SPOKEN DIALOG SYSTEMS: FROM AUTOMATIC SPEECH RECOGNITION TO SPOKEN LANGUAGE UNDERSTANDING

Antonio Rosario Intilisano - PhD Thesis

DECEMBER 10, 2015

DIEEI

University of Catania

Tutor Didattico : Prof. Vincenzo Catania

Tutor Aziendale : Ing. Maura Turolla

TABLE OF CONTENTS

ABSTRACT

A spoken dialogue system (SDS) is an intuitive and efficient application that allows humans to engage natural communication dialogs with machine to generally perform a specific task. From a high-level point of view, a SDS is composed of several modules which allow a user to engage a natural dialog with a machine in order to solve a problem or to retrieve information. The modules are involved in the dialog and work inside the SDS architecture like a pipeline: the output of a module is the input of the next one.

Each module of a SDS is very complex and it took years of research and hard work to provide solutions that can be exploited in real applications. During my thesis work, I have addressed the most important problems of the whole SDS Pipeline that are also the first two modules of the SDS pipeline: The Spoken Language Understanding (SLU) and Automatic Speech Recognition (ASR). The Spoken Language Understanding module takes as input the transcription of a user utterance, based on words, and output its interpretation, based on semantic concepts. The ASR works closely to the SLU module: it takes as input the audio speech signal of a user utterance, based on an audio file and output its translation (words), that is the input of SLU module.

In particular, ASR maps an acoustic signal into a sequence of phonemes or words (discrete entities). A typical ASR process performs the decoding of input speech data by means of signal processing techniques and an acoustic model. SLU component recognizes words that were previously included in its grammar. The development of a grammar is a time-consuming and error-prone process, especially for the inflectional or Neo-Latin languages. I developed a method that produces a grammar for different languages, in particular for Romance languages, for which grammar definition is long and hard to manage. This works describes a solution to facilitate the development of speech-enabled applications and introduces a grammar authoring tool.

A second problem concerning ASR is the limited availability of a valid speech corpus for designing a speech recognition acoustic model. Nevertheless, obtaining large datasets is generally both time- and resources consuming as it requires a continuous supervision of the entire building process. This works aims at showing the use of an algorithm for building speech corpora in an automatic way. It allows to

replace traditional manual transcription of audio-recordings and to automatically obtain a phonetic dictionary. An Italian acoustic and linguistic model were generated as use-case to test the effectiveness of the proposed procedure.

INTRODUCTION

In last decades Human-Computer interaction is becoming an easy available technology in everyday life. This thanks to impressive advancements in research topics like Automatic Speech Recognition and Understanding, Data-driven stochastic approaches, Text Processing and Speech Synthesis. One class of applications relying heavily on these technologies is Spoken Dialog Systems. A SDS is a computer agent that interacts with people by understanding spoken language. Nowadays, the SDSs market is a big slice of the human-computer interaction field. Many projects, open source and not, have been developed by several universities and companies. Many of these projects have been integrated into commercial technology. The first generation of SDSs was able only to recognize short dialogs or sometimes only single words. Specifically, each single word was bound to a specific functionality and there was no such a thing as a complete dialogue between system and user. The evolution of technologies and software architectures makes it possible to dialogue with the spoken systems and to perform actions that are the results of a dialog composed by different consequent interactions. Now, spoken dialogue technology allows various interactive applications to be built and used for practical purposes and research focuses on issues that aim to increase the system's communicative competence[1].

HOW SDS WORKS

The user starts a dialog as a response to the opening of a prompt from the system. The user utterance is automatically transcribed by the **Automatic Speech Recognition** (ASR) component. The ASR takes a speech signal as an input and produces its transcription in textual format. The SLU module takes the output of the ASR module and generates a meaning representation. Based on the interpretation coming from the SLU module, the **Dialog Manager** (DM) selects the next dialog turn, this is converted into a natural language sentence by the **Natural Language Generation** (NLG) module. Finally, the **Text-To-Speech** (TTS) module synthesizes the generated sentence as a speech signal, which is sent back to the user. The loop depicted in Fig. 1 is repeated until the application completes the modelled task.

[1] Marco Dinarelli, International Doctorate School in Information and Communication Technologies DISI - University of Trento Spoken Language Understanding: From Spoken Utterances to Semantic Structures

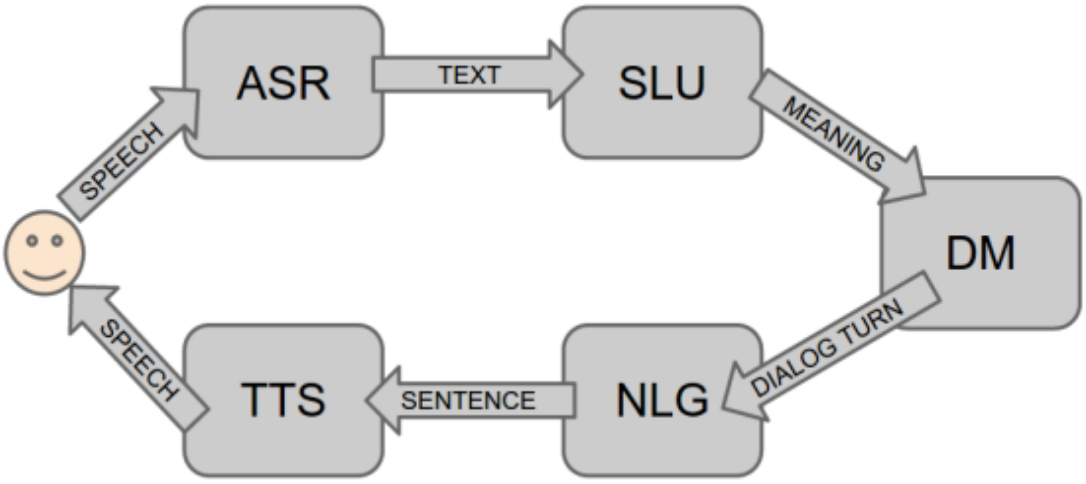

Figure 1 General SDS architecture

Spoken Language Understanding (SLU) is the semantic interpretation of signs conveyed by a speech signal. The goal of SLU is to extract a conceptual representation from spoken sentence transcriptions in a natural language. This task is very complex. Signs to be used to produce conceptual representation are coded in the signal along with other noisy information.

Spoken sentences many times don't follow the grammar of a language, they could contain self-correction, hesitations, repetitions and many other irregular phenomena due to the spontaneous nature of spoken language. Furthermore, SLU systems are applied to the output of an ASR so they must be robust to noise introduced by spontaneous events typical of spoken language and errors introduced by ASR. **ASR components produce a stream of words with no information about sentence structure**, like punctuation and sentence boundaries, so SLU systems must perform text segmentation and understanding at the same time.

OLYMPUS FRAMEWORK

In my PhD course I investigate on various SDS Framework. **I chose to use Olympus for its completeness, software license and current state of development**. Olympus is a complete framework for implementing spoken dialog systems created at Carnegie Mellon University. Olympus incorporates the Ravenclaw dialog manager [2], which supports mixed-initiative interaction, as well as components that handle speech recognition (Sphinx)[3], understanding and generation. Olympus uses a Galaxy[4] message-passing layer to integrate its components and supports multi-modal interaction. As shown in Fig. 2, the Galaxy architecture is a constellation of Galaxy Servers, which communicate with each other through a central Galaxy Hub. In Olympus, most of the agents are Galaxy Servers. Specific functions, for instance dialog planning, input processing, output processing, error-handling, etc. should be encapsulated in subcomponents with well-defined interfaces that are decoupled from domain-specific dialog control logic. Olympus users are able to inspect and modify each of these components individually, towards their own ends. For each Olympus application we need to specify resources like grammar, language model and dialog task specification. These resources serve as a basis of knowledge that is necessary to understand and to interact with users.

[2] Dan Bohus, Alexander I. Rudnicky, "The RavenClaw dialog management framework: Architecture and systems", Computer Speech and Language, vol. 23, no. 3, 2009

[3] K.-F. Lee, H.-W. Hon, and R. Reddy, "An overview of the SPHINX speech recognition system, "IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 38, No. 1, January, 1990, pp. 35 - 45.

[4] Joseph Polifroni, and Stephanie Seneff, "Galaxy-II as an Architecture for Spoken Dialogue Evaluation",

Proc. LREC, 725-730, Athens, 2000

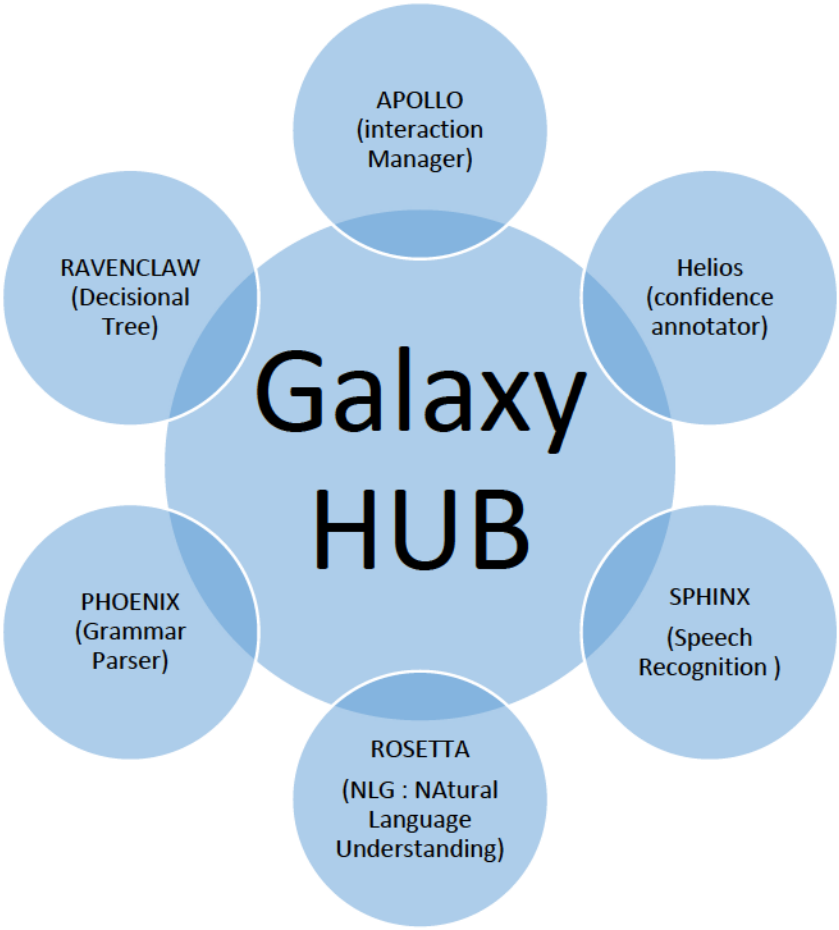

Figure 2 Galaxy architecture

SPOKEN LANGUAGE UNDERSTANDING

To build a Spoken Language Application in a specific user language, the developer has to design and develop a knowledge base called grammar for SLU. The development of a grammar can be greatly accelerated by using a corpus describing the application or a tool that automatically extends grammar coverage [5]. However, **the development of such a corpus is a slow and expensive process**[6]. In SLU research domains specific semantic grammars are manually developed for spoken language applications. Semantic grammars are used by robust understanding technologies[7] to map input utterances to the corresponding semantic representations. Manual development of a domain-specific grammar is time-consuming, error-prone and requires a significant amount of expertise. It is difficult to write a rule-set that has a good coverage of real data without making it intractable[8]. **Writing domain-specific grammars is a major obstacle to a typical application developer**. This specialization often does not cover any unspecified data and it often results in ambiguities[9]. These difficulties are further accentuated if a regular software developer does not know the desired user language that the spoken dialog system (SDS) uses. A further level of abstraction, especially for the Latin languages is necessary.

SLU GRAMMAR DEVELOPMENT

To facilitate the development of speech-enabled applications, it is necessary to have a grammar authoring editor that enables regular software developers with little speech/linguistic background to rapidly create

[5] S. M. Biondi, V. Catania, Y. Cilano, R. Di Natale and A.R. Intilisano, “An Easy and Efficient Grammar Generator for Spoken Language Understanding,” The Sixth International Conference on Creative Content Technologies (CONTENT) pp 13-16, Venice, May 2014.

[6] I. Klasinas, A. Potamianos, E. Iosif, S. Georgiladakis, and G. Mameli, "Web data harvesting for speech understanding grammar induction," in Proc. Interspeech, Lyon, France, Aug. 2013.

[7] J. F. Allen, B. W. Miller, E. K. Ringger, T. Sikorshi, “Robust understanding in a dialogue system,” 34th Annual Meeting of the Association for Computational Linguistics. Santa Cruz, California, USA, pp. 62–70, 1996.

