# Cube – A Knowledge Extraction System (November 30, 2009)

F. S. Komori, F.B. Colombo and M. N. P. Carreño

*Abstract*— **We describe an expert system that is currently in development and has as a goal automatic knowledge acquisition from documents written in plain English. The envisioned system will be able to answer simple questions based solely on information it has acquired unaided by any human operator. By reading articles in PDF format or web pages, such as Wikipedia, the system will be able to increase its knowledge base, providing the users with better answers. Currently the system works only with texts in English. We make use of the Stanford Parser to help in the natural language interpretation. To store knowledge a representation structure similar to a semantic network was developed. It offers a few advantages which will be described in greater detail.**

*Keywords*— **Expert systems, knowledge representation, automatic knowledge acquisition.**

## I. Introduction

The current revolution promoted by information technology offers anyone with internet access a large, and constantly growing, amount of information. However, in order to gain access to this information, certain tools are needed. Currently search engines like Google, Yahoo! and Bing are essential to find information on the web. These tools return a list of links to pages or documents on the Internet containing data that might be related to the query entered by the user (a set of keywords). The list of links is processed by a non-intelligent algorithm and the user is forced to navigate among the many links returned in order to find the desired information.

This method is important and has been very successful. However, it has expressive limitations. Although such systems can point to sources which have a high probability of containing the answers to a question, it cannot directly answer the question. This has motivated the development of systems capable of directly answering a users question via the web, such as Google Squared (1) and Wolfram Alpha (2). These projects try to automate the information extraction from the existing sources, formatting and filtering the requested information. They are also capable of dynamically generating results for a query based on the acquired knowledge. We consider the next step in this evolution to be the construction of knowledge servers (3); systems that are able of extract, store and provide information in an intelligent manner. The development of such a system is the goal of this project.

## II. Objectives

We aim to develop a system, dubbed Cube, which is able to automatically extract knowledge from different sources, like PDF documents and web sites. The system will use natural language processing to extracted knowledge from these sources. It must also be able to store this knowledge in a manner that allows for querying. The system will also be able to generate answer to user submitted queries from the stored knowledge.

## III. Methodology

In order to create our knowledge base we must be able to process the source text and extract knowledge from it. An input module that realizes this task was developed. Two important parts of this module are further described below.

### A. Natural Language Processing

The input texts are written in plain English. A natural language processor is therefore essential to obtain the semantics of a sentence. To aid in this task, we made use of the Stanford Parser (4). This parser is written in Java and it is statistical, using a probabilistic context free grammar (PCFG).

Our software extracts sentences from the source text and inputs them in the Stanford Parser. The parser returns a tree that represents the grammatical relationships between the words in the sentence. This structure is then parsed by an automaton we developed to generate a graph that represents the knowledge contained in the sentence.

### B. Knowledge Representation

There are several well known ways to represent knowledge, such as frames, the entity-attribute-value model and semantic networks. The entity-attribute-value model was discarded because it isn´t efficient at storing knowledge in a generic manner and generating knowledge through inference.

Frames (5) are based on the entity-attribute value model, but are able to represent knowledge in a more structured way. Frames are stereotyped situations that are previously defined, based on frequently experienced situations, which may be changed to define a new situation. A frame is a sort of network where the top levels represent knowledge about what is always true about a given situation and what could be expected from such a situation. There are also terminals which can be filled in to better describe the situation. Kicking a ball can be a situation. Scoring a goal can be an expected result from such an action. The size and type of the ball are terminals that help better describe the situation. However creating the frames to represent any kind of situation that a general knowledge acquisition system might encounter is difficult, so we discarded this structure as well.

The semantic network (6) was the approach implemented here. A semantic network is a graph composed by vertices that represent concepts and edges, that may or may not be directed, which represent relationships between these nodes. An

This project has been supported by FAPESP and CNPq.

example of a semantic network is pictured below.

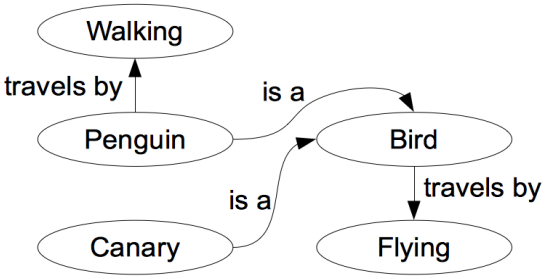


Fig. 1. A semantic network collected from 4 sentences and representing knowledge about two types of birds and how they travel.

Semantic networks provide a simple way to organize knowledge and they also allow inference. Knowledge that may not have been explicitly input into the system may be derived from this representation. If we ask how a canary travels, there is no direct answer. However, we can see from the network that a canary is a bird. If we ask how a bird travels, there is an answer: a bird travels by flying. This mechanism allows us to infer new knowledge from the network: a canary travels by flying.

