
Raffaele de Amicis
Graphitech
rda@graphitech.it

Giuseppe Conti
Graphitech
giuseppe.conti@graphitech.it

Abstract
The capability of processing spoken commands is one of the most important features of modern multimodal AR/VR environments. This feature though requires programmers to compile some human supplied knowledge in the form of grammars which are then used at runtime to process spoken utterances into complete commands. Further speech recognition (SR) must be hard-coded into the application. This process is time-consuming, error-prone and must be repeated every time a modification to the system code is introduced.

This paper presents a completely automatic process to build a body of knowledge from the information embedded within the application source code. The programmer in fact often embeds, throughout the coding process, a vast amount of semantic information in various forms. This is done for instance by defining classes, objects or reference names, or through the declaration of method definitions. This research work presented exploits this semantic richness and it provides a self-configurable system, which automatically adapts its understanding of human commands according to the content and semantic information defined within the source code of the application. This knowledge is used to automatically generate the data necessary to configure the speech recognition feature of the final application. Finally the framework provides the SR feature transparently and automatically with very limited specific coding.

1. Introduction

The quest for a more natural form of interaction between man and computers has lead, during the last decades, to the development of a number of multimodal VR/AR environments. For the computer to be able to make use of gestures or voice commands some knowledge must be pre-programmed into the system. As consequence of this the programmer is obliged to compile the set of data which then will form the body of knowledge of the system. At runtime, when the relevant command is identified by the Speech Recognition (SR) sub-system, the command ID is passed to the VR system, which then activates the command. More advanced true-multimodal systems post-process the data coming from independent unimodal engines (e.g. for speech, gesture etc.) unifying their semantic meaning into a single action before passing the command to the application engine.

The definition of the body of knowledge for the recognition of spoken commands is often accomplished following a time-consuming process where the user defines both a vocabulary of supported terms and a grammar. Grammars define the way user’s commands shall be related to actions by associating utterances, (i.e. spoken commands) to command IDs. Some systems let the programmer augment grammars with semantic information that can be then used by the system.

The process of coding an application yields a vast amount of textual information which inherently describes the functionality of the system. Such information, encoded according to specific programming language syntaxes, is often written in human-readable form. Source code files are not only a mere collection of terms, numbers and symbols but they formally express elements and actions in human readable sentences, which resemble statements used in near-natural spoken languages. Such high degree of “readability” is not only desirable but it is an essential requirement during software development to ensure adequate maintenance and future extensibility. Further, due to the very nature of Object-Oriented Programming, modern software is modeled using classes and objects that often represent abstract functions with specific and enclosed functionalities. For instance, if a programmer has to define the behavior of a geometry, they will create a class using a self-descriptive name such as “shape”, and its fields and functions, which define the means of interaction with the environment, will be in most cases defined using some form of human understandable terminology.

This acknowledged praxis has been a source of inspiration for the implementation of the system here described which is capable of automatically creating a knowledge repository from the semantics encoded in the code files. This readily-available body of knowledge is then automatically used to interpret the commands formulated by the user. This research work, although generally applicable to various contexts, has been tested in the specific context of VR/AR systems. This choice takes
advantage of the great level of semantic information intrinsically embedded in code for VR/AR systems due to the vast number of highly descriptive spatial definitions and functions. Our framework has been initially implemented for the recognition of voice commands but extensions to other modalities are planned for the future.

2. Related works

Traditionally speech recognition facilities have been embedded into VR systems to provide a “natural” means of interaction [21] with the virtual world, and to enhance the efficiency in the workflow [14]. Multimodal VR/AR systems [20] have been developed for various purposes. In [14] the AR system described has been developed for educational purposes, allowing control of interface components through the use of speech commands. Most systems developed for engineering applications [16], for complex assembly and maintenance tasks [17] usually use speech engines such as [19] to recognize short commands or in replacement of simple input from the keyboard [24]. Other works have brought to the creation of speech-enabled VR/AR environments based on PDAs [15]. Voice is used to navigate, annotate, and communicate (through voice-over-IP) with other users and a context sensitive interface shows the speech commands available.