[8] H. M. Meng and K-C. Siu, “SemtiautomaticAcquition of Semantic Structures for Understanding Domain-Specific Natural Language Queries,” IEEE Tran. Knowledge & Data Eng., pp. 172–181, vol. 14(1), 2002. [

[9] Y. Wang and A. Acero, "Grammar learning for spoken language understanding," Automatic Speech Recognition and Understanding, ASRU '01. IEEE Workshop, pp. 292-295, 2001.

quality semantic grammars for SLU[10]. More precisely, **the main purpose of my PhD thesis is to ease the development of a CMU Phoenix Grammar**[11], a SLU parser of the Olympus Framework. This is accomplished by introducing an intermediate grammar that helps generating a simpler, reusable, and more compact grammar. The development process allows obtaining large amounts of grammar contents starting from a few rows of the new grammar that we are introducing. The grammar has a greater coverage than the standard grammar developed by a regular software developer. In addition, it is possible to write this grammar in the English language and our tool creates the grammar in the SDS user language. Therefore, I have developed a standard grammar that produces a multiple-language support to an application SDS in a simple way. **The effort to build the corpus is reduced by the ability of my tools to automatically extend the coverage of the grammar**. It currently supports the generation of a grammar for the Italian language, but the method can be applied to all the Romance or Neo-Latin languages. In order to test the validity of our solution, a specified grammar editor has been developed. It permits the automatic conversion of the new grammar format to the CMU Phoenix grammar parser[12]. **The purpose of my PhD course is also to increase developer productivity**; experimental results show that it also improves the coverage of the final Phoenix parser grammar.

PHOENIX PARSER GRAMMAR

To build a specific Spoken Language Application in the Olympus Framework the developer has to design and develop specific grammar definition. The Phoenix parser uses a formal method and a hand crafted CFG Grammar. It requires combined linguistic and engineering expertise to construct a grammar with good coverage and optimized performance. First of all, the developer has to determine the main set of jobs that the application will handle. Each concept or action defined in the dialog manager is mapped in one or more grammar slots. Therefore, the design of the grammar is strictly bound to the design of the dialog tree. Grammar rules are specified in the source grammar file. The manual development of Phoenix grammars is a time-consuming and tedious process that requires human expertise, posing an obstacle to

[10] Y.-Y. Wang and A. Acero, “Rapid development of spoken language understanding grammars,” Speech Communication, vol. 48, no. 3-4, p. 390416, 2008.

[11] W. Ward, "Understanding spontaneous speech: the Phoenix system," Acoustics, Speech, and Signal Processing, ICASSP91, 1991 International Conference, pp. 365-367 vol. 1, 14-17 Apr 1991.

[12] Phoenix Parser User Manual, http://www.ontolinux.com/community/phoenix/Phoenix_Man ual.pdf (last visited: 22 November 2013).

the rapid porting of SDS to new domains and languages. A semantically coherent workflow for SDS grammar development starts from the definition of low-level rules and proceeds to high-level ones. The Olympus framework provides English generic grammar files, which contains some standard forms such as greetings, social expressions and yes/no, as well as discourse entities such as help, repeat, etc. This grammar has to be extended by introducing domain-specific phrases. As Minker and Chase[13] explained, the Phoenix parser is based on case grammar which analyze the syntactic structure of utterances by studying the combinations of verbs and their related nouns. The main characteristics of case grammars are the possibility of easily sharing the syntactic patterns and grouping related concepts or features together. As shown in Fig. 5 the Phoenix parser is based on frames. Every input word is mapped into a sequence of case frames composed by a set of slots. Each slot has a specified context-free grammar (CFG) that describes the possible content of the slot. The grammar is compiled into **Recursive Transition Networks** (RTNs) that are associated to each slot. The parsing mechanism consists on a comparison between given input words and frames, according to the rules defined in the grammar. When a RTN matches a word it is added to the chart. This is a recursive matching process in which RTNs can call other RTNs. At the end of the process the chart consists of a sequence of slots containing the semantic parse tree of the sequence of input words.

[13] Minker, Chase, "Evaluating Parses for Spoken Language Dialogue Systems", in First International Conference on Language Resources abd Evaluation – Proc. Workshop on The Evaluation of Parsing Systems, May 1998

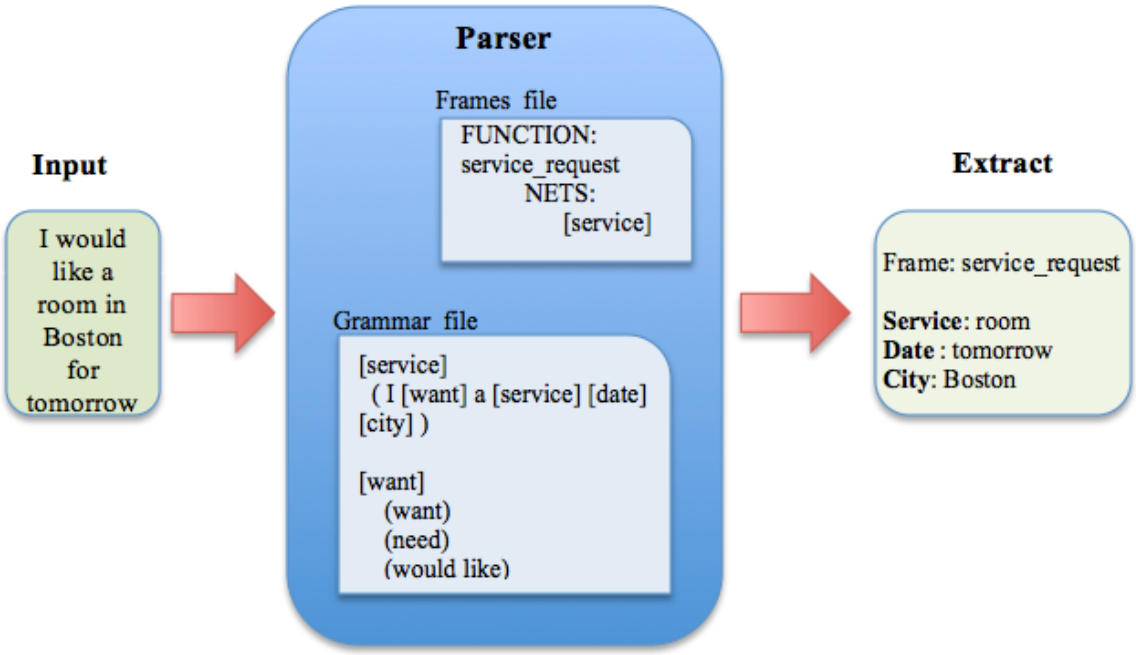

Figure 3 Phoenix Grammar Parsing

The Phoenix semantic parser is an easy and robust SLU parser and it is used into the Olympus framework[14]. This parser maps input word strings on to a sequence of semantic frames:

- Named set of slots, where the slots represent related pieces of information.
- Each slot has an associated CFG that specifies word string patterns that match the slot

The CFG describes all the sentences that the system can understand in a meaningful, readable and synthetic form. For SDSs that allow slightly more flexibility to the user, the number of possible

sentences are so huge that it becomes impossible to list them all. For example, a user could say “HELLO”, "GOOD MORNING" or "HI THERE", but they have the same meaning from the SDS’s point of view.

The Phoenix Grammar [14] contains one or more grammar-slots that define a specific meaning by a list of sentences. For example:

[14] D. Bohus, A. Raux, T. K. Harris, M. Eskenazi, and A. I. Rudnicky, “Olympus: an open-source framework for conversational spoken language interface research,” NAACLHLT ’07: Proceedings of the Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technologies, 2007

[greeting]

(hello)

(hi there)

(good morning)

;

OLYMPUS P2P

My first work in the SDS field is a multisession and self-learning version of Olympus called OlympusP2P. Olympus P2P introduces features such as **Multi-Session** and **Self-Learning Dynamic Grammar**. These two last features introduce the concept of shared resources. The system-user interaction is enhanced because the common resource can be updated by all active sessions. This creates new self-learning scenarios in which a common resource is continuously updated by different sessions. Take into consideration the following aspect: a single Olympus instance represents a single point of failure; a single Olympus instance is not able to "increase" or "decrease" the number of available sessions in function of the user's needs and availability of hardware/software resources and it is not able to replicate the active sessions in function of new users requests; the client must know the address of each Olympus session. Olympus P2P concurrently serves more users allowing the sharing of resources among all sessions. If one of the Olympus sessions is able to update its knowledge during the interaction with the user, then all other sessions will benefit from it. Moreover, allocating dynamically new Olympus sessions depends on the number of users. This becomes necessary in large scales systems. My goal is not only to manage multiple, scalable and robust instances and to face the growing demand for sessions by users, but also to use the parallelism of several active sessions to share useful resources in order to improve the user interaction. The new component, implemented by a WCF Technology is based on a partially decentralized P2P Network. **Olympus Super Peer nodes** play an important role in this network, carrying out specific functions, such as a distributed search and Olympus session discoveries among multiple peers around the network. This is completely transparent to the client application. The grammar and other resources are shared by all Olympus Super Peer nodes, unlike client-server architectures in which the server shares all contents and resources. P2P is more reliable since the central dependency is removed. Failure of one peer does not affect the work of other peers. Specifically, we have two types of nodes, as shown in Fig.

- **Olympus Peer** (OP): This is the interface to the client application (client login, client logout, exchange of information). It contains one or more Olympus active instances and it is connected to a single Olympus Super Peer.

- **Olympus Super Peer** (OSP): This deals with the research of active sessions in the network. It knows at least another OSP in the network. It maintains a list of active peers with its active sessions and it can perform a query only to the known OSP.

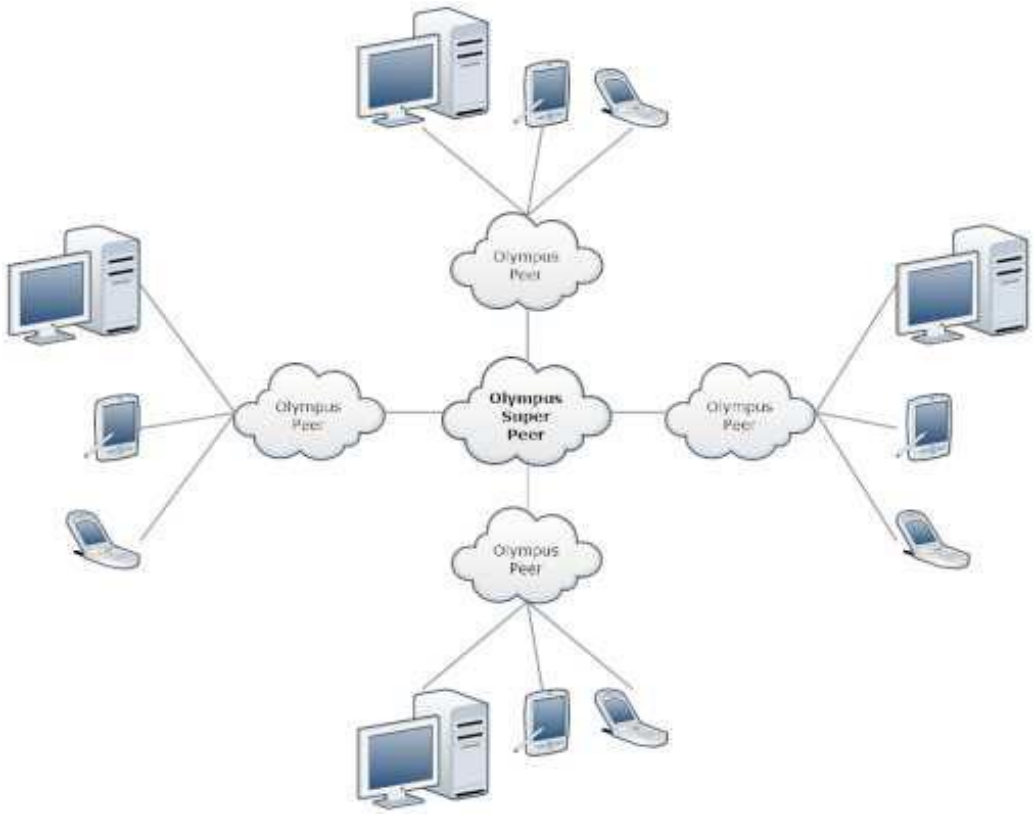

Figure 4 : Olympus p2p Architecture

OLYMPUSP2P ARCHITECTURE

For every OSP we have a set of OPs that are connected with a number of Olympus active sessions which serve a set of clients. Each client will only know a limited set of OPs in the network (the list can be centralized or available to the client at the time of installation). After that the authentication has been performed, if the OP contacted by the client has free sessions of the type specified by the client, then it accepts the connection. If the OP does not have free sessions, it performs a query to the OSP which will query the other OPs. The client will communicate through the same OP contacted and it will receive the URL of the OP with a free active session. The search is performed among a "network of OSPs" with techniques derived from flooding broadcast. The messages are exchanged only between the same OSP. Furthermore, each OSP contains the grammar used by its OP, which through the mechanism explained below can be updated in every iteration. Fig. 5 shows how the OSP at each update informs the other OSPs in the network and its own OPs that the grammar has changed. Finally, the compiled grammar will be downloaded and updated. In this way all sessions in the network will benefit from real sessions.

