these are subsequently sorted by four factors: ascending score, whether it retrieves some results from the database (binary classification), ascending number of restrictions composing it, and type of restrictions (this allows giving more priority to categories over titles, for example). Once sorted, the first complete or sufficient interpretation is chosen. An interpretation is said to be complete if more or less it recognizes the whole question, that is, its score is below a maximum threshold (e.g. 1.0). On the other hand, it is sufficient if, even though it does not recognize the whole question, its restrictions allow considering it good enough. For example, if it includes an AUTHOR or TITLE restriction and it has a minimum length (e.g. 50% of the question). If no interpretation fulfills these requirements, the question is not recognized by the Dialog Manager.

Following these rules, the best interpretation would be the one containing the TITLE restriction. In addition, it would be complete since just one word of the whole question (the "*that*" stopword) would not be produced by that interpretation.

`~TITLE_VALUE → around ~STOPWORD world ~STOPWORD 80 days illustrated edition;`
`~TITLE_VALUE → around ~STOPWORD world ~STOPWORD 80 days;`
`~TITLE_VALUE → around ~STOPWORD world;`
`~TITLE_VALUE → world ~STOPWORD 80 days;`
`~TITLE_VALUE → 80 days;`
`~STOPWORD → about || above || after || again || against || all || and || any || as || at || ...;`

Figure 17: Dynamic grammar rules for TITLE: "Around the World in 80 Days (Illustrated Edition)"

Although at this moment we have just focused on the best interpretation found, the system could be improved if for example the top 3 interpretations were combined when the best one has a pretty high score (i.e. matches only loosely the sentence). However, to prevent users from being confused, the results provided by those interpretations should not be merged but shown independently, as Amazon or Google do when a query does not return any results.

Memory management. It is determined whether the selected interpretation is a new query, or whether it refines the previous one and compatible restrictions from memory could be reused. Using a rule-based system, an interpretation is considered a new query if it fulfills one of the following conditions:

- it contains a TITLE, ISBN, or EAN restriction.
- it contains an AUTHOR restriction and the previous interpretation already contained an AUTHOR, TITLE, ISBN, or EAN restriction.
- it contains a LABEL restriction and the previous interpretation already contained a LABEL, CATEGORY, TITLE, ISBN, or EAN restriction.
- it contains a CATEGORY restriction and the previous interpretation contained a TITLE, ISBN, or EAN restriction.
- it contains a CATEGORY restriction and the previous interpretation contained a CATEGORY and AUTHOR restriction.

On the other hand, if the new interpretation is a refinement of the previous one, the Dialog Manager tries to reuse all previous restrictions that are not incompatible with any created by the new interpretation. In this way, the user could make a general query and successively refine it if too many results are returned.

Regarding our example, the chosen interpretation would be considered a new query since it contains a TITLE restriction.

Adaptation. If the interpretation does not retrieve any record from the database, the Dialog Manager starts to relax restrictions. We prefer to return similar results rather than just answer that no items matched the query. In any case, the user would be informed about which restrictions could not be satisfied.

Having arrived at this point, two things might happen in our example. If the interpretation returned some results, it would be kept as it was. However, if all the versions of the searched book were more expensive, the PRICE < 10 restriction would be transformed into an ORDER BY PRICE ASC restriction.

4.2.4. The Communication Generator

The Communication Generator translates the interpretation selected by the Dialog Manager into a concrete answer that the user can understand. This is made up of a summarized sentence, a sound file, a detailed sentence, and some search results.

Summarized sentence. Sentence for the avatar to read that briefly describes the restrictions identified in the last question, and if any of them could not be fulfilled and had to be relaxed.

For instance, if a user asked the question “*The Divine Comedy by Dante*”, the Bookseller would make the interpretation TITLE & AUTHOR, being the summarized sentence “*Here are the books with the title and author you provided.*”. Next, if many results were returned, the user could refine her search with a second question “*a recent version published by Penguin*”. The system would combine the new DATE and PUBLISHER restrictions with the ones from memory, making the interpretation TITLE & AUTHOR & DATE & PUBLISHER. If none of the books satisfied the DATE restriction, the Bookseller would adapt the interpretation to TITLE & AUTHOR & PUBLISHER & ORDER BY DATE DESC, being the summarized sentence in this case “*I haven’t found any book with that date restriction, but I show you the ones having that publisher.*”.

Voice. A phrase dynamically generated by the text-to-speech Loquendo from the summarized sentence.

Detailed sentence. It shows exactly all the restrictions that are being applied, either because they have been recognized from the last question, or because they have been reused from memory.

If we consider the same two questions as before, in the first case the detailed sentence would be “*These are the 21 results that I have found with the title ‘The Divine Comedy’ and the author ‘Dante’.*”. On the other hand, the second sentence would contain all the restrictions being applied at that moment, “*These are the 3 results that I have found with the title ‘The Divine Comedy’, the author ‘Dante’ and the publisher ‘Penguin’.*”.

Search results. Finally, the answer contains the set of books matching the interpretation of the question and which might be sorted by a certain criterion.

4.3. The Decision Agent

The Decision Agent is the joint point between the user and the expert agents. Although the possibility of reusing the information provided by different expert agents to make a more

complete final answer is very attractive, choosing the response of one expert agent is enough most of the times. For this reason, in this first version of SmartSeller we have implemented the following approach:

1. The user makes a query which is received by the Decision Agent. This agent sends the question to the Virtual Assistant to see if this expert agent is able to recognize it.
2. If the Virtual Assistant recognizes the whole question, the Decision Agent does not send it to the Bookseller because it already knows the right answer.
3. If the Virtual Assistant does not recognize the whole question, the Decision Agent sends it to the Bookseller.
4. If the Decision Agent just receives a response from one expert agent, that answer is sent to the user straight away.
5. If both expert agents provide an answer, these are compared by means of a scoring measure like the one used by interpretations, and the best answer is given to the user.

The fact of not sending a question to the Bookseller when the Virtual Assistant is able to perfectly recognize it increases the system performance. However, this optimization affects the way some ambiguous situations are handled because it gives a higher priority to the Virtual Assistant expert agent.