More advanced multimodal VR applications such as [22] have proposed an agent based structure capable of processing inputs from different modalities through the use of Feature Structures (FS) unification, a generalization of expression unification in logic programming languages. A FS represents a type of object with its set of feature values. Their use allows comparing the consistency of two representational structures [23]. The unification process checks for the compatibility of data structures and it merges their features into a single data structure. Authors [23] stress how this approach is ideal for multimodal input characterized by a high degree of redundancy in the information. In [27] the authors propose the use of Interaction Graphs, diagrams whose tokens contain information coming from different modality, to show the user’s progress within the current task.

The StudierStube [29] VR/AR platform introduces a new level of abstraction to the multimodal interaction. The system makes use of an open architecture for tracking devices called OpenTracker [30] which provides high-level abstraction over different tracking devices and interaction modes. The architecture [30] supports SAPI (Speech Application Programming Interface) compliant recognition systems and it allows easy configuration through XML scripting, but, as most recognition systems, it must be pre-defined beforehand. The system is also capable of TTS (Text-to-Speech) functionalities. The programmer writes the XML that associate utterances with command IDs. These are passed, once recognized, to the main VR core through the OpenTracker data structure and used to interact with the system. The system has a number of pre-defined speech enabled objects such as widgets. However the programmer has still to manually create the vocabulary and the grammar and define the data wiring within the main code of the software.

As far as spoken commands are concerned several commercial systems [18] make use of finite state grammar, a technique that allows filtering the number of messages to be decoded by the system. Most modern engines [19] [26] make use of pre-defined dictionary and rule sets compiled into context-free grammars (CFG) which can be used by the system to retrieve, from spoken utterances, the information required to activate the relevant commands. Semantic information can be then included at a grammar level in various commercial recognizers [18] [19]. This approach enhances precision over standard dictation systems since it allows great narrowing of the number of commands that have to be interpreted by the system. In fact since early works [21] it has been showed that restricting the language scope, from general natural language to domain specific commands, yields a much higher precision. Vocabularies are thus defined a priori [25] and the number of recognizable commands is limited in size [28] and defined according to the specific context of the application. As noted by [25] the development of more comprehensive vocabularies and grammars represents an important achievement since this can enhance significantly the expressive power of the application.

Past works [20] stressed the need for expressing knowledge contained in virtual environments and for extracting semantics out of virtual scenes. In [20] is also highlighted the importance of having some form of linguistic knowledge which can be embedded in a VR system in an automatic way. Numerous works, in the field of Natural Language Processing (NLP), have tried to answer to this demand. In particular the problem of understanding the meaning of a term in a wide semantic context has been a subject of great interest. The problem, known as Word Sense Disambiguation or WSD, is in fact a fundamental step for the overall process of knowledge understanding. As noted by [2] Word Sense Disambiguation is a necessary process not only by itself but for the following processing levels such as for natural language processing. WSD requires building of a representation of knowledge.

As far as WSD is concerned the seminal work of [5], carried on in response to the need for automatic machine translation, highlighted the need for defined logical structuring of languages capable of mapping the semantics of natural language. Word Disambiguation, has being extensively studied for the last six decades by
scientific with an Artificial Intelligence (AI) background across several domains, from machine translation [3] to speech and Natural Language Processing (NLP) [6], and Knowledge Management [4]. The problem has been described by [2] as AI-Complete, e.g. that needs tackling of major AI problems manifested during the synthesis of a human-level intelligence.