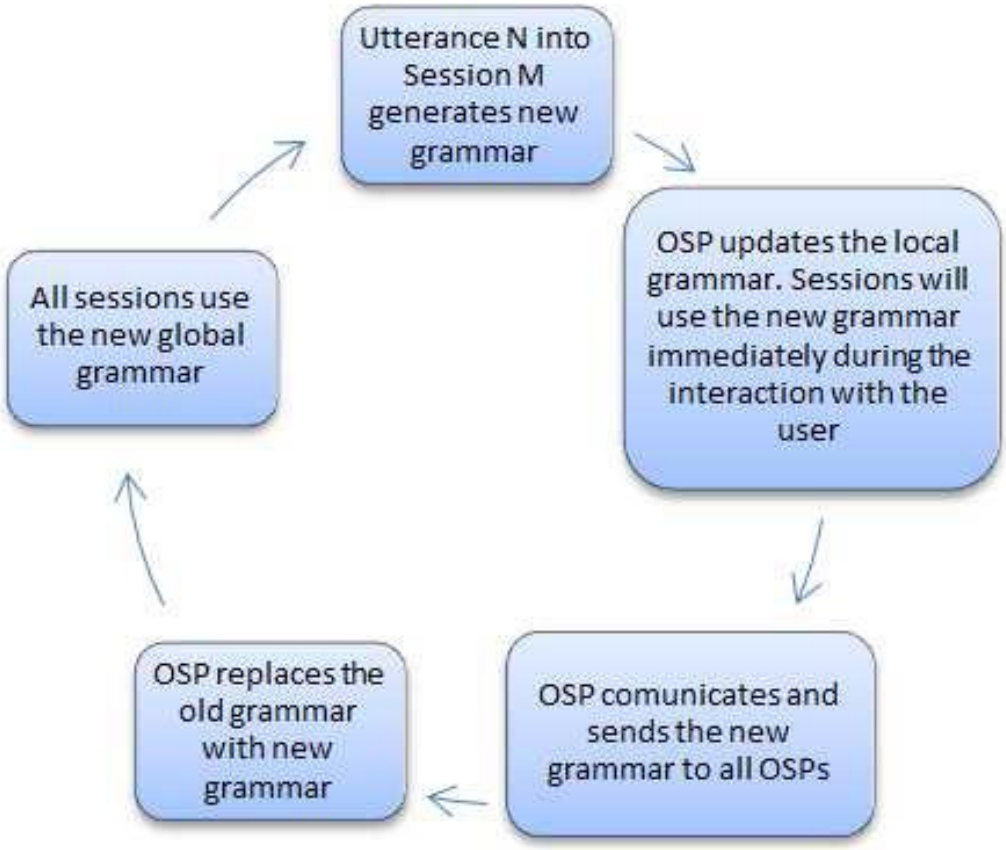

Figure 5 How session works

As shown in the Fig. 6, the idea is to create an additional software layer between application clients and Olympus, with the main goal to increase the active available sessions.

Figure 6 : P2P Architecture

These goals are made possible because of the P2P architecture that manages multiple instances of Olympus and the ability to globally share a self-learning grammar updated in real time by what happens in the individual sessions.

SELF-LEARNING DYNAMIC GRAMMAR

The development of new technologies has led companies and government institutions to provide services through automatic systems, which help people find information that is needed. In the past years the evolution of technology has changed the mechanism behind the user-system interaction. One of the first examples of an automatic interaction system uses dual-tone multi-frequency signaling over analogic phones, which guides users through different menu options. This set of options is given to the user who is then able to select his/her choice by pressing the corresponding number. These systems provide a finite number of possible choices that are extended introducing sub-menus in a classic tree structure. The introduction of speech recognition systems and spoken language generators has led to a different approach where users can have a "natural" conversation with the system. In these systems the parser has a central role, typically being the bond between the speech recognizer software and the language-processing unit.

The basic role of the parser is to give a meaning to user inputs. Many parsing algorithms have been developed enclosing syntactic and semantic rules inside the grammar. The pursuit of always having a better and more natural interaction brought the development of more complex grammars. The production of an efficient, self-learning and auto generating grammar requires a high amount of human and temporal resources. Different algorithms and approaches have been adopted in the literature of natural languages. These can be clustered in different categories according to different approaches or different grammar types.

PRE-COMPILED GRAMMAR

The actual version of the framework uses the Phoenix parser and its **pre-compiled grammar**; therefore, any spoken dialog system built over Olympus needs a pre-compiled grammar. The grammar compiler in the Phoenix parser creates the output compiled grammar files from a set of grammar specification files (.gra, .class). This grammar mechanism used by the Phoenix parser, and therefore by the Olympus framework does not allow an easy update of the grammar. The system has to be stopped and the grammar has to be re-compiled every time that a new word is added to the system knowledge. In a research environment this is acceptable, however in the industrial field this mechanism is tedious especially for those companies who want the service always available without delays. Thanks to this mechanism when a new word is added to the system knowledge all active sessions in the network will automatically use the new grammar.

DYNAMIC GRAMMAR COMPILER

My implementation, OlympusP2P, **introduces a dynamic grammar compiler mechanism**. This mechanism is based on a new compiling method along with a self-learning algorithm, and it allows the grammar to be updated without having to stop the system. The idea is to build a **new apposite Olympus module that has the responsibility of updating and then compiling the grammar**. This new module, like other Olympus modules, can be implemented as a Galaxy server. Therefore, the entry point can be reached by using an IP address and a port number. The communication between the new module and the Phoenix parser can be accomplished in two ways: by using direct sockets between them, or by using the Olympus HUB module as a dispatcher of events. In the first case, the overhead of the communication does not affect the dispatching of events by the HUB module, therefore the Olympus framework performances are not affected. In the second case, we use the basic Olympus framework mechanism of sending frames

to the HUB module. When the grammar specification files are updated and re-compiled, the new module informs the HUB that a new compiled grammar is available, and then the HUB informs the Phoenix parser to load the new grammar. This solution introduces two new frames to be dispatched over the Olympus framework network. Realistically, at the beginning these new frames will be exchanged with a high frequency due to the fact that it is more likely that there are new words not contained in the grammar. The frame exchange will occur more rarely when the grammar reaches a steady state. Therefore, at a regime state, this solution does not affect the general overhead in terms of performance and latency of messages, and so the system maintains an acceptable response time.

ALGORITHM

My implementation changed the way in which the grammar is loaded by the Phoenix parser, making it possible to load the grammar and reinitialize all the structure currently in the memory at run time. **In this way the system can continuously update the grammar without having to stop and restart its activities**. Hence, even while conversing with a user the grammar can be updated, and all the next input utterances will be processed with the new grammar. The pursuit of self-learning grammar has characterized a big portion of the natural language processing, and in general the artificial intelligence field. Many unsupervised, semi-supervised or supervised algorithms have been developed in the past years. My approach represents a simple way to achieve the goal of populating basic grammar notions in a specific context. The purpose of my implementation is to enlarge the initial grammar database with all the new terms that are somehow related to those already in the grammar specification files. The basic idea is to correlate these words according to some criteria (for example, synonyms). When one, or more criteria are triggered, the self-learning grammar module updates the grammar files. The set of criteria that we chose will affect the grammar update frequency rate. Therefore, the less strict the criteria are, the less frequently the grammar will be updated. Instead, if we set weak criteria the grammar will be updated too often and there will be a higher chance to introduce words that do not make sense. Starting from a given an input word we retrieve a list of synonyms by using online web services, which provide a set of synonyms through a simple GET request. It is possible to select one or more web services and to use the union or the intersection of their results.

Fig. 7 describes the algorithm supposing that we want to populate the grammar according to synonyms rules. When an unknown word is synonymous of a word that is already present in the knowledge, it will be added to the grammar. This mechanism is carried out by processing the parser output for each input sentence. By analyzing the Phoenix parser output we create a list of words that are not recognized. Since we do not know if one or more words on the list are correlated between each other, we perform a combination of all of them.

```
Input:

      List of utterances U = {u1, u2, …, un}

      Starting Grammar

Output:

      Updated Grammar

Algorithm:

FOR EACH cycle

      compile(grammar)

      FOR EACH inputUtt ∈ U do

            unknownWords <- parse(inputUtt)

            wordsToProcess <- combination(unknownWords)

            FOR EACH word ∈ wordsToProcess

                  Synonyms[word] <- getSynonyms(word)

                  FOR EACH synonymous ∈ Combination(Synonyms)

                        newUtt <- replace(inputUtt, word, synonymous)
```

Figure 7: Self-learning grammar algorithm based on synonyms

For instance, if the list of unknown words contains the words “look and for”, we will process the words look, for, and look for. A list of synonymous for each of these terms is retrieved from external resources and stored along with the original word. Each synonymous of each unknown word is replaced in the original utterance and given to the Phoenix parser as input. If the number of parsed words of the new utterance is higher than the number of parsed words with the original sentence, then we have found a synonymous. **The synonymous found is then written in the corresponding file in the grammar**. Fig. 6 shows how, step-by-step, the new module adds words that are not directly correlated to the knowledge. For instance, the verb look for is not seen as possible synonymous of want, but when the system discovers seek as possible synonymous of want, then the word look for will be also discovered as synonymous of want.

SLU IN NEO-LATIN LANGUAGES

All Romance languages have common features, so one could imagine a SLU system that takes into account these characteristics and shows the same behavior for such languages. This section explains some features of the Romance languages. The Neo-Latin or Romance languages are the direct continuation of Latin, a language with a very rich dictionary. In fact, Latin has a high level of perfection as it was the idiom of a population with an advanced degree of civilization[15]. Today, many voices have disappeared; instead, others are present in the Romance languages[16].

The Latin lexicon had been always in continuous evolution and at the time it had become wealthy of new elements, at times taken from foreign languages, but mainly through the addition of suffixes, for example to create diminutive forms. The creation of new forms through the addiction of suffices is also a characteristic of the Romance languages, in fact, based on a word that has a particular suffix, infinite other words can be formed.

Today, there are many Neo-Latin languages, the main are: Italian, Portuguese, Spanish, French, Provenzal and Romanian. A characteristic of the Romance languages is, as in Latin, the creation of inflections. The Romance languages are highly inflectional, in which each inflection does not change the part of speech category but the grammatical function. In general, the inflected forms are obtained by adding to the root of a canonical form a particular desinence (but there are some irregular cases in which also the root changes, this phenomenon is called apophony).

Conjugations are inflections of verbs; they provide information about mood, tense, person, number (singular or plural) and gender (masculine or feminine) in the past participle. Declensions are inflections of nouns and adjectives; they provide information about gender and number.

The conjugations, which in Latin are four, in Romance languages are three. According to the conjugations, different declensions are applied: the first conjugation is for verbs that end in "-are", the second is for

[15] Lingue Neolatine in Treccani.it - Enciclopedie on line. Istituto dell'Enciclopedia Italiana.

[16] Grammatica Storica in Treccani.it - Enciclopedie on line. Istituto dell'Enciclopedia Italiana.

verbs that end in "-ere", the third is for verbs that end in "-ire" (in Latin, there is a distinction between -ĒRE and -ĔRE).

In the transition from Latin to the Romance languages, in some cases there have been passages of conjugation (called "metaplasmi"). In general, in verbs moods and temps have not changed, but there are disappeared or innovated forms for function or meaning. The disappeared forms are: deponent verbs (that are verbs with passive form and active meaning), simple future (the simple future of the Romance languages is not derived from Latin), perfect subjunctive, future imperative, future infinitive, supine, and gerundive.

The alterations have occurred for various reasons, for example, for phonetic problems, many "b" were turned into "v", because their pronunciation was very similar (e.g., "cantabit" in Italian becomes "cantavi"). In Latin verbs, many tenses have similar declensions (e.g., the future perfect and the perfect subjunctive, the subjunctive and the infinite present). As a result, many verbal forms have disappeared (e.g., "supine", gerundive, declensions of infinitive, future participle) and have been replaced by forms that are more expressive. In this way, new verb forms were born.

To form the future, different periphrastic constructions were created, for example, the most common is derived from the union of the infinitive and the reduced forms of the present indicative of "habere", with the accent on the auxiliary verb (e.g., the Latin form "cantābo" becomes "canterò" in Italian, "chanterai" in French, "cantaré" in Spanish). The conditional does not exist in Latin and in the Romance languages it is derived from the union of infinitive and the reduced forms of perfect or imperfect of "habere" (e.g., "canterei" in Italian, "chanterais" in French, "cantaria" in Spanish). Periphrastic forms with the past participle, as passive forms and all the compound tenses, are typical of the Romance languages (e.g., the Latin form "amor" becomes "io sono amato" in Italian, "je suis aimé" in French).

There are Latin verb forms that have transformed their function, for example the pluperfect subjunctive has the meaning of imperfect subjunctive (e.g., the Latin "cantavissem", that meant "avessi cantato" in Italian, now means "cantassi", "chantasse" in French, "cantase" in Spanish); this happened because the imperfect subjunctive in Latin ("cantarem") was too similar to the present infinitive. Therefore, the Romance languages have a very similar way to create inflections of verb, nouns and adjective and suffixed forms. This allows creating, for these idioms, similar algorithms for generation and morphological analysis.

ITALIAN LANGUAGE

Like all the Romance languages, Italian is highly inflectional. Italian has three conjugations for verbs, each conjugation involves the application of specific suffixes: the verbs that end in "-are" belongs to the first conjugation, the verbs that end in "-ere" belongs to the second and the verbs that end in "-ire" to the third. Each inflected form of a verb gives information about mood, temp, number and person, and gender and number in the case of the participle. In Italian, there are many irregular verbs. Irregular verbs that end in "-rre" belong to the second conjugation. Italian irregular forms often originate from Latin irregular forms.