On the other hand, although it has not been considered in the particular case studied in this paper, the general architecture presented in Figure 3 allows including meta decision agents to choose (or combine) the other agents' individual decisions based on their set of experts. We have not experimented this possibility yet, but we think it will be really useful as more expert agents are incorporated into the system.

4.4. The User Interface

As can be seen in Figure 18, SmartSeller is placed over the target website in a different layer. The avatar is a 3D model which is able to speak and emotionally react to the questions asked by users. It is similar to the one included in Elvira, the virtual assistant of the University of Granada¹⁸. As Figure 19 shows, the model is created from a single photography using the technology of Motion Portrait¹⁹. We have made an additional layer on top which defines emotional attributes and provides emotion management. On the other hand, its answer panel contains all the conversation and the recommendations of the system to continue the dialog. Finally, in addition to generating dynamic pages with information about the books that match the query as it happens in Figure 18, the system can redirect to certain webpages to support its answers.

5. Analysis

In Section 4 we identified a list with some important features that a good natural language interface (NLI) system should have, considering the broader sense of NLI that was described

in Section 1 and depicted in Figure 1. The main objective of this paper was to provide a series of improvements in those key factors. Therefore, those features will be used here to make a comparative analysis between our SmartSeller system and some representative alternatives that were described in Section 2. The results of this study have been summarized in Table 2, and they will be analyzed along the following paragraphs. All this information has been purely gathered from the papers that describe those systems, with the exception of Anna, a commercial system that has been tested online.

2D/3D character. When the system is open and it is aimed to be used by the general public, the use of a virtual character may generate more confidence on users and influence their perception of the system. The fact of considering a 3D character instead of a 2D one might contribute to this sensation, although sometimes the difference in quality is not very significant.

Many of the systems included in Table 2 lacked a virtual character. JUPITER and MERCURY were two over-the-telephone natural language interfaces, so this lack was justified. Of the other systems, only Anna was represented by a 2D interactive character which allowed engaging with users in conversation in a friendlier way. She could tilt her head, blink, and smile. All the other systems were not represented by any character and the communication was just performed by means of text sentences.

SmartSeller uses the 3D model described in Section 4.4 to achieve a much more realistic user experience. It is perfectly integrated with the model of emotions described in Section 4.1.4. It can move its head and its eyes, blink, nod, deny, or blush. The opening of the mouth is synchronized with its voice. Moreover, many different accessories can be included (e.g. glasses, hats, or wigs).

Built-in Automatic Speech Recognition (ASR). Nowadays, there are more and more portable devices where traditional input interfaces are limited and not very effective. In addition, accessibility is more and more important. In both cases, automatic speech recognition (ASR) becomes really useful.

Many of the systems included in Table 2, like the vast majority of the systems described in Section 2, did not include a built-in ASR module. Maybe, the low reliability of ASR technology during many years has contributed to this situation. There are some exceptions such as JUPITER or MERCURY, which provided information over the telephone using spoken dialog, so they included their own speech recognition system. In the case of NLA, it supported mixed-initiative dialog with multiple modalities, including typed-in text and speech. Nowadays, it is not very normal that this kind of systems develop their own ASR technology since a great effort is required to achieve as good results as a commercial software.

SmartSeller is in that group of systems that do not include a built-in ASR module. Nevertheless, ASR technology has been significantly improved over the last few years, and nowadays is present in almost all mobile devices, and the results are really impressive. For this reason, the lack of a built-in ASR module is not so important since that functionality could be borrowed from the operating system of the device.

¹⁸<http://tueris.ugr.es/elvira/?lang=en>

¹⁹<http://www.motionportrait.com/>

Figure 18: The user interface of SmartSeller for a real Spanish bookstore

Figure 19: Components of the 3D avatar

Built-in Text-to-Speech (TTS). As happens with automatic speech recognition (ASR), accessibility and new portable devices make text-to-speech (TTS) become really useful.

Traditionally, TTS technology has achieved better results than ASR since the former is a generation process and the latter is a recognition process. However, this has not been clearly reflected in the reviewed systems, maybe because TTS is viewed as an accessory feature and in the past the quality of the generated voices (unemotional and canned) was not very good. From the systems of Table 2, just JUPITER and MERCURY included a commercial-off-the-shelf solution (DECTalk), since they were over-the-telephone conversational interfaces.

SmartSeller includes its own TTS server with a local cache

built on top of the Nuance Loquendo technology²⁰. This speeds up the answer generation process. Nevertheless, most modern operating systems include a TTS engine. Therefore, in most of the cases we should not pay too much attention to this feature when evaluating the quality of a system.

Database management system independence. When developing a generic natural language interface, the independence from the underlying database management system (DBMS) is very important. This allows the system to be used to extract information from any kind of database.

²⁰<http://www.nuance.com/for-business/customer-service-solutions/loquendo-small-business-bundle/index.htm>

	JUPITER	MERCURY	NLA	PRECISE	ORAKEL	NaLIR	Anna	SmartSeller
2D/3D character	X	X	X	X	X	X	✓ (2D)	✓ (3D)
Built-in ASR	✓	✓	✓	X	X	X	X	X
Built-in TTS	✓	✓	X	X	X	X	X	✓
DBMS independence	✓	✓	✓	✓	✓	✓	?	✓
Dialog management	✓	✓	✓	X	X	X	✓	✓
Ease of portability	✓	✓	✓	✓	✓	✓	?	✓
Emotions	X	X	X	X	X	X	✓	✓
Fuzzy queries	X	✓	✓	X	✓	X	X	✓
General domain queries	X	X	X	X	X	X	✓	✓
Heterogeneous data sources	X	X	X	X	X	X	✓	✓
Linguistic phenomena	✓	✓	✓	✓	X	✓	✓	✓
Memory management	✓	✓	✓	X	X	X	✓	✓
Multilingual	✓	X	X	X	X	X	✓	✓
Temporal queries	✓	✓	X	X	?	X	X	✓

Legend: JUPITER (Zue et al., 2000); MERCURY (Seneff and Polifroni, 2000);

NLA (Natural Language Assistant) (Chai et al., 2002); PRECISE (Popescu et al., 2003); ORAKEL (Cimiano et al., 2008);

NaLIR (Li and Jagadish, 2014); Anna [<http://www.ikea.com/us/en/>]

Table 2: Comparative analysis between SmartSeller and other representative systems