Interest emerged in the past years over computational linguistics has refueled research in WSD. In particular the work done by [8] at the Cognitive Science Laboratory at Princeton University has lead to the development of WordNet, today’s most famous resource for computational linguistics. WordNet was built upon the theory of semantic networks emerged during the late 1950s, and it can be considered as a knowledge repository based on modern psycholinguistic theories of human lexical memory. Research activity is currently ongoing in order to enrich the database of specific domain lexicon and support for various languages [9] [11]. Several authors have tried to use WordNet for WSD. Authors in [13] calculate the semantic distance between terms through the assignment of weights based on types of semantic relations between words (i.e. synonymous, hyponymy etc.). Shorter distances define closer senses between words. Authors in [6] use WordNet for WSD with good results. The algorithm, implemented in Prolog, tries to develop a domain-independent syntactic parser for text fragment analysis. The algorithm analyzes both nouns and verbs using heuristic methods. The authors measure semantic similarity at different levels taking into account if the words are strict synonyms (e.g. they belong to the same synset), if extended synonyms (if they are connected by hyponymy/hypernym relations), or if they share the same parent node. According to [6] semantic similarity between words is inversely proportional to the semantic distance between words in a WordNet hypernymy / hyponymy hierarchy. The algorithm is based on 8 steps where pair of nouns are checked and ranked accordingly. Authors in [7] use a mixed approach where WordNet ranking is mixed with word-to-word concurrences based on Internet-based statistics. Finally authors in [1] have developed an algorithm that performs WSD to find appropriate terms during the process of creation of concept maps. Concept maps are sets of concepts drawn in a table which can be used during brainstorming sessions. These can help represent a person’s cognitive structure by externalizing concepts and propositions [1] and by defining links between them. The algorithm makes use of hypernym sequences to represent the system’s understanding of the knowledge contained in the concept map. It then analyses the most appropriate terms, according to the context of the map, to aid the user to create both concepts and linking phrases between concepts.

3. Background

In Figure 1 is represented a modern VR/AR system [29] [31] [32] which allows convenient definition of relations between spoken utterances and systems actions via Context-Free Grammar (CFG). These can be defined through human-readable Extensible Mark-up Language (XML) files which then are read by the system and used by a SAPI engine to isolate the corresponding lemmas. Advantages in using CFG are the narrowing of the term domain to be interpreted by the recognition sub system and the definition of a precise language model for the recogniser. This approach is preferred when a relatively limited set of commands must be decoded, as in the case of VR applications. The recognition subsystem is interfaced to the VR/AR environment and the relations between spoken commands and computer actions are hard-coded. The output of the speech recognition process are probability ranked lists, which are passed to a language parser that interprets them as proper commands.

OpenTracker [30] offers a more flexible approach that allows convenient abstraction, over the speech-recognition process, providing the means for simple transmission of commands from the speech sub-system to the application engine. This approach requires hard-coding of functions, which implies, every time the application is changed, the compilation of a new dictionary and grammar. Since there are no means for compiling such semantic information automatically into the system, they are provided by the programmer as result of a rather time-consuming process, and, in case of change of the system, the entire process has to be repeated over again. Finally, at the application level, the programmer is still forced to hard-code the procedures that handle the information generated by the recognition sub-system, in
4. System Overview

Our research work has been inspired by the observation that programmers, during the development process, implicitly define a vast knowledge, whose formalism, strictly defined by the rules of the programming language adopted, is only partially exploited, by the compiler and later by the executer, to create the program. Such body of knowledge carries a wide range of implicit semantic information which is not exploited at later stages. This information is contained in class names, objects or reference names, in method definitions and method arguments. For instance, a typical VR/AR system would define the objects of the 3D environment through classes named e.g. “Geometry” or “Cube”. Actions upon these objects would be defined by methods such as “extrude_line”, “scale_shape”, “bend_shape”. These methods would have arguments of the type “bend_shape(float angle_X, float angle_Y, float angle_Z)”. Although the latter expression could be seamlessly replaced by “funcN23(float a, float b, float c)” it is acknowledged that writing readable code which resembles near-spoken language is an important requirement for good programming and it is a primary prerequisite for maintenance and future developments.

Our system is capable of turning such semantic information, intrinsically contained in the source code, into a data repository, which can be then used to transparently and automatically support speech recognition at the application level. As illustrated in Figure 2, the system comprises a C++ library, which is linked by the general application, and a Normalization application. Normalization is performed before compilation of the final application for automatically extracting semantic information from the code. It also creates the knowledge repository which is later used to provide the speech-recognition functionality. To make the application speech-enabled the user must follow a few simple guidelines. Every time the programmer requires a function to be speech-enabled he/she tags it with a specific preprocessor macro. During the normalization process the tagged source-code is parsed and the relevant semantic information extracted. This information is used to create the knowledge repository for the automatic compilation of the Context-Free Grammar (CFG). The CFGs are then used at runtime by the system to provide support for the framework, which automatically and transparently links the data coming from the recognition sub-system to the main application. The only requirement for the system to take full advantage of the underlying semantics is the definition of function names in near-natural language.