Latin had five declensions of nouns and adjectives, which have undergone a significant rearrangement. In Italian nouns and adjectives create inflections in various ways, for example to form the plural some nouns remain the same, others have many plural, sometimes with different meanings. Many nouns are irregular when the gender changes. Nouns and adjectives are subject to alteration that is the addition of a suffix to change the meaning in evaluation, quantity or quantity. Adverbs are not inflected and can be obtained by adding a particular suffix to some adjectives.

Italian has many orthographic rules related to its phonetic. Italian words can be reproduced by the combination of 28 different sounds called phonemes. There is not always a correspondence between phonemes and letters, in fact, some letters represent different sounds according to the following vowel[17]. For example, if "c" and "g" are followed by the vowels "a", "o" and "u", they produce a hard sound and if the vowels "e" and "i" follow them they produce a soft sound. To obtain the corresponding hard sound the letter "h" is inserted between these characters and vowels "e" and "i"; to obtain the soft sound with the vowels "a", "o" and "u" the character "i" is inserted.

There are other orthographic rules that concern the behavior of groups of two or three characters as "sc", "gn" and "gl".

[17] F. Musso, N. Prandi, "Per dirla giusta. Fonologia, ortografia, morfologia", S. Lattes & C. Editori SpA, 2012.

MORPHOLOGICAL ENGINE

For the Italian language field, we extend my previous research[18] [19]in which we addressed the problem of the multi-session management and of a hand-crafted self-learning grammar. We present an alternative implementation of the SLU Grammar of an SDS for inflectional languages. The new version will introduce benefits such as: minor grammar development effort, better grammar representation and structure, easier grammar management and easier user behavior prediction. This is achieved by using a morphological analyzer[20] of a specific inflectional language. We will show a test evaluation upon a case scenario in the Italian language, but the same idea may be extended and reused for other inflectional languages Dialog Systems. The aim of the Morphological Engine is to provide sentences with all words in their canonical form for the Phoenix Parser. For this reason, a morphological engine for the Italian language has been used. In Italian, a word can have different meanings in a sentence: the word “cammino”, for example, can be a verb (in English “to walk”) or a noun (in English “path”). So, it is necessary to choose the right form. For this reason, the morphological engine has been combined with a syntactic analyzer that is a specific version of the Link Grammar Parser (LGP)[21] for the Italian language. It solves cases of ambiguity and it provides sentences containing all the words in their canonical form. In this way, the Phoenix Parser does not have to know all the inflected forms, but only the canonical forms. Following paragraphs describe each component of the Morphological Engine in detail.

[18] V. Catania, R. Di Natale, A. Intilisano and A. Longo. 2013. A Distributed Multi-Session Dialog Manager with a Dynamic Grammar Parser, International Journal of Information Technology & Computer Science (IJITCS) (ISSN: 2091-1610).

[19] Vincenzo Catania, Raffalele di Natale, Antonio Rosario Intilisano, Salvatore Biondi, Ylenia Cilano. "An Easy and efficient grammar generator for Understanding Spoken Languages". International Journal on Advances in Intelligent Systems, vol 8 no 1 & 2, year 2015,

[20] Vincenzo Catania, Ylenia Cilano, Raffaele Di Natale, Daniela Panno and Valerio Mirabella, A morphological engine for Italian language, 2nd International Conference on Internet, E-Learning and Educational Technologies (ICIEET 2013), Istanbul - Turkey, 2013.

[21] Daniel Sleator and Davy Temperley. 1991. Parsing English with a Link Grammar. Carnegie Mellon University Computer Science technical report CMU-CS-91-196, October 1991.

LINK GRAMMAR PARSER (LGP)

The morphological engine has been combined with a syntactic analyzer that is a version of the LGP for the Italian language. Originally, the LGP has been created for the English language at the Carnegie Mellon University. The LGP is based on the link grammar, a theory of English syntax that builds relations between pairs of words. Each group of words is associated with a set of rules that determine these relationships. In the standard version, the LGP loaded all words in a very large dictionary that for each “tag” included all inflected forms associated. By the fusion with the algorithm of morphological analysis the LGP receives as input a vocabulary with only the words of the phrase. Each rule refers to a “tag” that is associated to the words with particular characteristics. For example, there was a "tag" associated with all transitive verbs in the third singular person, another associated with the feminine singular nouns, one for prepositions, etc.

Through the analysis of grammatical categories and the desinences, the morphological engine associates a particular "label" to each word of the sentence. This label contains information regarding the inflection and the grammatical category to which the word belongs. In this way, the LGP receives as input only a string in which each label corresponds only to any words in the sentence. Precisely, each label corresponds to a particular "tag", each of which is associated with different syntactic rules. This mechanism allows the LGP to reconstruct more quickly the parse tree of the sentence.

PHOENIX GRAMMAR

Since spontaneous speech is often ill formed and the recognizer makes errors, it is necessary that the parser is robust to recognition errors, grammar and fluency. This parser is designed to enable robust parsing of these types of input. The Phoenix parser uses a specific CFG grammar that is organized in a grammar file. Names of grammar files end with a “.gra” extension. This contains context-free rules that specify the word patterns corresponding to the token. The syntax for a grammar for a token is in Fig. 8. In Fig. 9 there is an example.

```
# optional comment
[token_name]
      (<pattern a>)
      (<pattern b>)
;
```

Fig 8 Generic Phoenix grammar syntax.

```
[city]
      (New York)
      (London)
      (Paris)
;
```

Fig 9 Example of a Phoenix token.

A token can also contain other tokens, for example (Fig. 10):

```
[token_example]
      (word1 [other_token] word2)
;
```

Fig 10 Example of a Phoenix token containing other tokens.

This format allows recognizing several sentences with the combination of different slots and words; furthermore, each token can be reused in many tokens.

In the inflective languages, as Italian or Romance languages in general, words can occur in several forms, verbs can change its form depending on conjugations and nouns and adjectives depending on declensions. Their forms can change also applying different suffixes or prefixes. This means that the Phoenix grammar must contain all the possible inflected forms. For this reason, the grammar can become long and hard to write, because the developer must manually write it and he might forget some inflected forms: the result can be a not complete grammar. This increases the development time.

Thus, inflected forms add complexity to the Phoenix grammar, since they generate multiple different rules with similar patterns.

NEW GRAMMAR: SMART GRAMMAR

The development of a new domain application needs a new Context Free Grammar (CFG) that is able to define the concepts and their relations of such domain. Alternative approaches learn structures from a set of corpora. However, this process appears too expensive and potentially not exhaustive[22]. My approach consists of creating a new intermediate grammar called Smart Grammar[23] that focuses on the meaning of a grammar token rather than on its content. The legal combination of individual words into constituents and constituents into sentences represents a semantic context free grammar (CFG). When a regular software developer develops a new application for a new domain, he must define a new grammar by a CFG that is able to define the concepts and their relations of such domain. Even if other approaches suggest learning structures from a set of corpora, this process appears too expensive and probably not exhaustive, my solution can facilitate grammar development by supporting the flow of information from a manually written resource to language contents automatically generated.

The goal is to make sure that the programmer needs only to think about the meaning of a grammar token and not about their content. This new schema, thanks to a Morphological Generator, generates a file that can be reused and edited like a standard phoenix grammar. The new format description is in Fig. 11.

```
Function = NAME_ACTION
{
      [token1] term1 [token2] term2
}
;
```

Fig 11 Smart Grammar syntax.

A set of slots, that represent information that is relevant to the action or object (in this case "NAME_ACTION"), are defined by the "Function" keyword that defines the tag name (Function = NAME_ACTION). The content between curly brackets is described by a new grammar tag definition mode. The number and the order of tokens and terms can change. Each token is written in square brackets.

[22] S. Knight, G. Gorrell, M. Rayner, D. Milward, R. Koeling and I. Lewin, "Comparing grammar-based and robust approaches to speech understanding: a case study", EUROSPEECH 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event, Aalborg, Denmark, September 3-7, 2001, pp. 1779-1782.

[23] SmartGrammar: A dynamic spoken language understanding grammar for inflective languages. Authors: V. Catania, R. Di Natale, A. R. Intilisano, Y. Cilano, D. Panno. Journal: International Journal on Natural Language Computing (IJNLC), June 2014, Vol. 3, No.3. DOI : 10.5121/ijnlc.2014.3301

In such a way, it is no longer necessary to write the word pattern of the token, but only the "keyword name" like [word, characteristic]. The new schema creates a grammar file containing a token and its generated word patterns and it can be reused and edited like a standard Phoenix grammar. Fig. 6 depicts the new format description.

```
Function = SLOT_NAME
{
      [word,characteristic] term1
      [word,characteristic] term2
}
;
```

Figure 12 : Grammar format description

the result can be a not complete grammar. This increases the development time.

Thus, inflected forms add complexity to the Phoenix grammar, since they generate multiple different rules with similar patterns. The development of a new domain application needs a new Context Free Grammar (CFG) that is able to define the concepts and their relations of such domain. The goal is to make sure that the programmer needs only to think about the meaning of a grammar token and not about their content. This new schema, thanks to a Morphological Generator, generates a file that can be reused and edited like a standard phoenix grammar.

The new format description is in Fig. 13.

```
Function = NAME_ACTION
{
      [token1] term1 [token2] term2
}
;
```

Fig 13 New grammar syntax.

A set of slots, that represent information that is relevant to the action or object (in this case "NAME_ACTION"), are defined by the "Function" keyword that defines the tag name (Function = NAME_ACTION). The content between curly brackets is described by a new grammar tag definition mode. The number and the order of tokens and terms can change. Each token is written in square brackets.

In such a way, it is no longer necessary to write the word pattern of the token, but only the "keyword name" like [word, characteristic]. The new schema creates a grammar file containing a token and its generated word patterns and it can be reused and edited like a standard Phoenix grammar. Fig. 12 depicts the new format description.

```
Function = SLOT_NAME
{
      [word,characteristic] term1
      [word,characteristic] term2
}
;
```

Fig 14 New grammar format description.

The grammar slots are defined by the "Function" keyword that defines the slot name (Function = SLOT_NAME). In this way, a tag is defined as a couple "[word, characteristic]" and it is used by the editor to generate the appropriate word patterns according to the characteristic.

The couple [word, characteristic] is defined as below:

If "word" is a verb, "characteristic" can be replaced with:

- "Presente" if the Italian present forms are desired;
- "Passato" if the Italian past forms are desired;
- "Futuro" if the Italian future forms are desired.

If "word" is a noun or an adjective, "characteristic" can be replaced with:

- "Singolare" if the Italian singular forms are desired;
- "Plurale" if the Italian plural forms are desired;

My version of the Morphological Generator generates all new forms specified by the characteristic.

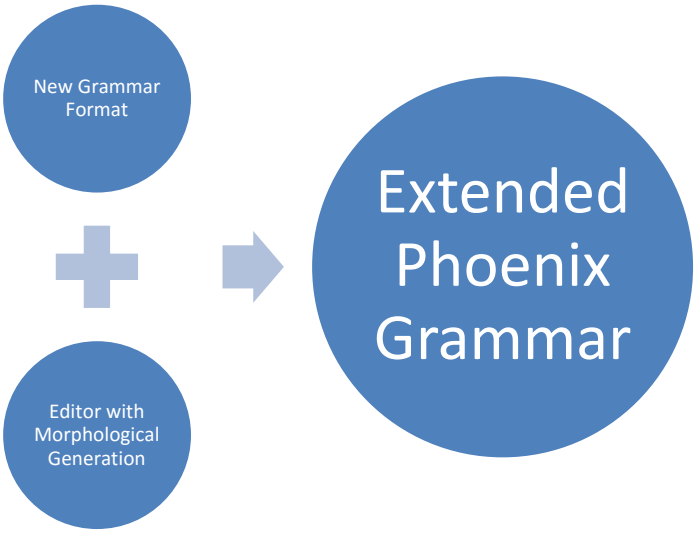

Figure 15 Extend Phoenix Grammar Generation

NEW GRAMMAR GENERATION.

My editor generates an extended standard Phoenix grammar with increased coverage of the new grammar, by performing the following actions:

- Creation of a token named SLOT_NAME in which new tokens and terms are included;
- Creation of a token for each new defined token, in which terms generated by the Morphological Generator are included.

My tool consists of the Editor component, which takes the new grammar as an input and, with the aid of the Morphological Generator, generates the grammar format for Phoenix. The following paragraphs explain in detail the other components.

MORPHOLOGICAL GENERATOR FOR THE ITALIAN LANGUAGE

The Morphological Generator allows you to generate specific inflected and altered forms of nouns, adjectives and verbs. It is a fundamental tool as it allows generating the inflected forms of the language supported. Since each lemma follows different rules for the creation of the inflections, the Morphological Generator uses a word-list in which a grammatical category is associated to each lemma, according to the following format:

- lemma,grammatical_category;

The grammatical category is a string that contains information about the part of speech of the lemma and its way of creating inflections.

For the verbs, there are four grammatical categories:

- One for the intransitive verbs (VI);
- One for transitive verbs (VT);
- One for auxiliary verbs (VA);
- One for modal verbs(VS).

Suffixes for the different conjugation are chosen by analyzing the last three characters with which the verbal lemma ends: these determine the verbal group code. In this way, if the verbal lemma ends in "-are", the suffixes of the first conjugation are applied; if it ends in "-ere" or "-rre", the suffixes of the second conjugation are applied; if it ends in "-ire" suffixes of the third conjugation are applied. If the verb is irregular, the grammatical category contains also an inflectional code that is a number that allows deriving irregular inflections.