Although in some cases it is not possible to determine the degree of independence, in general most of the systems try to avoid being tied to a specific DBMS. JUPITER obtained forecasts from a relational database using SQL, but this database was completed from multiple websites, so the system had to recognize when two sources provided overlapping information. NLA also maintained a local database that was populated directly from the original databases. This automatic process converted data types and extracted product specifications from multiple attributes on a daily basis. NLA generated constraints that were later translated into SQL statements. ORAKEL was one of the most interesting systems regarding this aspect. It translated questions into logical form, which allowed using an inference engine to provide an answer, even if it was not explicitly contained in the knowledge base but could be inferred from it. Its architecture allowed porting the system to any query language, in particular some RDF query languages or plain SQL to access conventional relational databases. NaLIR also focused on relational databases, although another system from the same author named NaLIX (Li et al., 2005) was able to work with XML files. In the case of Anna, we could not confirm the independence from the underlying DBMS, but the fact of being a commercial system made by an independent company makes us think so.

SmartSeller translates queries into an intermediate representation language using the dynamic semantic grammar described in Section 4.2.1. This constitutes an abstract layer that makes restrictions and interpretations independent from the DBMS. In addition, a set of indexes is used to make the system independent from the particular structure of the database of products. Moreover, SmartSeller enriches the database providing an additional classification by means of a hierarchy of labels.

Dialog management. When a user interacts with an NLI, sometimes she does not have a clear objective in mind. She can

specify an initial set of restrictions which may change afterwards. For this reason, it is important to include a dialog manager which can engage in conversation with users, warn them if a certain restriction cannot be fulfilled, and so on.

PRECISE, ORAKEL and NaLIR were traditional NLDBs, that is, they did not include conversational capabilities. In the case of NaLIR, it was interactive, but just in the sense of including menus that allowed changing some parameters of the recognized question. JUPITER and MERCURY were two over-the-telephone conversational systems, so the amount of information that they could provide in a turn had to be very concise. The domain of JUPITER, weather forecast information, did not require extensive dialog management. It was mainly basic information access. However, MERCURY required dozens of turns to accomplish a typical task of making a round-trip flight reservation. MERCURY used a mixed-initiative approach and a confirmation strategy when “dangerous” actions were requested to the user (e.g. to avoid an extensive repair process to recover when city name requests were misrecognized and the source city was changed during the itinerary planning stage). Confirmation could become wordy and inefficient, so it had to be used carefully. NLA also proposed a mixed-initiative interaction. Although it gave the initiative to users, the dialog manager asked them specific questions to narrow the recommendation list. It differentiated between two types of users, technology savvy and general, to ask about the specific product attributes or usage patterns that best discriminated among the retrieved products. It also utilized several strategies to deal with different situations: clarification screens with possible queries when no constraints were identified from a user input, valid attribute ranges when the user specified invalid constraints, and so on. A similar strategy was carried out by Anna. For example, at the beginning of the conversation, she encouraged us to tell her our nearest store, or if we looked for a specific product in IKEA and then we specified a color in which the product did not ex-

ist, Anna kindly offered us to do a new search with a different color.

SmartSeller is also a mixed-initiative system. The feedback that it always provides to the user and the inclusion of delete rules to clear restrictions that we saw in Section 4.2.1 avoid having to interrupt the conversation with annoying clarification dialogs. Nevertheless, what makes SmartSeller different from all the systems of Table 2 is that each agent has its own dialog manager. This is an advantage because it allows each agent to adapt itself to the particular features of each problem. For example, the Virtual Assistant agent uses a rule-based system to set a group of priorities and filter the list of recognized Information Units to decide about which one it is going to talk. On the other hand, the Bookseller agent combines in an intelligent way the different restrictions that have been identified to remove collisions and ambiguity. Thus, it generates all valid interpretations to the question in order to choose the best one according to a scoring measure. Finally, the Decision Agent has a general overview of the system and can perform a unified strategy.

Ease of portability. It is the ability to configure or adapt the system to different domains without a significant effort.

During the first years of natural language interfaces, it is true that this posed a serious problem. However, in general, nowadays there is not a really big difference between most of the systems regarding this aspect. Maybe those that perform a statistical parsing or gather some of the domain knowledge automatically have some advantage over the others. In JUPITER, a scripting language controlled the flow through each dialog so that the system could be specialized to a variety of different configurations. NLA allowed scaling to multiple languages and geographies with minimal reconfiguration and its authors developed various tools and processes for the maintenance of the business rules and the statistical parser of the system. Concretely, they maintained a local database to avoid problems with the structure of external databases. PRECISE was able to automatically build the lexicon from the information of the database and WordNet²¹ (Miller, 1995), and it was tested on three distinct databases in the domains of restaurants, jobs, and geography. ORAKEL focused on minimizing the effort of adapting the system to a given domain and allowed users which were not familiar with methods from natural language processing or formal linguistics to port the system to a certain domain and knowledge base. NaLIR was used against a number of real application scenarios including Microsoft Academic Search²², Yahoo! Movies²³, and DBLP collection²⁴. In any case, if we want to achieve very good results and a great performance, it will always be necessary to adjust or tune the system in order to take account of the distinctive features of each application domain.

In the case of SmartSeller, the fact of being a multi-agent system gives it a certain advantage because we can reuse the different expert and decision agents between applications. In addition, all grammar rules are specified independently, so they can

be reused between different domains. Moreover, the possibility of editing the interpretation of fuzzy concepts explained in Section 4.2.1 gives each administrator the opportunity of adapting the system to his own necessities. Finally, the use of indexes makes the system independent from the particular structure of the database.

Emotions. One of the problems of conversational systems is their difficulty to emotionally engage in dialog with users. They usually seem cold and with no feelings, and this affects their credibility. Their answers should entail an emotional reaction, which should be reflected both at a textual level of the answer itself and the visual behavior of the agent.

Finding systems that are able to emotionally react during the interaction is not so common. Some of them reflect certain personality or character when they are asked some personal questions. However, it is very superficial because the reaction is only a mere textual output and it does not have an underlying emotional model that could really affect the state of the agent. From the systems of Table 2, just Anna was able to show a certain emotional reaction. She smiled with the question “*gift cards*”. However, in general her emotional behavior was quite limited and it did not seem to integrate a full emotional model.