Figure 2. General Architecture: the system uses its source code to automatically generate grammars for its speech-recognition subsystem

As detailed in the following section the system takes advantage of the terms used by the programmer to model its own knowledge repository. This knowledge is modeled according to so-called contexts. We refer to the term context meaning the set of information, confined to a certain scope, which boundaries a specific body of knowledge. Enforcing the Object-Oriented Paradigm, the system assumes that terms within the same class have strong relation one another, since they are usually related to a shared concept. The algorithm therefore considers each class as representation of a different context whose tagged method names and arguments are used as sources for the knowledge repository. The linguistic knowledge
built upon such repository is used to automatically expand the meaning defined by each action. A Semantic expansion lets the system respond to a much wider set of commands, if compared with the standard fixed-expression approach. The following example illustrates the advantages of our approach. We take the method \texttt{"bend\_shape (float angle\_X, float angle\_Y, float angle\_Z)"} that belongs to the \texttt{"basic\_shape"} class as example. The programmer tags this declaration as follows:

\begin{verbatim}
SSPEECH\_FUNCTION (void bend\_shape (float angle\_X, 
float angle\_Y, float angle\_Z));
\end{verbatim}

During the process of normalization a context for each class is automatically created. Other tagged methods, such as \texttt{"extrude\_geometry"} or \texttt{"scale\_shape"} will be also used to create the context's knowledge. A first level analysis separates nouns, verbs, adverbs and adjectives. Then through a Word Sense Disambiguation (WSD) algorithm each lemma is expanded and its relevant synonyms are computed. In our example the algorithm recognizes the term \texttt{"bend"} as a verb, then, selects synonyms, such as \texttt{"twist"}, according to its context and excludes those whose sense is not related with the general semantic meaning of the context, such as \texttt{"crouch"}, \texttt{"stoop"} or \texttt{"bow"}. The same process is repeated for the term \texttt{"shape"}. Finally the relevant CFG is built and loaded by the system when the corresponding object is in scope. Enforcing the Object-Oriented Paradigm, the use of CFG takes advantage of the inheritance between classes. In the following example a class \texttt{"spline"}, which was extended from \texttt{"basic\_shape"}, would inherit the parental \texttt{"semantic"} properties. The user can thus command \texttt{"twist\_that\_shape by five-point-eight three-point-four four-point-eleven"} and the system interprets the command bending the selected shape according to the values set by the user. The comprehension of the complete command, which was made of the \texttt{"specific"} expression plus of a series of 3 double precision values, is enforced throughout the process. In fact, during the normalization process, argument types are detected and used to automatically configure the system to wait for particular types of arguments after detecting certain commands. In our example the system recognizes the command referring to the \texttt{"bend\_shape"} method and activates at runtime the CFG related to the detection of double precision values. It is worth underlying how the entire process is automatically implemented with nearly no extra code for the programmer, who is only required to type a few macros when appropriate. The next section will describe in detail the entire architecture providing an insight of its algorithm.

\section{Semantic Expansion}

Semantic expansion provides the means for enriching terms with their synonyms according to a specific context. The problem of understanding the meaning of each selected term in relation to its wider semantic context has been tackled through the definition of the Word Sense Disambiguation (WSD) algorithm illustrated in the following sections. WSD provides the understanding of the correct sense of polysemic words (i.e. with multiple meanings). WSD is performed with the help of knowledge extracted by the code of a single class, therefore not taking into account code of other classes. This is necessary since different classes may refer to different semantic domains (i.e. one might refer to visualization another to creation of geometry) and they require independent WSD processes. Our algorithm was inspired by the work done by [1] and it makes use of WordNet as lexical reference system.

