An Easy and Efficient Grammar Authoring Tool for Understanding Spoken Languages

A Novel Approach to Develop a Spoken Language Understanding Grammar for Inflective Languages

Antonio Rosario Intilisano, Salvatore Michele Biondi, Raffaele Di Natale and Vincenzo Catania
Dipartimento di Ingegneria Elettrica Elettronica e Informatica
University of Catania
Catania, Italy
aintilis@dieei.unict.it, salvo.biondi@dieei.unict.it, raffaele.dinatale@dieei.unict.it, vincenzo.catania@dieei.unict.it

Ylenia Cilano
A-Tono Technology s.r.l.
Catania, Italy
ylenia.cilano@a-tono.net

***Abstract*— In a Spoken Dialog System, the Spoken Language Understanding 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. In fact, the developer must include manually all the existing inflected forms of a word. Generally, a regular software developer does not combine linguistic and engineering expertise in spoken language understanding. For this reason, we developed a tool that produces a grammar for different languages, in particular for Romance languages, for which grammar definition is long and hard to manage. This paper describes a solution to facilitate the development of speech-enabled applications and introduces a grammar authoring tool that enables regular software developers with little speech/linguistic background to rapidly create quality semantic grammars for spoken language understanding.**

Keywords- Spoken Language Understanding; Natural Language Understanding; Spoken Dialog System; Grammar Definition.

I. INTRODUCTION

To build a Spoken Language Application in a specific user language, the developer has to design and develop a knowledge base called grammar for Spoken Language Understanding (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 [1]. However, the development of such a corpus is a slow and expensive process [2]. In SLU research domain-specific semantic grammars are manually developed for spoken language applications. Semantic grammars are used by robust understanding technologies [3,4] 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 [5]. 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 [6]. 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.

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 quality semantic grammars for SLU [7]. More precisely, the purpose of this paper is to ease the development of a CMU Phoenix Grammar [8], 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, we are developing 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 our 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 [9]. The purpose of this tool is to increase developer productivity; experimental results show that it also improves the coverage of the final Phoenix grammar.

This paper is organized as follows: Section II describes the behavior of SDSs. Section III explains the features of the

Romance languages with particular regard to the Italian. Section IV introduces Olympus, which is a framework for implementing an SDS, and its grammar parser, Phoenix, that use a particular grammar format. The proposed grammar format method is presented in Section V. The two following sections introduce the grammar generator that takes as input the new grammar format; its components are a Morphological Generator for the Italian language (Section VI) and a grammar editor (Section VII). Section VIII shows an example. Finally, in Section IX, we draw conclusions.

II. 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.

A. 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.

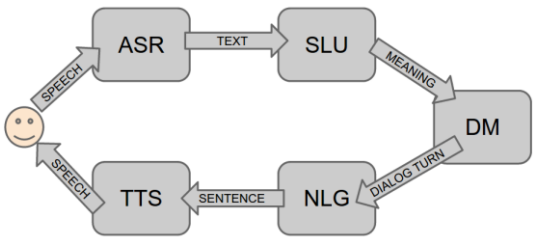

Figure 1. SDS structure.

B. How SLU Works

SDS needs a sophisticated SLU module [9] in order to implement dialog applications that go beyond solving simple tasks like call routing or form filling [11]. SLU is performed as a semantic parsing of spoken sentences. Current works in language modelling focus on two main areas: formal and stochastic approaches. Formal approaches to language modelling come in many forms and serve many motivations. This problem relates to the hand-coding of definitions of a language. Stochastic approaches involve the compilation of a finite-state machine in which the likelihood of a given word occurring is calculated based on the corpus, possibly, having the context of the preceding *n* words. All the stochastic models for SLU proposed to so far, perform the translation from a spoken sentence to a semantic constituent-based representation using statistical learning models. These systems are TINA [12], Chronus system from AT&T [13].