If we compare it to the reviewed systems, SmartSeller is, by far, the most complete, natural and realistic system when we talk about emotions. As we saw in Section 4.1.4, it includes an emotional state controller based on a dynamic probabilistic fuzzy rules based system. It defines eight types of emotions and six different personalities, and they are visually reflected by means of a 3D model. Emotions can completely change answers, up to the point that they could make the system deny answering any other question until the user apologized if he has had a lack of respect to it. On the other hand, personalities modify the influence that events have on the system.

Fuzzy queries. People do not always use a perfectly accurate language to express themselves, but they often do it in an imprecise or incomplete way. Although this is something natural for humans, it entails an extra level of complexity in human-computer interaction by means of natural language. For this reason, it is really important for the system to be able to recognize imprecise queries that contain fuzzy terms.

MERCURY used a fuzzy matching heuristic to understand users when they employed inexact terms to refer to departure/arrival times, airline names, flight numbers, and source/destination/connection cities. For example, a flight departing at 7:56 AM might be referred to as “*the eight o'clock flight*”. NLA mapped qualitative constraints (low or high constraints on numeric product attributes) to specific constraints based on automatic partitioning of the current range of values. For example, considering high the top one-fifth of the capacities of that time, “HDD size: high” was mapped to “HDD size > 20 GB”. NLA also allowed queries containing terms such as “*cutting edge*”. ORAKEL expected a definition of the semantics of an adjective in terms of a rule, and it could only handle scalar adjectives such as “*big*”, “*high*”, “*long*”, etc. All the other systems did not handle fuzzy queries. For example,

²¹<http://wordnet.princeton.edu/>

²²<http://academic.research.microsoft.com/>

²³<http://movies.yahoo.com>

²⁴<http://dblp.uni-trier.de>

although NaLIR could handle complex queries containing aggregations, comparisons, and various types of joins and nesting, it was not designed to handle imprecise terms. The same happened with Anna, who could not correctly recognize questions containing fuzzy terms such as “cheap” or “small”.

SmartSeller not only recognizes fuzzy queries, but it also allows configuring how these queries are interpreted, as we saw in Section 4.2.1. For example, the administrator of the system can modify the price range for what is considered cheap, or the date range that makes something be recent, since these values can be significantly different for two bookstores that sell different kinds of books.

General domain queries. It has always been interesting for any conversational system to include enough knowledge to answer general purpose questions about the personal information of the avatar, its hobbies, and so on. This is something that appeals to users and entertains them, and it is able to arouse their curiosity to see how far the system could get. For this reason, a large number of questions are always related to this matter. However, if we focus excessively on this type of questions, we run the risk of making users think that the system has been designed for that purpose instead of being a natural language interface. In addition, the fact of giving a bad answer to any of that out-of-domain questions might negatively influence the perception that users have of the system.

Most of the systems included in Table 2 were traditional NLDBs that focused exclusively on the access to the information stored in the database, and they completely forgot about general domain questions. Maybe it is not so important to answer those questions properly, or at least how the user would expect, but the fact of being able to recognize those questions and use the answers to reorient the conversation towards the objective for which the system has been designed (e.g. product selling). This is what some systems like Anna do. For example, she tried to relate intimate questions with specific products of IKEA.

SmartSeller shares this idea of reorienting the conversation, but it also contributes a different point of view in the sense that it uses several expert agents that allow making use of different data sources. For example, the Virtual Assistant uses a hierarchy of ontologies that not only indexes the website to answer frequently asked questions about the shop (e.g. “opening hours”, “payment methods”, or “delivery charges”) but it also allows answering questions about the weather, the time, or the personal information of the assistant. In addition, we could include new expert agents to extend that functionality. For instance, an agent to query Wikipedia or DBpedia (Auer et al., 2007) would suppose a huge source of knowledge.

Heterogeneous data sources. Information is not stored in just one place but it is scattered across different sources. There are innumerable services on the Internet that offer very specific information and of great quality. Making one of those systems from scratch could be really expensive. For this reason, services like Apple’s Siri ²⁵ use third-parties to offer certain con-

tents (e.g. Wolfram Alpha ²⁶ for mathematical questions, or Yelp! ²⁷ for relevant business near you). This confirms the importance of integrating heterogeneous data sources that could be accessed by the system at the same time. Sometimes the information will be stored in a certain location (e.g. a product database), and others in another different place (e.g. a website). However, this must be completely transparent to users.

As we can see on Table 2, most of the systems were not designed to work with different types of data sources at the same time. JUPITER just handled one type of content, weather forecast information that was integrated into a local database, although it is true that that information was gathered from multiple websites and this entailed additional problems. In MERCURY, the information on flight schedules and pricing was obtained in real-time from the Sabre Group’s Travelocity service, although it was complemented with a small local database of pragmatic information, such as geographic location of airports. ORAKEL was DBMS independent, but it was not designed to work with heterogeneous data sources simultaneously. NaLIR was an interactive NLI for querying relational databases, and although a similar system from the same author, NaLIX (Li et al., 2005), could query XML files, they were independent systems that did not communicate themselves to resolve a same problem collaboratively. There seemed to be just one exception, Anna, who could not only answer questions about the product database, but also index (probably by hand) certain sections of the IKEA’s website (e.g. “return policy”). However, this ability was rather limited and it was not as versatile as the one proposed by SmartSeller.

The architecture and multi-agent nature of SmartSeller suppose a great advance since they allow the user to access the information independently of being stored in different locations or using diverse formats. As we saw in Figure 1, SmartSeller could interact not only with databases or websites as it happens with most of the systems included in Table 2, but also any kind of data source like ontologies, social networks, XML or plain text files, forums, APIs, and so on. Of course, provided that an expert agent has been made to handle that type of knowledge, in the same way as an operating system needs different drivers to communicate with distinct devices.

Linguistic phenomena. Making a good conversational system is rather complex due to the richness of natural language. There are many linguistic phenomena that could appear: anaphora (the use of an expression the interpretation of which depends upon another expression in context), ellipsis (the omission of one or more words that are nevertheless understood in the context of the remaining elements), ambiguity, incomplete search values, and so on. Handling as most of these phenomena as possible is very important to perform a good recognition.