The information extracted by the Normalizer is modeled using \texttt{"synsets"} of WordNet. A \texttt{synset} is a set of synonyms (hence the name), which defines, according to psycholinguistic theories, a common lexical concept [9] and therefore a meaning. A description of the functionalities of WordNet is beyond the scope of this paper (see [8] [9] [10] [11] [12]), however, it must be highlighted how WordNet allows for retrieval of both lexical relationships, occurring between words, and semantic relationships, occurring between meanings. Semantic relations, such as synonymy/antonymy (terms with same/opposite meaning), hyponymy/hypernymy (relation which defines that “a” is a kind of “b” and vice versa), meronymy/holonymy (“a” is the part of “b” and vice versa) are represented through pointers between synsets.

\subsection{Normalization}

When the Normalizer is called it parses the source code looking for preprocessor tags. To avoid conflicts if the process of normalization is not called before the final compilation of the application the tags are expanded into blank spaces. During the normalization, for each speech enabling tag (i.e. the preprocessor macros) the Normalizer extracts all the relevant information (e.g. the function name, return values and arguments). As far as the semantic expansion is concerned only function names will be used. The remaining data is used at later stages. The normalization replaces each tagged method with a more appropriate data structure, which will be used by the system to automatically activate these functions at later stages. The new data structure operates as a signature and contains information on the method name, arguments
and return type, which are then used at runtime to activate the proper functions. For our example the Normalizer automatically generates the following source code:

```plaintext
SSPEECH_FUNCTION_ENCODED (Shape, void bend_shape_float_angle_X_float_angle_Y_float_angle_, bend_shape)
```

Then the algorithm starts building the knowledge of the context. This is done in two phases: performing a morpho-syntactic disambiguation and then a sense disambiguation. During the first process a set of heuristics defined in the system help determine the syntactic category of each lemma. Due to the nature of the text being parsed this task does not present particular complexity. In fact, in most cases method declarations are defined following an essential style (e.g. “move_cube (int x, int y, int z)”). A set of heuristics then was developed to process and perform efficient syntactic analysis in order to determine whether a term is a verb, a noun, an adjective or an adverb (for instance the term “shape” can be both a verb and a noun). At the end of the morpho-syntactic analysis for each i-th class or context the following four sets are defined:

- N_i \{n_0 \ldots n_i\}
- V_i \{v_0 \ldots v_i\}
- A_i \{a_0 \ldots a_i\}
- D_i \{d_0 \ldots d_i\}

These respectively define the set of nouns, verbs, adverbs and adjectives present in each class. The next step, the sense disambiguation requires much more complex analysis. At the present stage the algorithm provides disambiguation for only nouns and verbs, which however define the most significant part of the semantic information present in the source code. The process of Word Sense Disambiguation (WSD) aims at identifying the correct meaning (or sense) of a polysemic word in a given context. The algorithm, as also described by [2], can be split in two phases: it searches the different meanings of each word belonging to the context and then it establishes the relation of the correct senses with the given context. The way a sense is defined has been matter of intense debate within the scientific community and among lexicographers, but it is beyond the scope of this research work. Here sense definition relies on WordNet and the proposed algorithm looks after how to define a method to represent a context and how to provide the knowledge that the system requires to associate senses to the given context.

5.1.1. Context Definition and Noun Disambiguation

As underlined by [1] the selection of words used to create the context is a crucial aspect of WSD. After completing the morpho-syntactic analysis, the set of lemmas identified as nouns N is used to build up the context of the class. This choice allows efficient analysis and construction of the context, dramatically reducing the overhead required for the semantic processing. Yet this approach features good accuracy. In fact, as noted by [10] verbs have a tendency to change their meaning according to the nouns they are related with, whilst this tendency is limited for nouns. For instance the set of terms \{curve; cube; sphere; geometry; line\} generally let the majority of people think about the same topic (i.e. generation and manipulation of shapes) and can be easily used to identify a context. However the same is not true for the set \{create; delete; move; modify\}, which defines more generically different types of actions that can be performed upon various contexts (not necessarily related to geometries). Therefore the second set is less suited for our purpose. In short, building a context using nouns has proven an efficient way to represent a fast, stable yet representative approach.