Development time, reusability and expertise required to create the language model, play a role in determining an appropriate solution in many cases [14]. Furthermore, manually developed grammars require combined linguistic and engineering expertise to construct a grammar with good coverage and, therefore, performance. It takes multiple rounds to fine tune a grammar, and it is difficult and expensive to maintain it. The second research paradigm adopts a data-driven, stochastic modelling approach. While it alleviates the labor-intensive problem associated with the first paradigm, it requires a huge amount of training data, which is seldom available for industrial applications.

These are difficulties [15] and the research community has potential areas of improvement focusing on these two problems:

- Systems have to be developed with little or no data. The manual grammar authoring is necessary for initial system deployment. Tools for fast grammar handcrafting make easier to enlarge the coverage of a grammar and, therefore, are crucial in this case.
- There are huge amounts of data available after deployment. It is hard to manage and manually analyze the data in order to find the problems in the initial deployment.

In this article, we introduce a grammar-authoring tool represents a solution for the first problem.

III. 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 [16]. Today, many voices have disappeared;

instead, others are present in the Romance languages [17]. 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 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.

A. 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 [18]. 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".

IV. OLYMPUS

This work was designed and tested within the framework Olympus [19]. Olympus is a complete framework for implementing SDSs created at Carnegie Mellon University (CMU) during the late 2000's. Olympus includes a dialog manager called RavenClaw [20], which supports mixed-initiative interaction, as well as NLU components that handle speech recognition (Sphinx) and understanding (Phoenix). Olympus uses a Galaxy [21] message-passing layer architecture to integrate its components and supports multi-modal interaction. The Galaxy architecture is a set of Galaxy *Servers*, which communicate to each other through a central Galaxy module called *Hub*. Olympus provides the infrastructure upon which it is possible to build Spoken Dialog Applications. Specific application functions such as instance dialog planning, input processing, output processing and error handling are encapsulated in subcomponents with well-defined interfaces that are decoupled from domain-specific dialog control logic. Each application needs the following domain specific components: a specific grammar, a dialog manager, a back-end server and a language generator module. These modules are strictly domain dependent and represent the core of the spoken interaction. The Phoenix parser represents the NLU module in the Olympus framework. The Phoenix parser [8] was developed by the University of Colorado in 2002 to develop easy and robust Natural Language Processing systems. It was then adopted by the CMU and used in the Olympus framework. The parser performs the human language syntactic analysis according to the rules that are defined in its grammar. For each user input sentence, the Sphinx module of the Olympus framework produces *n* text output. Each of them is associated with a probability. The higher is the probability, the more likely is the association between a text and a user sentence. Each of these *n* texts is parsed by Phoenix. The meaning extracted from the input sentence will then direct the Dialog Manager in deciding the corresponding action. Subsequently, the Natural Language Generation module will produce the output sentence.

A. SLU grammar in Olympus Framework

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 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.

B. 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. 2. In Fig. 3 there is an example.

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

Figure 2. Generic Phoenix grammar syntax.

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

Figure 3. Example of a Phoenix token.

A token can also contain other tokens, for example (Fig. 4):

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

Figure 4. 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.

V. A NEW GRAMMAR FORMAT

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].

Our approach consists of creating a new intermediate grammar 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 [23], our solution can facilitate grammar development by supporting the flow of information from a manually written source 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 [24], generates a file that can be reused and edited like a standard phoenix grammar.

The new format description is in Fig. 5.

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

Figure 5. 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. 6 depicts the new format description.

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

Figure 6. 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;

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

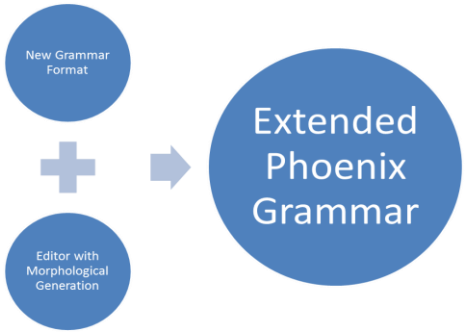

Figure 7. New grammar generation.

Our 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.

Fig. 7 shows the full process.

Our 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.