JUPITER, like most of the systems, allowed unimportant words to be skipped. MERCURY developed a fuzzy matching heuristic that accounted for departure and arrival times, airline names,

²⁵<https://www.apple.com/ios/siri/>

²⁶<http://www.wolframalpha.com/>

²⁷<http://www.yelp.com/>

flight numbers, and cities. It also handled ambiguous intentions to book a flight. For example, it was unclear whether “*the later flight*” meant that the user wanted more information or she wanted an actual booking. In that case, MERCURY adopted the general strategy of treating those potential booking requests as more information requests and followed up with “*Shall I book this flight for you?*”. NLA handled incomplete search values matching the expression in a particular attribute category with values of that attribute in the product database and choosing the most similar one (e.g. the constraint OS = “*win xp*” was matched to OS = “*Windows XP Professional*”). PRECISE was able to handle free-standing values, where the attribute was implicit (e.g. in “*What French restaurants are located downtown?*”, the word “*French*” referred to the value French of the implicit database attribute Cuisine). For string matching, it saved words and their WordNet synonyms in a hash table so that it could convert each word into the original value in the database. In case of ambiguity, PRECISE simply asked the user to choose between the distinct semantic interpretations. ORAKEL assumed a full parse of the input sentence and thus expected the sentence to be grammatical. However, if the question was not grammatical it simply failed and told the user that it did not understand the question. This behavior is unacceptable since handling ungrammatical input and unknown words is a must for any ‘commercial’ natural language interface. NaLIR extended the generated parse tree to avoid that some words failed in mapping to database elements due to the vocabulary restriction of the system. Regarding ambiguous interpretations, it always showed multiple options for the user to choose from. This way of handling ambiguous interpretations was also used by Anna. She let you choose from a set of different options (e.g. product types, product names, colors, or price ranges).

Regarding SmartSeller, it employs several approaches to handle these problems. On the one hand, the Virtual Assistant uses a token-based recognition algorithm that allows extracting information from questions even though the system is used like a search engine and many words are omitted. The ambiguity in this case is resolved by means of a rule-based system which assigns different priorities to the recognized information units. In addition, it can use the guided tours that were mentioned in Section 4.1.1 to implement decision trees. On the other hand, as we presented in Section 4.2.1, the Bookseller uses a dynamic semantic grammar which allows detailing the structure of sentences up to point that is needed in each specific case. All of this without having to overanalyze the sentence as it happens when using a syntactic parsing. This fine-grained control is used in delete conditions, where the Bookseller takes advantage of certain words that are usually omitted by most of the other systems, but which are essential to understand the meaning of the sentence (e.g. the word “*any*” in “*any author*” suggests to clear a previously imposed AUTHOR condition). The Bookseller also handles the use of incomplete search values as it was seen in Section 4.2.2. Moreover, it has its own mechanism for resolving ambiguity. In Section 4.2.3 we saw how it combines restrictions in an intelligent way to generate a list of possible interpretations that are ranked according to a certain score.

Memory management. During a dialog, people often omit certain words that can be perfectly understood according to the situation. In a conversational system, the use of the memory allows putting the dialog in context. In this way, the user can refer to some concepts implicitly. On the other hand, when users interact with a natural language interface, sometimes they do not have a clear objective in mind, but that idea takes shape as the conversation flows. Initial restrictions are modified, new ones are defined, etc. For this reason, it is very important for the system to have a memory that can record any piece of useful information that can be found in the conversation.

Some systems like PRECISE and ORAKEL were not interactive and questions had to be self-contained since they could not reuse any past restriction because they did not have a memory to remember things. NaLIR showed a “query history”, but it just contained a list with the questions asked by the user. JUPITER did take account of the prior dialog context for processing user queries. For example, if the system listed a set of cities in California, during the recognition it preferred hypothesis that contained one of the cities on that list. MERCURY permitted users to modify previous legs by specifying updated constraints (e.g. “*I want to arrive in Tokyo a little earlier*”). NLA maintained a session context and integrated the constraints identified from the last input with the ones captured previously in the session. It also provided feedback on what constraints had been understood to that point. Finally, Anna could store objects, which could be used to answer questions about some of their properties without having to specify their names again. For example, if we asked “*black ADDE chair*”, and then “*how much does it cost?*”, Anna understood that the latter question referred to the object that was mentioned in the first question.

SmartSeller can also store different kinds of information units in memory. As we saw in Section 4.1.1, the Virtual Assistant uses an ontology to distinguish between objects, properties, specific subjects, general subjects, subject lists, and guided tours. Although this set is broader than those of the reviewed systems, the concept is essentially the same. What makes SmartSeller special is that we can specify different rules to control the way each expert agent reuses the memory. For example, a rule states that if a user asks a question that includes a PRICE restriction, the Bookseller may later transform it into an ORDER BY condition instead of throwing it away if it cannot be fulfilled when new restrictions arrive. Additionally, in Section 4.2.1 we saw how the Bookseller includes delete rules which allow the user to clear specific restrictions from memory without having to start a completely new query.

Multilingual. Internet is not only a mass medium but also a global medium. For this reason, being able to offer the information in different languages is essential when designing a conversational system that is aimed to the general public.

Although some systems like NLA stated that they could be easily scaled to support multiple languages, the reality is that many of them were only tested in one language. This is the case of MERCURY, NLA, PRECISE, ORAKEL and NaLIR, which were available only in English. The process of making a natural language interface multilingual is not always trivial. The

use of some techniques can help to reduce the effort that is necessary. For example, the statistical parser should enable NLA to scale to multiple languages and domains in a more robust and reliable way. They stated that to create a French version, they only needed to collect a corpus of French sentences and annotate them with the existing schemes, instead of recruiting French-speaking linguists to create rules for French expressions. JUPITER was originally in English, although the authors started to conduct research on multilingual conversational interfaces, including German, Japanese, Mandarin Chinese, and Spanish. At first, they just seemed to generate German reports and they did not handle input queries in these languages, but then they developed an end-to-end Japanese version (Nakano et al., 2001).