In order to create the context a first level of normalization is performed using the morphological processing tools of WordNet. This process eliminates inflectional endings from each n_i-th element of N_i \{n_0 \ldots n_i\}, for instance, from plural terms and third person present tense verbs (e.g. “shapes” ? “shape”, “bends” ? “bend”). Each morphologically-normalized element is then used to retrieve its corresponding synsets from the database according to its syntactic category. Specifically for each n_i-th noun a search is performed returning all the synonyms and in case of polysemous nouns, all synsets for each sense. Let S_i \{s_0 \ldots s_i\} be the set of synsets of the i-th class created from the original set of nouns N_i \{n_0 \ldots n_i\}. The full set of possible hypernym sequences H \{h_0 \ldots h_n\} whose first element is a synset belonging to S_i is then retrieved. Since a synset can have several hypernyms it derives that s = t. It is here worth underlying that a concept represented by the synset X \{x, x', \ldots\} is a hyponym of the concept represented by the synset Y \{y, y', \ldots\} if in spoken English the sentence build as “An X is a (kind of) Y” is considered correct [9]. If such event occurs Y is said to be a hypernym of X. According to the convention widely acknowledged in literature [8] [9], the hypernymic (or superordinate) relation is formally defined through the symbol “@? “. For instance the term “square”, considered as a noun (and not as a verb or adjective), has 8 different senses. Two hypernym sequences referring to two different senses are then:

```plaintext
Sense 1 = square, foursquare
  @? rectangle
  @? parallelogram
@? quadrilateral, quadrangle, tetragon
@? polygon, polygonal shape
@? plane figure, two-dimensional figure
@? figure
@? shape, form
```
We call $H_i \{h_{0i} \ldots h_{si}\}$ the context of a class. In fact $H_i$, which contains the full set of possible hypernym sequences extracted from $N_i \{n_{0i} \ldots n_{ni}\}$, is the body of knowledge used to describe the semantic knowledge embedded into the code of a class and it is used at runtime to expand the semantics of the speech recognition feature. Thanks to the properties of the hypernym relationship [9], it is guaranteed that there will be no loop in the data structure and that each sequence will terminate with one of the 25 unique beginner nouns [9], which define the most general concepts in WordNet (i.e. act, substance, event etc.). In analogy with [1] a hypernym sequence is defined as an ordered list of synsets where the following Boolean condition is satisfied:

$$\left( h_{ni} \lor h_{n(i+1)} \right) \land \left( h_{ni} \in S_i \right)$$

The formula considers each $n+1^{th}$ element as a hypernym of $n^th$ and the first element of the sequence as a synset belonging to $S_i \{s_{0i} \ldots s_{si}\}$. Once the context has been built it can be used to disambiguate each noun belonging to $N_i \{n_{0i} \ldots n_{ni}\}$. The approach followed implements a clustering process of hypernym sequences inspired by [1]. Our approach departs from the original research work delivering a better control of the different elements belonging to the context, and it introduces a new process which operates verbs disambiguition.

As represented in Figure 3, for each noun belonging to $N_i$ let $P_N \{p_0 \ldots p_i\}$ be the set of its hypernym sequences and $R_i \{r_0 \ldots r_i\}$ a copy of the context $H_i$. For each $n^{th}$ sequence $p_n$ of length $l$ every possible sub-sequence $p_{nk}$ of length $k$ (with $l = k = l$) is extracted. Each sub-sequence will contain the last $k^{th}$ elements of $p_n$, starting from one of the 25 beginners nodes of WordNet. This way the algorithm starts processing the full sequence reducing at each step the value of $k$. The process is repeated until $k = l$ is found, i.e. until the one-synset sequence containing only one of the 25 beginners nodes is reached. Each sub-sequence $p_{nk}$ is used to filter the copy of the context $R_i$ through a clustering process. Each cluster groups the sequences, belonging to the context $R_i$, whose first $k^{th}$ elements coincide with the $k^{th}$ synsets of the sub-sequence $p_{nk}$ taken as reference. Every time such a match is found the corresponding sequence is removed from $R_i$ and added to the cluster. As suggested by [1], the weight of each cluster is calculated as sum of the differences between the number of elements of each sequence of the cluster and $k$, i.e. the length of the reference sequence $p_{nk}$.