There are many grammatical categories of nouns and adjectives. For example, there is a grammatical category of neuter nouns that can generate four different inflectional forms (one of the masculine singular, one of the feminine singular, one of the masculine plural, one of the feminine plural), another of feminine nouns, another of masculine forms, another of neuter nouns that have invariable feminine forms, and so on; similarly, for the adjectives. Inflections are chosen because of the grammatical category and the last characters with which the lemma ends, which determines the noun group code (for nouns) or the adjectival group code (for adjectives). In fact, to each grammatical category of nouns and adjectives some rules are associated. There is also a grammatical category of irregular nouns and one for irregular adjectives; these do not follow rules to create the inflections, so the inflections are obtained from a specific list that contains all irregular forms. For the other parts of speech, there are the following grammatical categories:

- E for prepositions;
- C for conjunctions;
- B for adverbs;
- R for articles;

- P for pronouns;
- For these lemmas, inflections are not applied.

Fig. 15 shows the algorithm; the lemma is the input and the list of all obtained inflected forms is the output. If the lemma is not declinable, the output is the same lemma in input.

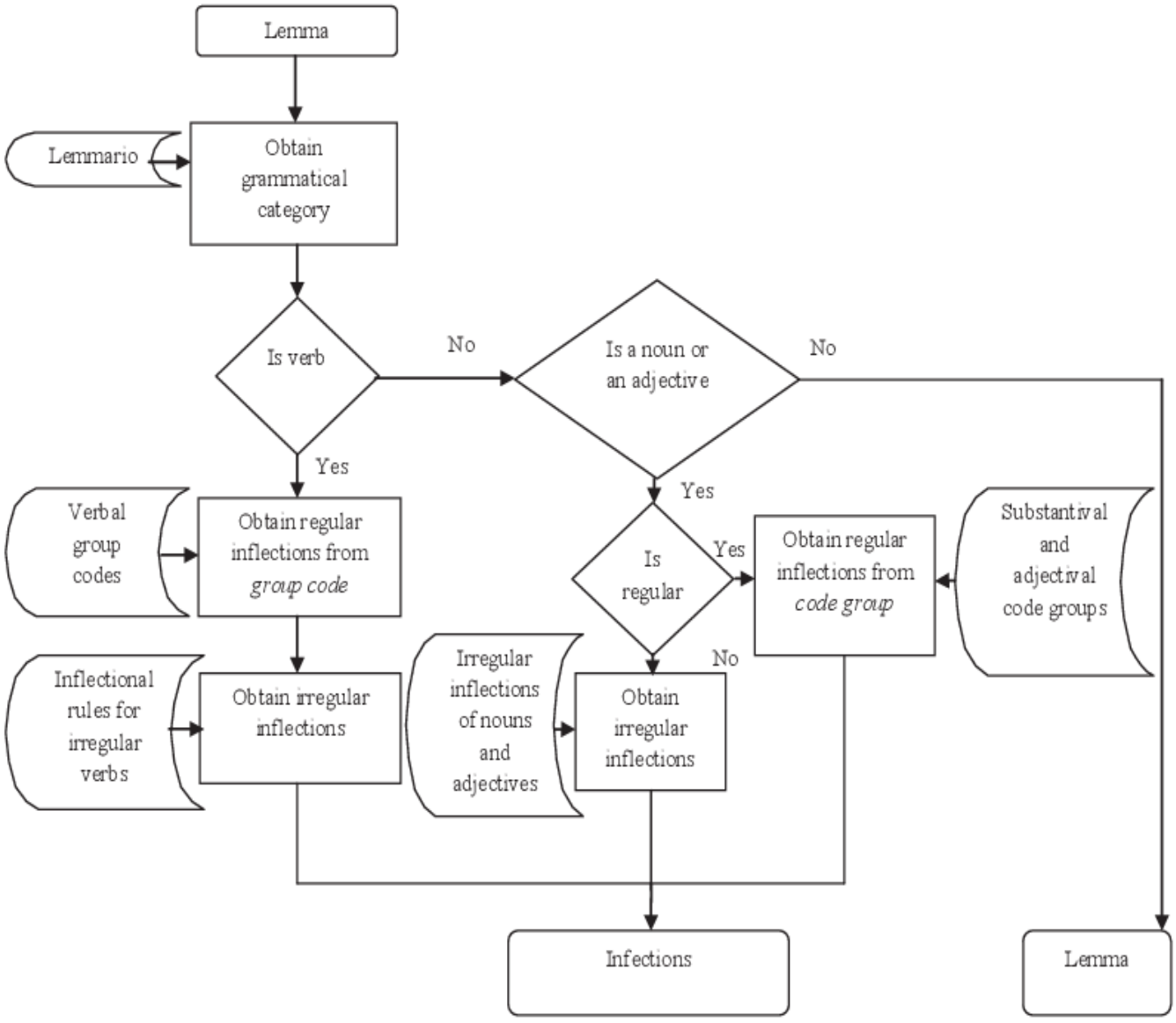

Figure 15 Morphological Generator Algorithm

In Italian, nouns and adjectives can be altered by adding particular suffixes. The alteration modifies the meaning of a word in quantity or quality. The Morphological Generator applies 9 adjectival suffixes for the alteration, each of which can be inflected in gender and number, so in total there are 36 (9x4) possible altered adjectives. There are also 8 substantival suffixes for the alteration, each of which can be inflected, so in total there are 32 (8x2) possible altered nouns. Furthermore, 7 prefixes can be applied to all forms of nouns and adjectives.

When inflectional suffixes are applied, orthographic rules for Italian are respected. The Italian words can be uttered by the combination of many sounds, but sometimes there are not correspondence between the sound and the characters, in fact, some letters have different sounds according to the vowel that follows them. When the suffix of the lemma is removed, the root is obtained and in general, the following rules are applied:

If the root ends in “-c” or “-g”:

If the desinence of the canonical form is “-a”, “-o” or “-u” (forming with the root a hard sound) and the suffix to be applied starts in “e” or “i”, the character “h” is inserted before the suffix.

If the desinence of the canonical form is “-e” or “-i” (forming with the root a soft sound) and the suffix to be applied starts in “a”, “o” or “u”, the character “i” is inserted before the suffix.

- The vowel “i” is removed from the root if:
- The root ends in “-ci” or “-gi” and the suffix starts in “e”;
- The root ends in “-i” and the suffix starts in “i”.

There are also particular words that not follow these rules. In these cases, the words belong to a particular grammatical category that nullifies the rules above. Furthermore, there are particular orthographic rules for verbs.

Each generated word is stored in a structure that saves information about the inflection: part of speech, mood, temp, gender, number, suffix applied and prefix applied. Therefore, the algorithm is able to give in output only the inflections of a lemma required by the user (for example, all the past tenses of a verb or

the singular forms of a noun or an adjective). This characteristic is used for the generation of the new grammar.

This method can apply not only to the Romance languages. It can be applied to others inflective languages. For example, many Morphological Generators [24][25][26] one for a given language, can be utilized and the editor can generate many Phoenix grammar files, one for each language. In this way, the developer writes the grammar in a single language and obtains a multilingual result.

[24] Anandan, P., Ranjani Parthasarathy & Geetha, T.V., "Morphological Generator for Tamil," Tamil Internet Conference, Kuala Lumpur, Malaysia, 2001.

[25] George Petasis, Vangelis Karkaletsis, Dimitra Farmakiotou, George Samaritakis, Ion Androutsopoulos, Constantine D. Spyropoulos , "A Greek Morphological Lexicon And Its Exploitation By A Greek Controlled Language Checker," In Proceedings of the 8th Panhellenic Conference on Informatics, pp. 8 – 10, 2001.

[26] Habash, N., Ower, R., and George, "Morphological analysis and generation for Arabic dialects," Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages, pages 17–24, Ann Arbor, June 2005.

GRAMMAR EDITOR

The grammar editor (Fig. 15) consists of a text editor modified for my purposes. This editor supports the new grammar format and the user produces the corresponding .gra file, by clicking on the "generate" button.

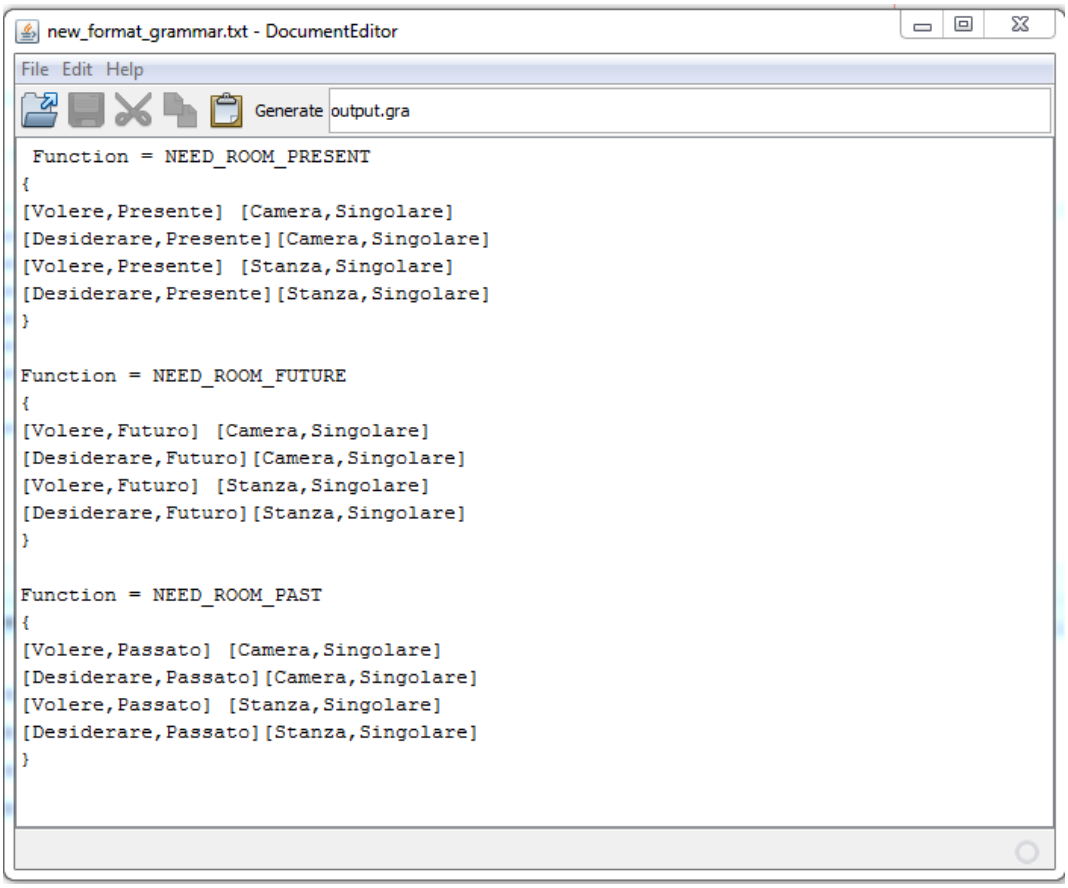

Figure 7 GUI of the editor.

This component reads and processes the grammar files (new format) and, using the **Morphological Generator**, obtains a Phoenix grammar file (Fig. 16).

Figure 8 : How Editor works

If the programmer does not know the SDS domain language, he enables the "translator" module (Fig. 17) between the Morphological Generator and the Editor. For example, an English language grammar, as shown in Fig. 16, is translated by a component of the Editor into the target language and then is used by the Morphological Generator to generate the grammar of the target language in the Phoenix grammar format.

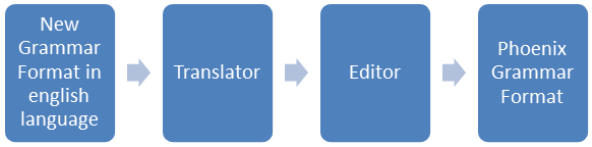

Figure 9 New grammar generation process with translator.