If we only took the number of languages into account, Anna would clearly stand out over the others since it was available in 16 different languages (English, Chinese, Czech, Danish, Dutch, Finnish, French, German, Hungarian, Italian, Japanese, Norwegian, Portuguese, Russian, Slovak, and Spanish). Nevertheless, it should also be considered how easy is to incorporate a new language into the system. Unfortunately, measuring this in an objective way is a difficult task.

SmartSeller has been designed to share the knowledge structure amongst all languages in order to avoid duplicities and reduce the maintenance costs. The system has been successfully tested both in English and Spanish, and the Virtual Assistant expert agent has also been tested in Portuguese. Although the language can be changed during the conversation, at this moment the system does not include a language detection mechanism, so the user must change it manually by means of the interface.

Temporal queries. When a conversational system or a natural language interface is aimed to search items in a catalog, it is perfectly normal for users to make temporal queries, that is, questions that involve relative dates and might retrieve different results depending on when they are asked.

Most systems are not able to handle temporal queries and, at the most, they just sort the results or filter them by a certain range of dates explicitly specified in the question. However, they cannot resolve relative dates such as “*this month*” or “*last decade*”. For example, Anna was not able to recognize questions like “*top ten tables 2014*” or “*best selling tables 2014*”. In the former case, she understood “*table top-angled*”. In the latter, she just recognized the word “*table*”. JUPITER did recognize temporal terms such as “*today*” or “*tomorrow*”. This could entail additional considerations. For example, calls after midnight asking for “*tomorrow’s*” weather really wanted “*today’s*” weather, defined from midnight to midnight. In MERCURY, the interpretation of dates was also essential for the proper recognition of questions (e.g. “*I want to fly from Boston to London on British Air next Friday*”). In the case of ORAKEL, this feature could not be confirmed, although it might be present because the use of a foundational ontology DOLCE provided predicates and relations related to time and space, which were crucial for representing the semantics of spatial or temporal prepositions.

SmartSeller is not only able to answer questions regarding advanced date filtering and sorting (e.g. “*poetry books from the*

50’s” or “*the last book of Suzanne Collins*”), but as we saw in Section 4.2.1, the grammar of the Bookseller also allows resolving temporal queries such as “*the best book of last year for children*” or “*ten most popular books of the month*”. This is a very powerful feature since users could get a general overview of the catalog by means of summaries.

6. Evaluation

In this section, we will present some details about the implementation and usage of the Bookseller.

The ontologies of the Virtual Assistant expert agent contains around 140 information units (IUs). The 38% of them are related to the bookstore, whereas the other 62% correspond to general knowledge and information about the avatar’s personality. These IUs are distributed in objects (21%), properties (29%), specific subjects (10%), general subjects (38%), and subject lists (1%).

The current version of the Spanish grammar of the Bookseller agent has approximately 150 semantic rules containing around 500 expressions (i.e. right parts). The English version is about 20% smaller (around 120 rules and 350 expressions), although the kind of questions it is able to answer is the same in both cases. The rules are organized in four different categories: restrictive rules (52%), sorting rules (15%), top rules (18%), and delete rules (15%). Top level rules (i.e. $\sim$ CONDITION) represent the 25% of all rules.

Most of the rules are restrictive because these include all the attributes of the database that can be used in the search process (title, author, publisher, category, label, date, price, ISBN, and EAN). On the other hand, at this moment sorting rules are only available for the most common attributes (date and price). However, the effort required to include new rules to allow sorting by other attributes is minimum. Moreover, as we said in Section 4.2.1, this set of rules is dynamically extended with specific rules that depend on the concrete question asked by the user.

The first version of the grammar was made in few days of development. It just contained the essential rules that allowed capturing the values of the different attributes of the database. In this way, we could have a completely functional system from the very first moment. However, when the first tests were carried out, we realized that this reduced set of rules generated a lot of ambiguity. Some of the questions asked by users contained common expressions that were not being produced by the grammar rules that we had made, but they were being captured by the dynamic rules that were added to the grammar automatically. In particular, these expressions partially matched some book titles generating undesirable restrictions. For this reason, we had to add more expressions, some complex rules and restriction filters during the following weeks, as users continued interacting with the system.

The effort necessary to adapt the system to different languages or application domains depends on each particular case. Taking our experience of switching from Spanish to English into account, we can state that, in general, it could be consid-

ered low or medium. Although each language has its own peculiarities, many grammar rules (e.g. those about dates, numbers, or prices) can be reused easily. In addition, the system can perform well with a relatively small number of rules. Obviously, the more complete the grammar is, the better for the recognition since less ambiguity is generated. On the other hand, all specific grammar rules are made dynamically cleansing the information from the database, so new domains might require to adapt those filters or implement new ones. In our case, considering a new language did not require new types of filters.

Regarding the usage of the Bookseller, although it is not publicly available yet, a reduced number of people (around 40) have interacted with it so far. Therefore, the following results should be taken carefully since they might be biased.

Users were asked to ask some questions as if they were actually looking for a book. They were demographically diverse. Some of them were colleagues from our computer science department, so they had a strong technical background. Others were owners and trustworthy customers of several real bookstores, so they had a more in-depth knowledge of the domain and the necessities of real users.

Users have asked approximately 500 questions along 200 sessions. These questions have an average length of 3.5 words. If we only consider those answered by the Virtual Assistant, the average falls to 2.4 words, whereas it reaches the value 4.3 for those answered by the Bookseller. Questions about books are usually larger because they contain some information about the book the user is looking for (e.g. the title). From our experience with other conversational systems, we think that this average will increase as more users interact with the system. For example, our virtual assistant for the University of Granada has received more than 500,000 questions along the first five years. The use of a 3D avatar makes some people ask longer sentences than what has been apparently observed in these tests, although the average is around 3.9 words.

If we focus on the behavior of the multi-agent system, 40.73% of these questions have been answered by the Virtual Assistant agent, whereas the other 59.27% have been handled by the Bookseller. After having performed a deep analysis, we can say that the 93.95% of all questions have been answered by the right agent. If we consider the remaining 6.05%, many of the errors (80%) are questions that had not been initially introduced in the knowledge of the Virtual Assistant (e.g. *"I would like a refund"*) and they partially matched the titles of some books.