In other words the weight of each cluster is defined as follows:

$$W_j = \frac{1}{\sum_{n=0}^{\text{cluster.size}} (l_n - k)}$$

$W_j$ provides a measure of the extent to which the sequences belonging to the cluster differ from the sub-sequence $p_{nk}$. The higher $W_j$ the less the sequences contained will differ from the one chosen as reference. The clustering process is repeated until every subsequence of each element belonging to $P_N \{p_0 \ldots p_i\}$ has been extracted. The cluster(s) with the highest value of $W_j$ is taken and $p_{nk}$, the first synset of its reference subsequence, is used to retrieve the sense of the noun.

**Figure 3. The clustering process**

This value in fact indicates the highest degree of matching of the chosen sub-sequence with the general context represented by the set of sequences belonging to $R_i$. This way it is possible to measure the “closeness” of a sequence to the general meaning of the context $H_i \{h_{0i} \ldots h_{si}\}$. Consequently the sense that is referred to by the chosen sub-sequence is considered to be the closest to the general semantic context of the class and it is used to extract the synonyms of the noun.

A more complex process is required for $V \{v_0 \ldots v_k\}$, which contains verbs. As noted by [10], verbs tend to be more polysemous than nouns, which might indicate their
higher flexibility. This observation has led the authors not to use verbs to build the contexts for the classes. The approach followed by our algorithm takes advantage of the data structure or context already built to disambiguate the nouns. As in the previous case, the aim is to identify the senses of the verbs according to the context taken into account, in our case the context of the class. To do so we parse the list of verbs $V_i \{v_{i0} \ldots v_{it}\}$ that was extracted during the syntactic analysis. For each verb we extract the list of derivationally related forms using WordNet. Derivational forms are nouns which are morphologically related to a verb. In our example among the derivationally related forms of verb "bend" there are terms such as "curve", "flexure", "bender" etc. The algorithm selects the sense of the verb whose derivationally related forms are characterized by highest degree of matching with the semantics of the context, and this sense is then used to retrieve its synonyms. Specifically for each $n$th verb belonging to $V_i \{v_{i0} \ldots v_{it}\}$ the system extracts the list of synonyms $D_v \{d_{i0} \ldots d_{im}\}$ pointing to the verb derivational forms. Then, for each $synset$ $d_m$ we extract the list $W_d \{w_{0} \ldots w_{j}\}$ of $j$ synonyms within the sense specified by $d_m$. For each $s$th element $w_s$ of $W_d \{w_{0} \ldots w_{j}\}$ the approach previously described for noun disambiguation is repeated. Thus, for each $w_s$ the correspondent $k$ sequences of hypernyms are retrieved, the sub-sequence of hypernyms extracted and each sub-sequence is used to create a group of clusters from a copy of the context $H \{h_{i0} \ldots h_{is}\}$. As illustrated in Figure 4 (now proceeding from top to bottom) for each hypernym sequence the weight is calculated. The highest weight among the clusters is associated with the correspondent $j$th word belonging to the list $W_d \{w_{0} \ldots w_{j}\}$. In order to find a measure of the general correspondence of the hypernym sequences to the semantic context $H \{h_{i0} \ldots h_{is}\}$, the mean value $w^T_{m}$ of the $j$ maximum weights is calculated as follows:

$$w^T_{m} = \sum_{i=0}^{j} \frac{w_{si}}{k}$$

This process is repeated for each $m$th derivational form, extracted from the verb, which belongs to $D_v \{d_{0} \ldots d_{m}\}$. The derivational related form, whose weight $w^T_{m}$ is the highest, indicates the most suitable sense for the context of the class, and the synonyms according to that sense are extracted from WordNet and used for the rest of the algorithm.

The process of semantic expansion described so far for nouns and verbs is eventually repeated for each class, or context, where a speech enabling tag has been found by the preprocessor.

5.1.2. Automatic Creation of Grammars

After the disambiguation of nouns and verbs the relevant synonyms are extracted from WordNet. These are used to create different CFGs, one for each class or context that was tagged. Here sets of synonyms are used to compile flexible grammars that consider all the matching combinations of synonyms for terms used in the source code. The application creates XML SAPI-compliant grammars [19] which are then used by the speech recognition engine. This process yields as result a speech recognition system that is still based on CFGs, thus assuring a high level of precision, but is significantly more flexible in the comprehension of the user’s commands. Most importantly our method does not require any intervention from the programmer since the system ensures that the every time a change in the code is made the set of grammars are consistently updated with the changes introduced.