VI. 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. 8 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.

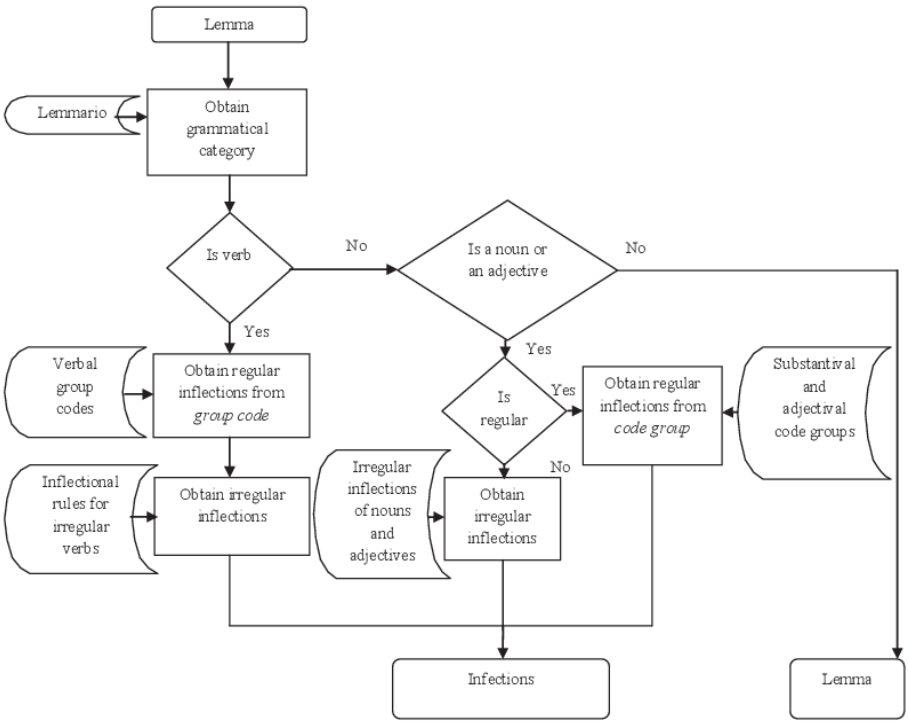

Figure 8. Morphological Generator.

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 an 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 [26][27][28], 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.

VII. GRAMMAR EDITOR

The grammar editor (Fig. 9) consists of a text editor modified for our purposes. This editor supports the new grammar format and the user produces the corresponding .gra file, by clicking on the "generate" button.

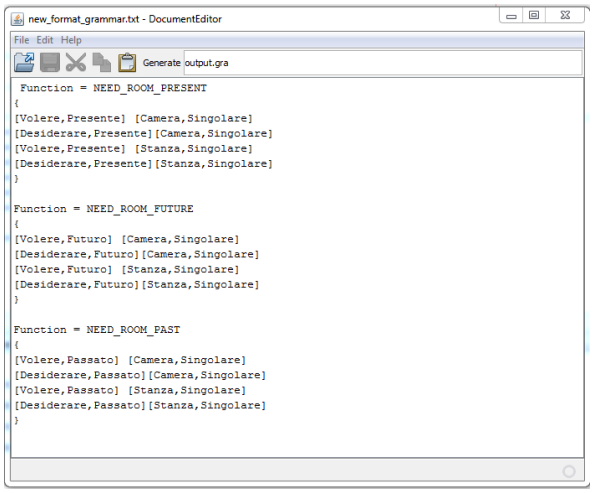

Figure 9. GUI of the editor.

This component reads and processes the grammar files (new format) and, using the Morphological Generator, obtains a Phoenix grammar file (Fig. 10).

Figure 10. New grammar generation process.

If the programmer does not know the SDS domain language, he enables the "translator" module (Fig. 11) between the Morphological Generator and the Editor.

For example, an English language grammar, as shown in Fig. 12, 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.

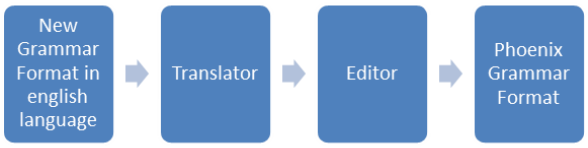

Figure 11. New grammar generation process with translator.