In this way, we have a grammar written in a universal language (English) with a high level of abstraction that can generate more coverage of a grammar written by a programmer in the same time. In addition, Phoenix grammars in different languages that are not initially known by the regular software developer. Each grammar that is produced requires a different morphological generator.

```
Function = NEED_ROOM_PRESENT
{
[Will, Present] [Room, Singular]
[Want, Present] [Room, Singular]
[Will, Present] [Room, Singular]
[Want, Present] [Room, Singular]
}
```

Figure 10 New grammar written in English language

The complexity of the grammar of Italian and Neo-Latin languages, in general, increases the effort in developing an efficient SLU grammar for a SDS. With this system, the regular software developer can generate a Phoenix grammar without worrying about all the possible variations, conjugations and alterations of words that are characteristics of the Romance languages

EXPERIMENTAL RESULTS

An example is reported to show the advantages brought by this approach. It shows a Phoenix grammar of a real SDS for a room-reservation application, based on the Olympus Framework. In a typical interaction, the user can express the same concept using a specific word, but in different tenses. For example, “I want a room” in Italian can be expressed like “Voglio una camera”, but also “Vorrei una camera” (“I'd like to have a room”) or “Vorrei una cameretta” (“I'd like to have a small room”, in Italian it is a term of endearment). Fig. 19 shows an example of grammar:

```
Function=NEED_ROOM_PRESENT
{
      [volere,presente][camera,singolare]
      [desiderare,presente][camera,singolare]
      [volere,presente][stanza,singolare]
      [desiderare,presente][stanza,singolare]
}

Function=NEED_ROOM_FUTURE
{
      [volere,futuro][camera,singolare]
      [desiderare,futuro][camera,singolare]
      [volere,futuro][stanza,singolare]
      [desiderare,futuro][stanza,singolare]
}
```

Figure 11 Example of Italian grammar

The new grammar consists of two parts. The first one, shown in Fig. 12, represents the definition of a grammar slot.

```
[NEED_ROOM_PRESENT]
      [volere_presente][camera_singolare]
      [desiderare_presente][camera_singolare]
      [volere_presente][stanza_singolare]
      [desiderare_presente][stanza_singolare]
;

[NEED_ROOM_FUTURE]
      [volere_futuro][camera_singolare]
      [desiderare_futuro][camera_singolare]
      [volere_futuro][stanza_singolare]
      [desiderare_futuro][stanza_singolare]
;
```

Figure 12 Generated Grammar Slot

The second part, shown in Fig. 20, defines each token, including their word patterns. A more detailed explanation is along with the resource code (output.gra file)[27]. The initial grammar, consisting of 21 rows,

[27] Source code, http://opensource.diit.unict.it/vctsds/GrammarEditor.zip (last visited: 22 November 2013).

generates a 140-row-long Phoenix grammar that allows the SLU module to recognize a large set of utterances. This way, the developer focuses his attention on the meaning of an intermediate-grammar token and not on its content.

```
#tag auto generated
[volere_presente]
      (voglio)
      (vuoi)
      ...
;

#tag auto generated
[volere_futuro]
      (vorrò)
      (vorrai)
      ...
;

#tag auto generated
[stanza_singolare]
      (stanza)
      (stanzaccia)
      ...
;

#tag auto generated
[camera_singolare]
      (camera)
      (cameraccia)
      ...
;

#tag auto generated
[desiderare_presente]
      (desidero)
      (desideri)
      ...
;

#tag auto generated
[desiderare_futuro]
      (desidererò)
      (desidererai)
      ...
;
```

Figure 13 Generated Token

Furthermore, the developer does not need to write all possible forms (mood, tense, person, etc.), some of which could be difficult to predict. The advantage of the generated grammar is the ability to easily simulate and predict the large variety of interactions that can occur.

The same grammar can also be obtained starting from an initial grammar written in another language, for example, in English, and enabling the translator module, as shown in Fig. 22.

```
Function=NEED_ROOM_PRESENT
{
      [to want,present][room,singular]
      [to desire,present][room,singular]
      [to want,present][apartment,singular]
      [to desire,present][apartment,singular]
}

Function=NEED_ROOM_FUTURE
{
      [to want,future][room,singular]
      [to desire,future][room,singular]
      [to want,future][apartment,singular]
      [to desire,future][apartment,singular]
}
```

Figure 14 Example of English grammar

The generated grammar slots are shown in Fig. 24 and the associated tokens in Fig. 23.

```
[NEED_ROOM_PRESENT]
      [to_want_present][room_singular]
      [to_desire_present][room_singular]
      [to_want_present][apartment_singular]
      [to_desire_present][apartment_singular]
;

[NEED_ROOM_FUTURE]
      [to_want_future][room_singular]
      [to_desire_future][room_singular]
      [to_want_future][apartment_singular]
      [to_desire_future][apartment_singular]
;
```

Figure 15 Generated grammar slots from English

```
#tag auto generated
[to_want_present]
      (voglio)
      (vuoi)
      ...
;

#tag auto generated
[room_singular]
      (stanza)
      (stanzaccia)
      ...
;

#tag auto generated
[desire_present]
      (desidero)
      (desideri)
      ...
;
```

```
#tag auto generated
[to_want_future]
      (vorrò)
      (vorrai)
      ...
;

#tag auto generated
[apartment_singular]
      (camera)
      (cameraccia)
      ...
;

#tag auto generated
[desire_future]
      (desidererò)
      (desidererai)
      ...
;
```

Figure 16 Generated tokens from English

One of the major obstacles in the development of a spoken dialog application is the writing of a domain-specific grammar. The development of a grammar is a very complex task when the used language is inflective [28] [29]. In general, the grammar is updated in a semi-automatic and unsupervised way by the automatic retrieving of the relations between meaning and user utterances.

My proposed method is able to discover a relationship between an expression of the user and its meaning and to update the grammar in a semi-automatic and unsupervised way.

The web represents a large corpus of information which can be used to extract knowledge and automate the association of a word with its meaning. In this context, the web can be considered as corpus that is continuously updated and from which it is possible to retrieve knowledge to define a grammar in a semi-static way. We introduce an unsupervised approach for the SDS domain based on the PANKOW method[30] (Pattern based Annotation through knowledge on the Web) to use knowledge from the web to update the CFG in grammar files. Thanks to this method an SDS can recognize utterances not included in the grammar initially. My technique uses these words to associate words with their respective meanings. The PANKOW method uses the Google Search Engine to retrieve information from the web and it requires a list of meanings that the Google Search Engine uses in its queries. In particular, the choice of the pair word-meaning will be made according to the confidence level of the relations derived from the Google Search Engine results. Furthermore, the system can retrieve information from Facebook through Google to increase its list of names or surnames.

For the SDS domain, the problem described above has been addressed in[31]. In particular, that paper shows how to implement an incremental On2L ontology to obtain knowledge at runtime. That system for semantic annotation is based on models of the natural language to extract the relations between a term and

[28] V. Catania, R. Di Natale, A. R. Intilisano, Y. Cilano, D. Panno. "SmartGrammar: A dynamic spoken language understanding grammar for inflective languages"

[29] S. M. Biondi, V. Catania, Y. Cilano, R. Di Natale and A.R. Intilisano. 2014. An Easy and Efficient Grammar Generator for Spoken Language Understanding, The Sixth International Conference on Creative Content Technologies – Vol. 7 nr 1 and 2, Venice, Italy. [

[30] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab. 2004. Towards the self-annotating web. In Proceedings of the 13th international conference on World Wide Web (WWW '04). ACM, New York, NY, USA, 462-471

[31] Berenike loos. 2006. On2L - A Framework for Incremental Ontology Learning in Spoken Dialog Systems. In Proceedings of the COLING/ACL 2006 Student Research Workshop, Sydney, Australia.

its corresponding concept in the ontology. All unrecognized words are classified as out-of-vocabulary (OOV). During my PhD course i show the advantage of making a distinction between global OOVs and local OOVs. Global OOVs include, for example, proper nouns of stars, movies, hotels, while local OOVs include proper nouns or names of countries or cities. In the first case, a search on Wikipedia is performed; in the second case, the Google Search Engine is used. To be more precise, if the OOV typology is unknown, the system performs a search on Wikipedia as a first step, and then it uses Google Search. If the search on Wikipedia produces results, these are used as a test for Google Search; otherwise, a search with Google Engine is performed. However, that should be considered as a work in progress and it has not been evaluated in detail so far. Its aim was to provide a task-oriented evaluation setup and to improve the results by using different techniques.

Different fields exploit semantic annotation to retrieve information from web. For example, it is used in Biomedical Literature to implement an automatic extraction of biomedical information[32].

PANKOW METHOD IN SDS DOMAIN APPLICATION

The web contains millions of documents, forming a big corpus that can be used as a resource of information in the development of an auto-updating grammar. The core of this application is based on PANKOW, a self-annotating method that uses linguistic patterns to annotate the meaning of a word by means of the Web. The PANKOW method is an unsupervised pattern-based approach that categorizes instances according to a series of concepts. The self-annotating method is based on a series of Linguistic Patterns such as the **HERST pattern**[33], that create regular expressions in order to identify instance-concept relations in a text. In this section we describe the major aspects of the PANKOW method and the way it has been adapted to my application. The PANKOW method uses a list of proper nouns as inputs. Every word uttered by the user is analyzed by the self-annotating method at runtime. Therefore, the user utterance represents the new input to the PANKOW method. Furthermore, the PANKOW method initially requires a series of concepts along with a series of linguistic patterns to perform self-annotation using the Web corpus. These concepts, in my domain, represent the meanings that the words can assume. The

[32] Rune Sætre, Amund Tveit, Tonje S. Steigedal, and Astrid Lægreid. 2005. Semantic annotation of biomedical literature using google. In Proceedings of the 2005 international conference on Computational Science and Its Applications - Volume Part III (ICCSA'05), Osvaldo Gervasi, Marina L. Gavrilova, Vipin Kumar, Antonio Laganà, and Heow Pueh Lee (Eds.), Vol. Part III. Springer-Verlag, Berlin, Heidelberg, 327337.

[33] Marti A. Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In Proceedings of the 14th conference on Computational linguistics - Volume 2 (COLING '92), Vol. 2. Association for Computational Linguistics, Stroudsburg, PA, USA, 539-545.

second step consists in deriving phrases by introducing an instance (word) and ontological concepts (meaning) potentially related to it in the linguistic patterns. For example:

<INSTANCE|WORD> IS A <CONCEPT | MEANING>

If the user utters the word “Italy” and “country” and “hotel” are the candidate meanings, the phrases derived by using the linguistic pattern “IS A“ are: “Italy is a country” and “Italy is a hotel”.

The second step ends when all possible phrases are derived, and these become the inputs for the third step, in which these are used as queries to the Google Search Engine. The Google Engine is called through its API and the retrieved results represent the number of hits for each phrase. Then, according to the results obtained, the service selects the meaning (among those in the list) to be associated with the utterance provided by the user. The following formula allows to retrieve the number of occurrences of an instance i of a concept c for all patterns p belonging to the set P.

$$count(i, c) = \sum_{p \in P} f(i, c, p)$$

In the literature, there are several approaches for analyzing a text corpus. They capture specific information or relations between words and concepts. HEARST describes an automatic acquisition method of hyponymy lexical relations from unrestricted text. HEARST identifies a set of lexical-syntactic patterns that occur frequently in a text corpus. These may in fact indicate lexical relations that could be of interest.

Linguistic patterns[34] are formulated using the variable '<INSTANCE>' to refer to a candidate noun phrase, such as the name of an ontological instance, while the variable '<CONCEPT>' refers to the name of a given concept. The HEARST patterns used in the PANKOW method are the following:

- <CONCEPT>s such as <INSTANCE>
- such <CONCEPT>s as <INSTANCE>
- <CONCEPT>s, (especially|including)<INSTANCE>
- <INSTANCE> (and|or) other <CONCEPT>s

The verb "to be" defines an entity as an instance of a concept. A possible linguistic pattern is:

<INSTANCE> is a <CONCEPT>

My method uses these patterns to realize an automatic unsupervised approach for discovering lexical relations between the utterance of a user and its meaning using the Web.

[34] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab. 2004. Towards the self-annotating web. In Proceedings of the 13th international conference on World Wide Web (WWW '04). ACM, New York, NY, USA, 462-471.

UNSUPERVISED SELF LEARNING GRAMMAR METHOD USING OLYMPUS FRAMEWORK.

The self-learning grammar system being proposed uses a revised version of the PANKOW method that is used to improve the performance of a SLU module. As mentioned in previous chapters, a specific Spoken Language Application requires the development a context-free grammar. According to the described grammar, the SDS system can understand the utterances pronounced by a user. The Phoenix grammar file definition comes in the form of a slot definition. A slot is a symbol that captures a subset of the language, such as "place names" or "country" of “surname”. The general structure of a slot definition is as follows:

```
[Country]
        (Italy)
        (South Africa)
        (Congo)
        (Guatemala)
;
```

The goal of the system is **to simplify the grammar development process**, automatically detecting the relations between utterances and meanings and it does not require pre-definite knowledge of the meaning of a word. It works by merging the Phoenix parser with the PANKOW service. Therefore, the slot definition changes according to the service features. For example, the grammar slot “country”, represents an ontological concept candidate of PANKOW, and it is also the meaning that the system tries to associate to the user utterances “Italy” and “South Africa”. If the user utters “United States”, the SLU parser will not recognize this word as a "country" meaning, because it is not included in the grammar slot definition. In fact, Phoenix does not recognize words not included in the grammar slot definition. Including all possible words for each concept is a very expensive process. My system, thanks to the Unsupervised Self learning grammar method, only requires the definition of the symbol of the grammar slot like “country”,

and it represents the meaning of a specific list of utterances. The meaning of the word "country" is retrieved from the web using the PANKOW method. This method attempts to discover if a linguistic relation exists between “country” (meaning) and a specific word ("United State"). The new grammar slot definition is shown below:

[Country]

(Italy)

(South Africa)

(Congo)

(Guatemala)

Africa (auto-generated by my method)

United States (auto-generated by my method)

Australia (auto-generated by my method)

;

The PANKOW method updates the old grammar file, adding the instances found. When the Phoenix parser does not recognize a word, it will become a candidate for my unsupervised method. The scheme of the described system is shown in Fig. 25. A series of tests were performed to show the improvements