6. Results
When running the final VR/AR application, multiple grammars are loaded by the speech recognition engine [19] as required by the system. The system at runtime identifies objects in scope and activates the relevant CFGs. If the user issues a complex command such as “twist that shape by five-point-eight three-point-four four-point-eleven” the system is able to respond correctly because it uses the CFG that contains the expanded method “void bend_shape (float angle_X, float angle_Y, float angle_Z)”. The loaded CFG permits to decode the first part of the command (i.e. twist that shape). Further, once the command has been recognized as part of the list of possible commands, the system activates at runtime the set of CFGs that allows the processing of the arguments. In the example the system knows (from the normalization process) that a set of 3 double precision values must follow the command. Therefore it activates the CFG that handles double precision values. If, as in the given example, the user issues the command with the expected data structure (i.e. followed by 3 double precision values), the system interprets and executes the complete command bending the shape according to the required values. A number of standard CFGs have been included in the system in order to provide support for Double, Floats, Booleans or, in the case for instance of strings, for switching to continuous CFG-free dictation. This approach can be considered as an enforcement of the Feature Structures (FS) [23] unification approach since it implicitly filters correct content (in this case the command and the argument) according to their data structure. Thus increasing the level of precision of the system.

7. Conclusions and Further Developments

This research work provides an original approach for creating the knowledge model necessary for multimodal systems. It is a first attempt towards the exploitation and reuse of the vast amount of semantic information that programmers embed throughout the development of applications. In particular this paper illustrates how the body of knowledge contained into code elements, such as method declarations, can be exploited by multimodal applications to make the system more adaptable to the user’s interaction. Sentences embedded in the application code are expressed in pseudo-natural human terms and they clearly contain a very high degree of semantic information such as the nature and kind of actions being carried out by the classes. Examples from the VR/AR domain have helped describing our approach. It has also been illustrated in detail how terms in the code are disambiguated in order to provide the semantic expansion required, and how the algorithm extracts the sense most suited for the various contexts. The algorithm makes use of WordNet to find the sense of lemmas which better relate to the context chosen. Interestingly the research work has highlighted the need for even more specific ontologies. In fact, although WordNet is a lexical ontology where specific relations between words have been formalized (such as synonymy, hyponymy, hypernymy and antonymy), its focus is not the creation of a domain ontology, but on organizing terms of various domains according to their lexical and semantic relations. This is an important distinction since for instance WordNet does not require the lexicographer to consider hyponyms of words at the same level of generality, which is instead typically required for the correct definition of the class hierarchy of an ontology. An evident limit using WordNet emerges if for example, we refer to the domain of a VR application, and we retrieve some hyponyms of the synset Shape. It is clear from Figure 5 that circle and square are not at the same level of generality as the synset solid. Those are at the same level in WordNet though since they are all hyponyms of the synset Shape. In a VR environment a distinction between 2D and 3D objects must be provided. Here the use of ontologies would advisable since they would help defining finer and customized relations between elements.

![Figure 5. Different levels of generality between siblings](image)

This research work is part of a project that aims at exploiting the advantages of semantic information for the development of VR/AR environments. The framework here illustrated has been developed to exploit such richness for a speech recognition engine embedded in a VR/AR-based design system [32]. Further development will implement this approach for other modalities and will explore the opportunity to take advantage of comments as further source of information. At the present stage the algorithm provides disambiguation for only nouns and verbs. In the future we will consider adverbs and adjectives although these play a minor role for semantic expansion. Finally WordNet contains many domains for doctrines and arts, i.e. linguistics, art, psychology, engineering, economy and medicine, but lexical limits still exist for specific domains. Therefore the research team is engaged in the integration of lexicon for VR/AR and design, which are domains specifically required for the system.
8. Acknowledgments

This research work is part of the projects InSiDe and SIMI-Pro financed by the Provincia Autonoma di Trento.

9. References