Considering the Virtual Assistant agent, the 86.63% of the questions have been properly answered. These are related to the bookstore (e.g. *"What's your phone number?"*, *"How could I buy a book?"*), the private life of the avatar, greetings, compliments, rude words... Regarding misunderstood questions, they are mainly about topics not covered by the assistant (e.g. *"after-sales service"*) and some out-of-domain questions not related to the bookstore.

On the other hand, the Bookseller agent has answered the 84.69% of its questions successfully. Interpretations are made of 1.62 restrictions on average, which specify book titles, authors, publishers, dates, prices, and so on. This makes evident

the importance of having a good memory management module since restrictions are usually specified by users little by little. In this case, some errors have been caused by categories which are not defined in the book database and we had not included in our extended label hierarchy (e.g. *"adventure books"*), so they partially matched some titles (not so bad, but not so good either). In addition, some questions asked the Bookseller for some advice (e.g. *"I need a gift for my mother"*, or *"I want a book which doesn't make me think"*). This reflects the necessity of adding an advising module to the Decision agent.

7. Conclusion and future work

In this paper we have presented a generic multi-agent architecture for conversational systems that allows accessing heterogeneous data sources, the kind of problems that can be usually found nowadays in which the information may be scattered across different origins, each one with a particular structure and format. This architecture has been used to make SmartSeller, a specific system for the e-commerce portal of a bookstore.

Our system provides a great improvement regarding others if we focus on the broader concept of natural language interfaces (NLIs) that can be extensively used by any kind of public and not only expert users or database administrators. The majority of the systems have just focused on a very specific part of the problem. PRECISE, ORAKEL and NaLIR had interesting features as NLIs. For example, NaLIR supported complex SQL queries. However, they used just one knowledge source and lacked conversational capabilities. On the other hand, JUPITER, MERCURY and NLA did include a dialog manager, but they neither supported general domain queries nor were able to work with different types of data sources at the same time. In the case of a commercial system like Anna, it was much more complete in the conversational side, but not so advanced in the interaction with the database (e.g. it did not handle fuzzy or temporal queries), and it did not seem to include a full emotional model.

Regarding the future work, some aspects of the system related to the errors mentioned in Section 6 should be improved. For example, to solve the issue of the categories that are missing from the book database, we could apply data mining techniques to the descriptions of the books in order to augment our label hierarchy automatically by means of n-grams. We should also define new fuzzy labels to be able to answer some of the questions asked by users (e.g. *"I want a book which doesn't make me think"*). Moreover, this extended label hierarchy would become really useful when making recommendations.

It would also be very interesting to include a specific advising module into the Decision agent. This could be based on a finite state machine. In this way, users could be guided when they did not have a clear objective in mind. For example, if a user wanted to buy a present for someone else, and she did not know which book to buy, SmartSeller could ask for a book that the other person had enjoyed and look for similar books. In addition, this module would help to make the capabilities of the system evident to users.

On the other hand, as we said in Section 4.2.3, although for each question we have just focused on the best interpretation

found by the Bookseller, the system could be improved if the top 3 interpretations were combined and showed to the user when the best one matches only loosely the sentence.

Finally, with regard to the out-of-domain questions, the general knowledge of the system would be greatly improved if we added new expert agents to query external resources such as Wikipedia or DBpedia (Auer et al., 2007).

8. Acknowledgments

This work has been partially supported by the Spanish Ministry of Science and Technology under research project TIN2007-67984-C02-01, the regional Andalusian Government under research project TIC-P06-01424, a FPU scholarship from the Ministry of Education and Science, a Torres Quevedo research contract from the Ministry of Science and Innovation, and the King Abdullah University of Science and Technology.