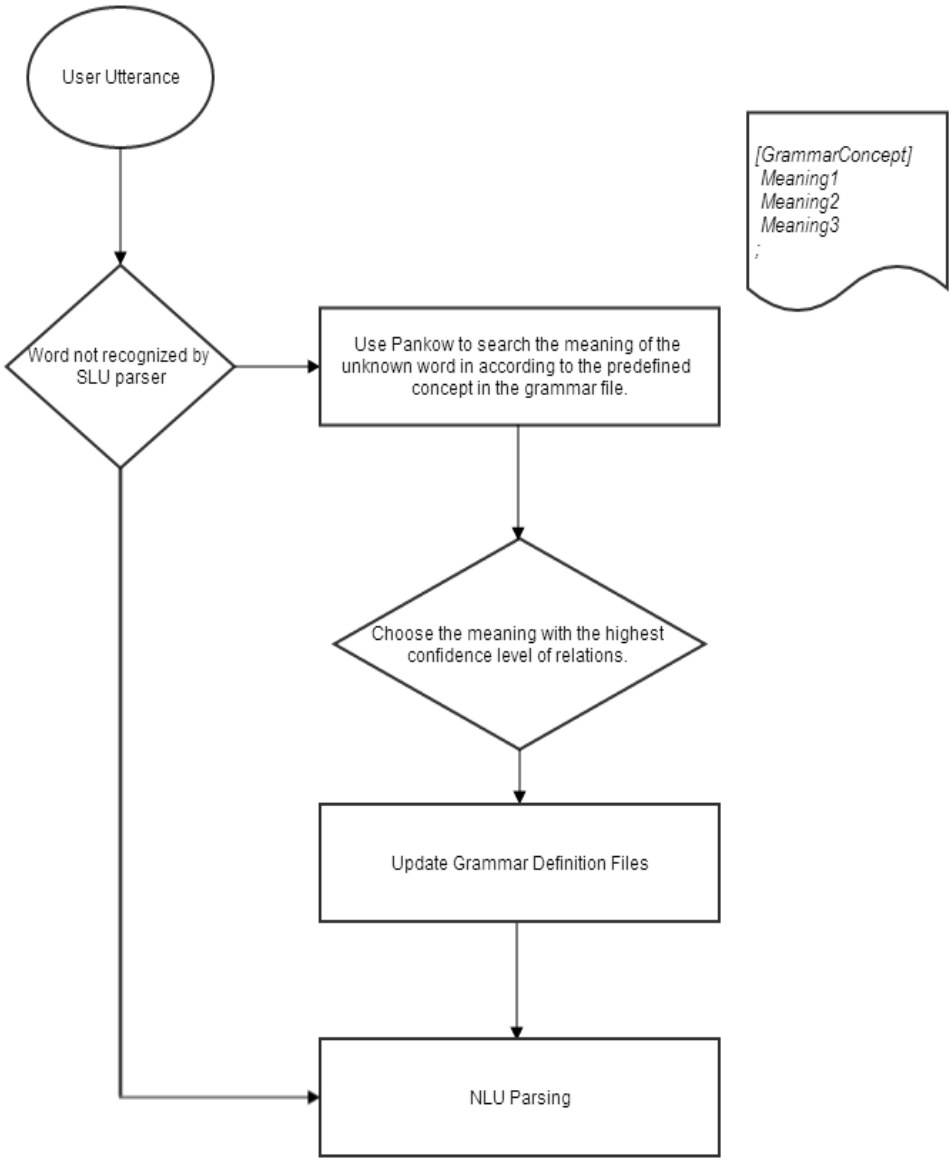

Figure 17 A self-learning Grammar System Schema

TEST AND RESULT

This section shows the result of the performed tests. Initially a grammar file was created as defined by "new grammar slot definition", with the choice of a series of Concepts. A list of random items is the input to the unsupervised method. Below, we show the concepts used in our method for the tests:

Concept (Meanings)

- Name
- Surname
- Smartphone
- Country

The input files contain a list of (Name, Surname, Smartphone and Country). For each input term, the unsupervised method tries to find a linguistic relation between the input item and all concepts. Thus, the Google Search Engine returns the number of the f(c,i,p) (Concept, Instance, Pattern) occurrences.

The number of occurrences generated by Google Search determines the confidence level of the relations between an instance and a concept, as shown in Table I. The results show a high percentage of hits of concepts like "country" and "smartphone". Although name and surname show smaller values, their percentages of hits are still relatively high. "Name" and "Surname" represent concepts with a larger probability of generating ambiguities than "smartphone" or "country", as these are generally more specific terms.

TABLE I

RESULT TEST UNSUPERVISED METHOD

Meaning	Number of instance	Hit percentage
Name	50	58%
Surname	50	60%
Country	50	94%
Smartphone	50	94%

We introduce a service to retrieve information about the concepts “name” and “surname”. Since these concepts are strictly related to one or more individuals, social networks such as Facebook may constitute an important source of information.

If a user queries Google Search with the name and surname of an individual, the first results are usually related to Facebook. Therefore, our method questions the search engine with queries like “Name Facebook” or “Surname Facebook”. This is used verify the existence of an individual that possesses a certain name or surname. Usually an individual's Facebook profile contains first name, other name and Surname. The method captures the meaning of the words, relying on their positions. In particular, the system uses the first 10 results of the Google Search Engine. The results are shown in Table II.

TABLE II

RESULT TEST UNSUPERVISED METHOD

Meaning	Number of instance	Hit percentage
Name	50	44%
Surname	50	88%

ACOUSTIC MODEL FOR ITALIAN LANGUAGE

During my work to SLU I worked parallel to the ASR. My work aims to solve the lack of an acoustic model suitable for the Italian language. I then developed a method to automate the developer of the acoustic model for the Italian language. I then expanded the work into other languages.

Automatic Speech Recognition (ASR) maps an acoustic signal into a sequence of phonemes or words (discrete entities). A typical ASR process performs the decoding of input speech data by means of signal processing techniques and acoustic or language models. Acoustic models refer to the representation of acoustics and phonetics, while language models describe the words that are recognized by the ASR process. The complexity of natural language makes its use for human – computer interaction a difficult task to perform[35]. For example, a specific context may influence the generation of phonemes or speech signal can vary significantly according to speaker sex, style, speed and can be affected by noise. Speech recognition engines require statistical representation of each of the distinct sounds that makes up a word (Acoustic model). This model is created by a training process that compiles recordings and their transcriptions into a statistical representation of the sounds for each word. Therefore, large amounts of transcribed recordings are generally required. Obtaining these data is an expensive and time-consuming task: several hours of recordings are necessary, recorded in a quiet environment and by different voices. Finding all these data with the relative transcription is not trivial. An alternative to build from scratch a database with audio recordings and manual transcriptions is the use of audiobooks.

HOW TO CREATE AN ACOUSTIC MODEL

Different approaches are available to generate an audio corpora database to be used for creating an acoustic model: a first approach[36] that uses speakers with different age and gender for providing recordings related to a predefined text; a second approach that uses available audio resources (radio broadcast, news broadcasts) and the corresponding text transcriptions [37] [38].However, the first method is accurate but time- and resource-consuming, while the second one is easy to develop but it often

[35] M. Forsberg, "Why is speech recognition difficult?" Technical Report from Department of Computing Science Chalmers University of Technology, Sweden, 2003.

[36] VoxForge, http://www.voxforge.org (last visited March 5, 2014).

[37] L. Lamel, J. Gauvain, and G. Adda, "Lightly supervised and unsupervised acoustic model training", Computer Speech & Language, Spoken Language Processing Group, CNRS-LIMSI, Orsay, France, Volume 16 Issue 1, 2002, pp. 115-129.

[38] C. Cieri, D. Graff, and M. Liberman, “The TDT-2 Text and Speech Corpus” in In Proceedings of Darpa Broadcast News Workshop, 1999, pp. 57–60.

provides incomplete results because most of its audio data resources have no corresponding accurate word transcriptions. For example, a complete commercial recording database of Italian language records already exists and is called APASCI [39]. This is an Italian speech database designed for researching on acoustic modeling. The process to create a database using these features from scratch requires much time. Another available solution is Voxforge [40], a free project that collects corpora of spoken speeches in several languages thanks to the collaboration of users who provide their voices. All audio files are available under the General Public License (GPL) license and allow obtaining acoustic models for many speech recognition engines such as Carnegie Mellon University (CMU) Sphinx and Hidden Markov Model Toolkit (HTK) [41]. Sphinx does not provide an Italian acoustic model, thus, it has to be created with a suitable audio database. Besides Voxforge, no Italian free audio databases, which work with the aforementioned engine, were created. At the time of writing this paper Voxforge contains about 11 hours of recordings for Italian language. Such a small amount of records does not allow to obtain a good acoustic model using Sphinx, because to create a new model 50 hours of audio recordings of 200 different speakers are required [42]. So, the need of a specific audio database for creating an Italian language acoustic model with Sphinx has arisen.

A NOVEL APPROACH

As described above, the acoustic model is a building block of an ASR system and it can manage a specific language only if an acoustic model for that language is available. In this paper, a novel approach for creating an acoustic and linguistic model for Sphinx is described. It provides a method to obtain in a simple and fast way many transcribed recordings from audiobooks. Audiobooks are a valid and reliable resource to create acoustic models because they are recorded in an echoic chamber by different voices and the text of the audiobook is available. Audiobooks are a good and free statistical basis in terms of cost / time, but the problem is that, in general, the training for creating an acoustic model requires small audio files in terms of duration (for example the optimal length for Sphinx is not less than 5 seconds and not more than 30 seconds). Because audiobooks provide audio files that are too long for Sphinx, a method to split audiobooks and to associate to each part obtained the corresponding transcription is necessary. The tool HTK was used for this purpose. To solve the problem for the creation of acoustic models is to provide the Italian phonetic transcription of each

[39] APASCI, http://catalog.elra.info/product_info.php?products_id=16 8 (last visited February 25, 2014).

[40] VoxForge repository, http://www.repository.voxforge1.org/downloads/it/Trunk /Audio/ (last visited March 5, 2014).

[41] HTK Documentation, http://htk.eng.cam.ac.uk/docs/docs.shtml (last visited February 25, 2014).

[42] Training Acoustic Model for CMUSphinx, http://cmusphinx.sourceforge.net/wiki/tutorialam (last visited March 3, 2014).

word in the audio files, an algorithm that creates phonetic transcriptions has been developed. For the training of the Italian acoustic model we used SphinxTrain (it is the tool used for the training of an acoustic model for Sphinx). **This method can be applied for the creation of acoustic models in several languages.**

TRAINING BY AUDIOBOOK

Audiobooks are usually available like a single long audio file with the corresponding text transcription. For my research we created an automatic method to develop a training set for Sphinxtrain from audiobooks. So, this method splits a single audiobook in several little audio files with the corresponding text transcription. These audio files make the Audio Database required from sphinx to extract statistic from the speech. HTK toolkit[43] was used to split the audiobooks. HTK [44] requires a little acoustic model to perform the work described above. This acoustic model was developed by using VoxForge training through a set of collected transcribed speech corpora. During tests, VoxForge held a database with about 11 hours recording with related text transcriptions. Usually an audio database with 11 hours of

[43] HTK Toolkit and tutorial, http://www.voxforge.org/home/dev/autoaudioseg (last visited March 3, 2014).

[44] HTK JuliusTutorial, http://www.voxforge.org/home/dev/acousticmodels/linux /create/htkjulius/tutorial (last visited March 4, 2014).

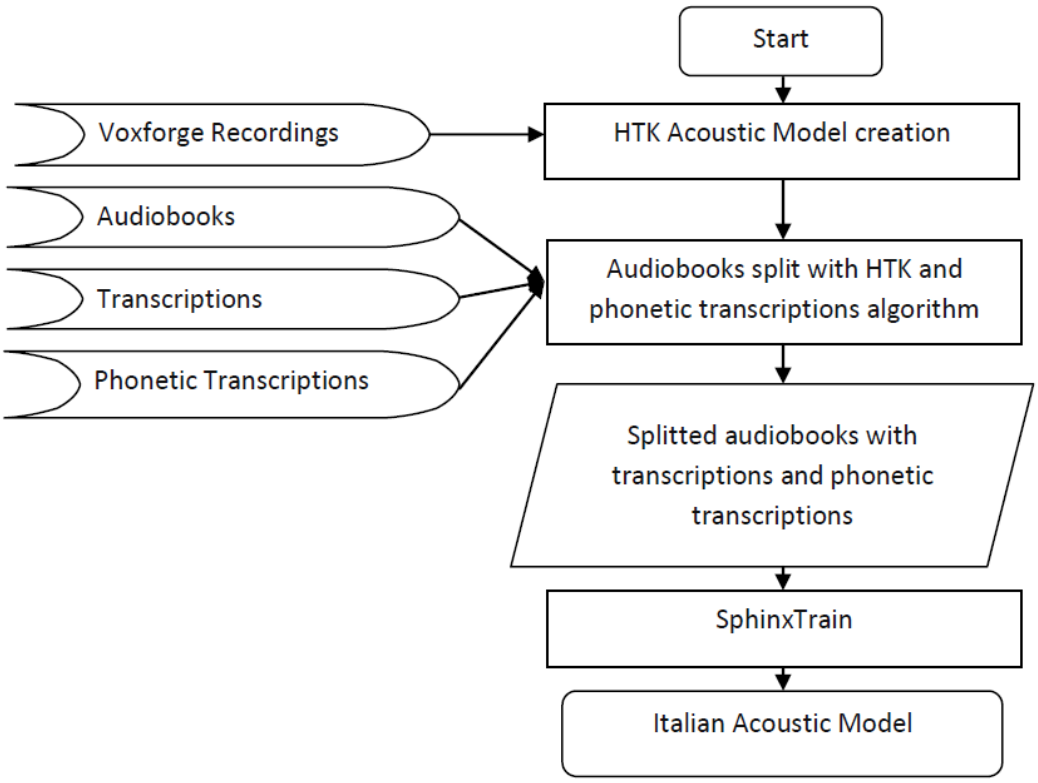

Figure 18 Procedure of training

recording is not enough to create an efficient acoustic model, but it was sufficient to use in my automatic split method through HTK. My test shows that this database is a valid choice for my task. In fact, we use this first model to split the selected audiobooks because it allows to align the audio to its transcription. Next, the phonetic transcriptions of the words must be provided: for this purpose, the algorithm described in Chapter 5 has been developed. Finally, audio recordings obtained by the split audiobooks and their phonetic transcriptions are input for SphinxTrain that produces an Italian acoustic model. Fig. 27 shows the whole procedure.