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 12. 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

VIII. 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. 11 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 13. 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 14. Generated grammar slots.

The second part, shown in Fig. 13, defines each token, including their word patterns. A more detailed explanation is along with the source code (output.gra file) [25].

The initial grammar, consisting of 21 rows, 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 15. Generated tokens.

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. 14.

```
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 16. Example of English grammar.

The generated grammar slots are shown in Fig. 15 and the associated tokens in Fig. 16.

```
[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 17. Generated grammar slots from English.

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

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

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

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

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

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

Figure 18. Generated tokens from English.

IX. Conclusion and Future Work

This paper investigates the problem of grammar authoring for initial system deployment when little data is available.

In this work, we propose a solution to simplify and reduce the amount of writing of the SDS grammar of inflectional language. This method reduces the effort to produce a grammar for a SDS especially for a regular software developer. The SDS used for our tests is the Olympus framework.

An editor has been developed for the translation of the new simple grammar format in the Phoenix grammar format. The editor uses a new Morphological Generator to obtain all possible inflected words that are used to create grammar tokens.

The proposed solution will be integrated in a major project called Olympus P2P [29], which is concerned with the upgrading and updating of an SDS grammar by means a peer-to-peer network to share new grammar tokens generated from the new grammar format.

ACKNOWLEDGMENT

The authors were supported by the Sicilian Region grant PROGETTO POR 4.1.1.1: "Rammar Sistema Cibernetico programmabile d'interfacce a interazione verbale".

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] S. Bangalore, M. Johnston, "Balancing data-driven and rule-based approaches in the context of a multimodal conversational system," Human Language Technology/Conference of the North American Chapter of the Association for Computational Linguistics. Boston, MA, USA, 2004.

[5] 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.

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

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

[8] W. Ward, "Understanding spontaneous speech: the Phoenix system," Acoustics, Speech, and Signal Processing, ICASSP-91, 1991 International Conference, pp. 365-367 vol. 1, 14-17 Apr 1991.

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

[10] Phd Thesis, International Doctorate School in Information and Communication Technologies DISI - From Spoken Utterances to Semantic Structures Marco Dinarelli.

[11] A. L. Gorin, G. Riccardi, and J. H. Wright, "How may i help you?," Speech Commun., 23(1-2): 113–127, 1997.

[12] S. Seneff. Tina, "A natural language system for spoken language applications," Comput. Linguist., 18(1): 61–86, 1992.

[13] N. Cancedda, E. Gaussier, C. Goutte, and J. M. Renders, "Word sequence kernels," J. Mach. Learn. Res., 3, 2003.

[14] Language Modelling for Spoken Dialogue Systems; Grammar-Based and Robust Approaches Compared and Contrasted Genevieve Gorrell, December 22, 2003

[15] R. Pieraccini, "Spoken language understanding, the research/industry chasm," HLT-NAACL 2004 Workshop: Spoken Language Understanding for Conversational Systems and Higher Level Linguistic Information for Speech Processing, pp 47-47, Boston, May 2004

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

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

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

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

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

[21] J. Polifroni, and S. Seneff, "Galaxy-II as an Architecture for Spoken Dialogue Evaluation," Proc. LREC, 725-730, Athens, 2000.

[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.

[23] M. Haspelmath and A. D. Sims, Understanding Morphology 2nd edition. London: Hodder Education, 2010.

[24] V. Catania, Y. Cilano, R. Di Natale, V. Mirabella and D. Panno, "A morphological engine for Italian language", ICIEET 2013: 2nd International Conference on Internet, E-Learning & Education Technologies, 2013, pp. 36-43, vol. 12(1).

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

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

[27] 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.

[28] 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.

[29] V. Catania, R. Di Natale, A. Longo and A. Intilisano, "A distributed Multi-Session Dialog Manager with a Dynamic Grammar Parser," 2nd International Conference on Human Computer Interaction & Learning Technologies, 2013, pp. 1-9, vol. 8(2).