References

- Androutsopoulos, I., Ritchie, G., Thanisch, P., 1993. MASQUE/SQL: an efficient and portable natural language query interface for relational databases. In: Proceedings of the 6th international conference on Industrial and engineering applications of artificial intelligence and expert systems. IEA/AIE'93. Gordon & Breach Science Publishers, pp. 327–330.
- Androutsopoulos, I., Ritchie, G., Thanisch, P., 3 1995. Natural Language Interfaces to Databases - An Introduction. *Natural Language Engineering* 1, 29–81.
- Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z., 2007. DBpedia: A Nucleus for a Web of Open Data. In: *The Semantic Web*. Vol. 4825 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 722–735.
- Auxerre, P., Inder, R., 1986. MASQUE Modular Answering System for Queries in English-User's Manual. Tech. rep., Technical Report AIAI/SR/10, Artificial Intelligence Applications Institute, University of Edinburgh.
- Bohus, D., Rudnick, A. I., 2003. RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda.
- Chai, J., Horvath, V., Nicolov, N., Stys, M., Kambhatla, N., Zadrozny, W., Melville, P., 2002. Natural language assistant: A dialog system for online product recommendation. *AI Magazine* 23 (2), 63.
- Chai, J., Lin, J., Zadrozny, W., Ye, Y., Stys-Budzikowska, M., Horvath, V., Kambhatla, N., Wolf, C., 2001a. The Role of a Natural Language Conversational Interface in Online Sales: A Case Study. *International Journal of Speech Technology* 4 (3-4), 285–295.
- Chai, J. Y., Budzikowska, M., Horvath, V., Nicolov, N., Kambhatla, N., Zadrozny, W., 2001b. Natural Language Sales Assistant - A Web-Based Dialog System for Online Sales. In: Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence Conference. AAAI Press, pp. 19–26.
- Cimiano, P., Haase, P., Heizmann, J., Mantel, M., Studer, R., 2008. Towards portable natural language interfaces to knowledge bases - The case of the ORAKEL system. *Data & Knowledge Engineering* 65 (2), 325 – 354, including Special Section: 3rd {XML} Schema and Data Management Workshop (XSDM 2006) - Five selected and extended papers.
- Codd, E. F., 1974. Seven Steps to Rendezvous with the Casual User. In: IFIP Working Conference Data Base Management, pp. 179–200.
- Damljanovic, D., Agatonovic, M., Cunningham, H., 2012. FREyA: An Interactive Way of Querying Linked Data Using Natural Language. In: *The Semantic Web: ESWC 2011 Workshops*. Vol. 7117 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 125–138.
- Eisman, E. M., López, V., Castro, J. L., 2009. Controlling the emotional state of an embodied conversational agent with a dynamic probabilistic fuzzy rules based system. *Expert Systems with Applications* 36 (6), 9698 – 9708.
- Eisman, E. M., López, V., Castro, J. L., 2012. A framework for designing closed domain virtual assistants. *Expert Systems with Applications* 39 (3), 3135 – 3144.
- Goddeau, D., Brill, E., Glass, J. R., Pao, C., Phillips, M., Polifroni, J., Seneff, S., Zue, V. W., 1994. Galaxy: A human-language interface to on-line travel information. In: *Third International Conference on Spoken Language Processing*.
- Grosz, B. J., Appelt, D. E., Martin, P. A., Pereira, F. C., 1987. TEAM: An experiment in the design of transportable natural-language interfaces. *Artificial Intelligence* 32 (2), 173 – 243.
- Gruber, T. R., 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5 (2), 199 – 220.
- Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D., Slocum, J., Jun. 1978. Developing a natural language interface to complex data. *ACM Trans. Database Syst.* 3 (2), 105–147.
- Huberman, B. A., Piroli, P. L. T., Pitkow, J. E., Lukose, R. M., 1998. Strong Regularities in World Wide Web Surfing. *Science* 280 (5360), 95–97.
- Kaufmann, E., Bernstein, A., 2010. Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases. *Web Semantics: Science, Services and Agents on the World Wide Web* 8 (4), 377 – 393, semantic Web Challenge 2009 User Interaction in Semantic Web research.
- Klein, D., Manning, C. D., 2002. Fast Exact Inference with a Factored Model for Natural Language Parsing. In: *Advances in Neural Information Processing Systems 15 (NIPS)*. MIT Press, pp. 3–10.
- Ley, M., 2002. The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives. In: *String Processing and Information Retrieval*. Vol. 2476 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–10.
- Li, F., Jagadish, H. V., 2014. NaLIR: An Interactive Natural Language Interface for Querying Relational Databases. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD '14. ACM, New York, NY, USA, pp. 709–712.
- Li, Y., Yang, H., Jagadish, H. V., 2005. NaLIX: an Interactive Natural Language Interface for Querying XML. In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. SIGMOD '05. ACM, New York, NY, USA, pp. 900–902.
- Miller, G. A., Nov. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38 (11), 39–41.
- Minock, M., 2010. C-Phrase: A system for building robust natural language interfaces to databases. *Data & Knowledge Engineering* 69 (3), 290 – 302, special Issue: 13th International Conference on Natural Language and Information Systems (NLDB 2008) - Five selected and extended papers.
- Moreo, A., Eisman, E., Castro, J., Zurita, J., 2013. Learning regular expressions to template-based FAQ retrieval systems. *Knowledge-Based Systems* 53 (0), 108 – 128.
- Nakano, M., Minami, Y., Seneff, S., Hazen, T. J., Cyphers, D. S., Glass, J. R., Polifroni, J., Zue, V., 2001. Mokusei: a telephone-based Japanese conversational system in the weather domain. In: *INTERSPEECH*. pp. 1331–1334.
- Noy, N. F., Sintek, M., Decker, S., Crubézy, M., Ferguson, R. W., Musen, M. A., 2001. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems* 16, 60–71.
- Pazos R., R., González B., J., Aguirre L., M., Martínez F., J., Fraire H., H., 2013. Natural Language Interfaces to Databases: An Analysis of the State of the Art. In: *Recent Advances on Hybrid Intelligent Systems*. Vol. 451 of Studies in Computational Intelligence. Springer Berlin / Heidelberg, pp. 463–480.
- Popescu, A.-M., Etzioni, O., Kautz, H. A., 2003. Towards a theory of natural language interfaces to databases. In: *Intelligent User Interfaces*. pp. 149–157.
- Raux, A., Bohus, D., Langner, B., Black, A. W., Eskenazi, M., 2006. Doing research on a deployed spoken dialogue system: One year of Let's Go! experience. In: *Proc. INTERSPEECH*, 2006. pp. 65–68.
- Seneff, S., Chuu, C., Cyphers, D. S., 2000. ORION: from on-line interaction to off-line delegation. In: *INTERSPEECH*. Citeseer, pp. 142–145.
- Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P., Zue, V., 1998. GALAXY-II: a reference architecture for conversational system development. In: *ICSLP*. Vol. 98. pp. 931–934.
- Seneff, S., Polifroni, J., 2000. Dialogue Management in the Mercury Flight Reservation System. In: *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems - Volume 3*. ANLP/NAACL-ConvSys '00. Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 11–16.
- Shankar, A., Yung, W., 2000. gNarLI: A practical Approach to Natural Language Interfaces to Databases. Term Report, Harvard University.

- Sijtsma, W., Zweckhorst, O., 1993. Comparison and review of commercial natural language interfaces. In: Proceedings of the fifth Twente Workshop on Language Technology. pp. 43–57.
- Staab, S., Studer, R., 2009. Handbook on ontologies. Springer, Berlin.
- Thompson, B. H., Thompson, F. B., 1983. Introducing ask, a simple knowledgeable system. In: Proceedings of the first conference on Applied natural language processing. ANLC '83. Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 17–24.
- Waltz, D. L., Jul. 1978. An English language question answering system for a large relational database. Commun. ACM 21 (7), 526–539.
- Warren, D. H. D., Pereira, F. C. N., Jul. 1982. An efficient easily adaptable system for interpreting natural language queries. Computational Linguistics 8 (3-4), 110–122.
- Wong, Y. W., Mooney, R. J., 2006. Learning for Semantic Parsing with Statistical Machine Translation. In: North American Chapter of the Association for Computational Linguistics.
- Woods, W. A., Kaplan, R., Nash-Webber, B., 1972. The LUNAR Sciences Natural Language Information System: Final Report. Final Report BBN Report No. 2378, Bolt, Beranek and Newman, Inc., Cambridge, Massachusetts.
- Zhou, L., Mohammed, A. S., Zhang, D., 2012. Mobile personal information management agent: Supporting natural language interface and application integration. Information Processing & Management 48 (1), 23 – 31.
- Zue, V., Seneff, S., Glass, J., Polifroni, J., Pao, C., Hazen, T., Hetherington, L., Jan 2000. JUPITER: a telephone-based conversational interface for weather information. Speech and Audio Processing, IEEE Transactions on 8 (1), 85–96.