PHONETIC DICTIONARY GENERATION

The training function must be configured with the sound units, the corresponding transcriptions and the phonetic dictionary. It maps every word into a sequence of sound units (phonetic transcription). To derive the sequence of sound units the phonetic transcriptions are associated with each word:

- ABACO a b a k o
- ABACHI a b a k I

In the example above, the word ABACO is the Italian for “Abacus”, while ABACHI is its plural. At their right there is the corresponding phonetic translation.

It initially uses an existing dictionary that consists in a customized version of the Festival [45]lexicon (brackets and number were removed). It includes more than 440,000 words, but it is not complete enough to perform the phonetic transcriptions of all the words present in the split audio files. Accents are important for Italian language because the meaning of the word may change based on their position. So, an algorithm to derive the missing phonetic transcriptions is necessary. The developed algorithm is able to derive two different phonetic transcriptions. The input of the algorithm consists in a list of words not found in the Festival lexicon and for each word it performs the following steps:

- [optional] an online search of the word in an Italian dictionary-database [46]: if the word is found in the dictionary the position of the accent is obtained. In fact, if the word is a lemma (or entry-word) missing in the phonetic dictionary, the algorithm generates the phonetic transcription with the correct accent. This step is optional since calling the web service takes a considerable time. Furthermore, grandidizionari.it service does not contain all the necessary words, so:
- If the accent was not obtained, the lemma from which the word is derived is looked up in the dictionary of Morph-IT! [47], that is a lexicon of inflected forms with their lemma and morphological features. Output of this module is a tuple composed by:

- Form
- Lemma
- Features

At this point, the substring with which the word ends are compared with a list of available desinences written according to the following format:

- vowel_or_consonant+desinence,category, part_of_speech,phonetical_transcription

[45] Festival website, http://festvox.org/ (last visited March 3, 2014)

[46] Grandizionari online service, http://www.grandidizionari.it/Dizionario_Italiano.aspx?i dD=1 (last visited March 3, 2014).

[47] E. Zanchetta and M. Baroni, "Morph-it! A free corpusbased morphological resource for the Italian language" , proceedings of Corpus Linguistics 2005, University of Birmingham, Birmingham, UK.

- vowel_or_consonant indicates if "desinence" must be preceded by a vowel, a consonant, or both. This parameter is optional and can take 3 values: V (vowel), C (consonant), VC (vowel plus consonant).
- desinence is the analyzed desinence, that is the string compared with the substring with which the word ends. The desinences have been obtained from [48]. For each desinence, its inflected versions are obtained by analyzing the category.
- category is used to get the inflections of each desinence. A letter represents each category. There is a list of categories in which each category is associated with some characteristics. Each characteristic is written according to the following format:
- category: s1,1-pt1,1 > si1,1-pti1,1,…, si1,n-pti1,n; … ; sk,1-ptk,1 > sik,1-ptik,1,…, sik,n-ptik,n;

(where s=substring, pt=phonetic transcription, i=inflected).

In general, if a desinence belongs to a certain category and it ends in sk,1, to obtain the inflected form, sk,1 is removed from the desinence and it is replaced with sik,1, ptk,1 is removed from its phonetic transcription and it is replaced with ptik,1. This proceeding is applied for all eventual n inflections.

For example, consider the "D" category that is written as:

- D:gia-dZ i! a>gie-dZ i! e;cia-tS i! a>cie-tS i! e;

[48] Desinences list, http://www.dipionline.it/guida/docs/DiPI_Terminazioni_ Desinenze.pdf (last visited March 3, 2014).

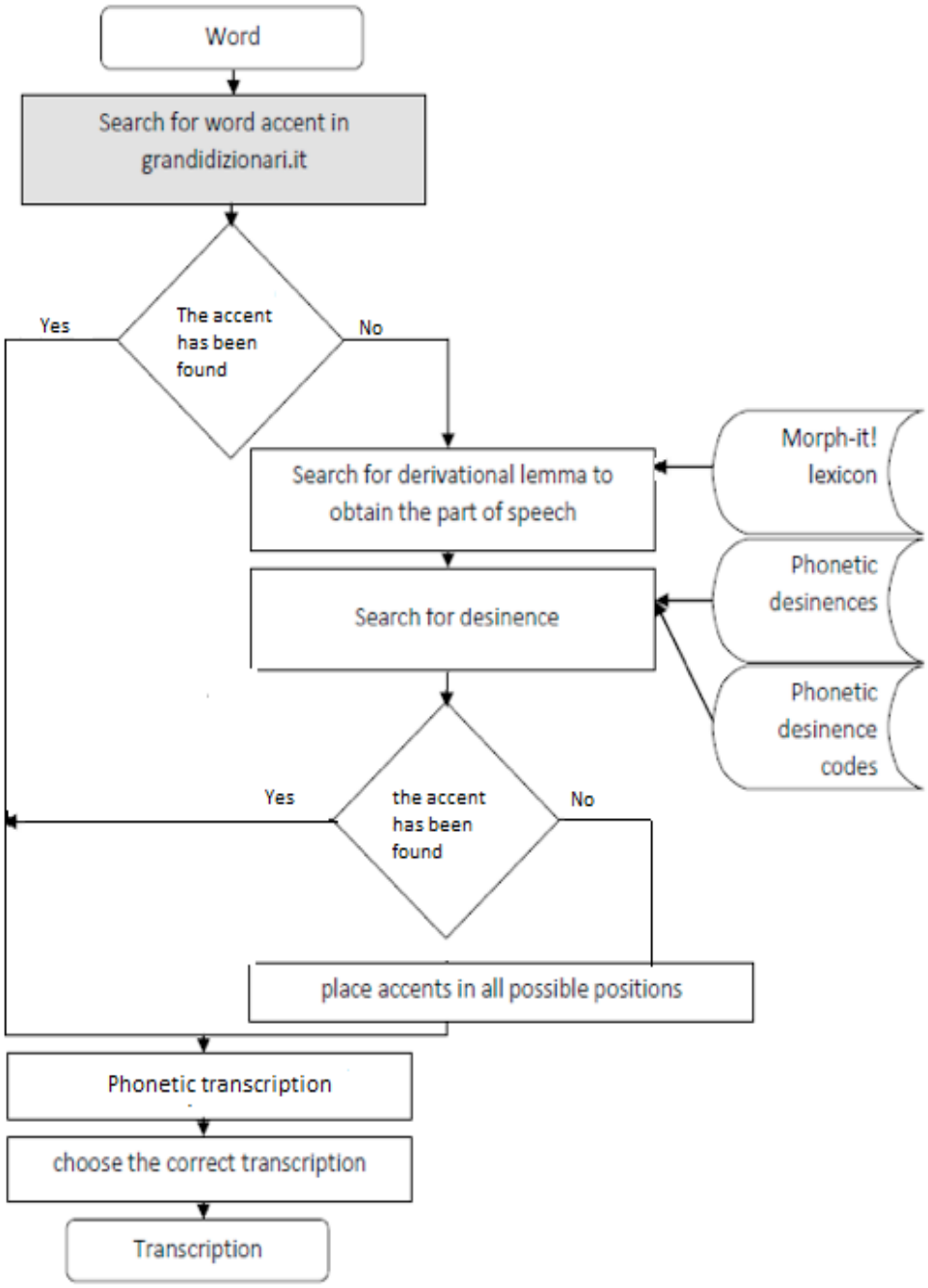

Figure 19 Algorithm for phonetic transcriptions, in gray the optional step

This means that if a desinence belongs to the category D, if it ends in "-gia" and the correspondent phonetic transcription ends in "-dZ i! a", the inflected form is obtained by removing "-gia" and adding "-gie " in the desinence and by eliminating " dZ-i! a" and adding "-dZ i! e" in the phonetic transcription.

There is also the "I" category for desinences that do not have inflected forms: I:;

- part_of_speech indicates the part of speech and is used because some equal desinences endings have different accents emphasis depending on the part of speech. The used parts of speech are: V for verbs, N for nouns and adjectives, A for other;
- phonetical_transcription is the phonetic transcription of the desinence.

If no desinence has been identified and the word does not have an accent, a set of strings is created. This set includes all possible inflected forms for the analyzed word. Transcriptions are created for each of these forms and the user can choose the correct one. The entire procedure is shown in figure 27. For the Italian language, the Italian phonemes are used [49] with the following simplification: close-mid front unrounded vowels and open-mid front unrounded vowels are mapped as a single phoneme "e". The same simplification is applied to the close-mid back rounded vowels and the open-mid back rounded vowels, mapped in a single phoneme "o". The algorithm provides the phonetic transcription of a word both with accents and without (setting by the software), and the user can choose which version to take into account. The word to be phonetically transcribed is analyzed character by character. Optionally, the software cannot take into account the position of accent and therefore does not generate a phonetic transcription that includes the accent, skipping the entire branch "no" in Figure 2 of the first if statement (including grandidizionari.it service).

EXPERIMENTAL RESULTS

The goal of this test was not to validate the Sphinx acoustic model we developed for the Italian language, since no other models were available for comparison. Instead, the purpose of the test was to demonstrate that the approach based on audiobooks can deliver a valid acoustic model. We expect that increasing the size of the training set should provide higher performance. A first acoustic model with HTK has been created by using Italian VoxForge database. The splitting algorithm has been applied to the free audiobooks [50]. We obtained about 34 hmys of split recordings, 56% of them being spoken by female voices. Audiobooks were read by six different male voices and two different female voices. The automatic phonetic generation gave the same results of querying the "grandidizionari.it" web service. In addition, we obtained phonetic translation for words not provided by the online dictionary.

[49] F. Albano Leoni and P. Maturi, "Manuale di fonetica" ,2002, Roma, Carocci.

[50] Free audiobooks repository from librivox, http://librivox.org/ (last visited March 3, 2014).

Finally, a linguistic model was obtained using the transcriptions of audiobooks. To test the performance of the acoustic model two different speakers (a male and a female) pronounced a list of 40 words taken from the vocabulary of the linguistic model.

TABLE I

TEST RESULT

Meaning	Speaker 1 (female)	Speaker 2 (male)
Recognized Words %	70	77.5%

The acoustic model was trained with two females and six men voices. Although female voices are 56% compared to the total, the male voice is better recognized than the female one. The results show that the variety of voices affects the quality of the result[51]. In addition, we noted that audio recordings that did not match with corresponding words are recognized as very similar words from the phonetic point of view (for example, the word “velocemente” is recognized as “velatamente”).

[51] M. Gerosa, D. Giuliani, and S. Narayanan: "Acoustic analysis and automatic recognition of spontaneous children’s speech", In Interspeech-2006, paper 1082Wed2CaP.9.

CONCLUSION

In my PhD course we addressed the problem of developing SDS for inflectional languages in particular for Italian language. These languages are characterized by words that have different suffixes depending on the conjugations, declensions and alterations. This leads to a high development effort and long grammar specification files hard to structure and to manage. My implementation introduces a morphological engine, which analyses each user input sentence and transforms every word in its canonical form. Hence, the grammar specification files will contain only the canonical form reducing the amount of information to store. This method introduces benefits such as minor grammar description file length, a better grammar structure and management, and an easier user behavior prediction. In my test, using the Olympus framework, we showed how Smart Grammar accomplishes a compression rate for grammar description files. So, the Phoenix grammar programmer does not have to worry about matching all possible inflected forms and with suffixes or prefixes. This also prevents the programmer to forget some utterances. Furthermore, the size of the Phoenix grammar is reduced even if the SDS is able to recognize a huge number of utterances that the programmer cannot directly know and write in the grammar definition file. This implementation can be reuse the idea of building morphological engines for other inflectional language and use the same principal to help building inflectional SDSs.

In ASR field, I developed an automatic method to obtain recordings, transcriptions and phonetic transcriptions in order to create an acoustic and linguistic model. My goal was to investigate the possibility of using a set of free audiobooks for generating a dataset as a complete database of ad hoc audio recordings. Then, an Italian acoustic model has been created by SphinxTrain[52].

[52] SPHINX free language model repository, http://sourceforge.net/projects/cmusphinx/files/Acoustic %20and%20Language%20Models/ (last visited March 3, 2014).