We should also note that the edge *travels by* connected to the *Penguin* vertex is necessary because otherwise we would incorrectly answer the question "How does a penguin travel?". If we were to infer the answer in the same manner as we did for the canary, the system would answer that the penguin travels by flying, which we know to be false. This is called exception handling.

We found that a semantic network was the best suited structure to our system. However even this structure had a few shortcomings. We therefore modified the structure to better fit our needs.

### C. *Extended Semantic Network*

The extended semantic network (ESN) is created from a common semantic network (SN). This modified structure allows us to represent any sentence, and therefore any knowledge that may be expressed by these sentences. An ESN has five types of vertices: entity, relationship, complement, entity modifier and relationship modifier.

An entity vertex represents the same concept that a vertex in a SN represents. The relationships represented by the edges of a SN are equivalent to the relationship vertices of the ESN.

The complement vertex is used to represent any type of complement that may be expressed in English. The complement vertex allows a simple and more elegant way to represent certain rules or limitations. For example, sometimes we have different relationships to model similar sentences, as in "The color of the sky is blue" and "The color of the sky is blue only on sunny days". A SN does not have a simple way to represent this type of exception. An ESN can represent both these sentences in a similar manner because the complement vertex is capable of adding more information about the main phrase.

The entity modifier vertex groups different entity vertices into a new vertex that represents a single more complex entity. Take the sentence "Robert's house is very pretty". Both "Robert" and "house" are entities for which more information could be available. It is therefore desirable to store them as such. However "Robert's house" is also an entity. An entity modifier vertex is able to represent this complex entity formed by two simpler entities. In a similar way, a relationship modifier will be linked to relationship vertices. Considering this, the diagram in Fig.2 represents an ESN created by our software for the same SN in Fig.1.

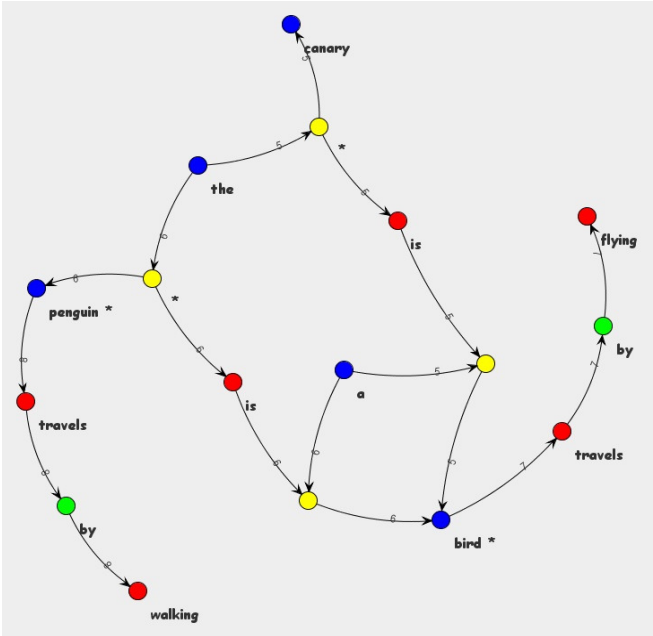


Fig. 2. An ESN representation of the same SN shown in Fig. 1. The vertex colors are: blue for entity, red for relationship, green for complement and yellow for modifiers (both entity and relationship)

Knowledge may also be inferred from an ESN in a manner similar to inference in a SN. The "is a" relationship edge in a SN is replaced by a "is" relationship vertex in an ESN. It is possible to infer how a canary travels by noting that a directed edge from the entity modifier vertex joining "the" and "canary" to an "is" relationship vertex exists. Once again, following the directed edge leaving the "is" vertex one arrives at a entity modifier vertex joining "a" and "bird". Therefore a canary is a bird. Applying a similar logic one determines how a canary travels.

An ESN can be defined through an extended semantic network grammar (ESNG), which is described by the following Wirth notation (7):

**esn** = "{" extent extrel "}" | "{" extent extrel extent "}" .
**exter** = extrel | extent .
**extrel** = genrel { "[" "compl" exter "]" } .
**extent** = genent { "[" "compl" exter "]" } .
**genrel** = "rel" | "(" relset ")" "rel" .
**relset** = "rel" { "rel" } .
**genent** = "ent" | "(" entset ")" "ent" .
**entset** = "ent" { "ent" } .
**ent** = "acorn" | "bird" | "cat" | "dog" | …
**rel** = "is" | "travels" | "lives" | …
**compl** = "as" | "by" | "and" | "of" | "in" | …

D. In this notation, "ent" represents an entity; "rel", a relationship and "compl", a complement. The "entset" and "relset" productions correspond to sets of entities and relationships, respectively. The "genent" and "genrel" productions represent an entity or a modified one, with the modified entity and the set of modifiers entities. The "extrel" and "extent" productions allow the insertion of complement phrases. Finally, the esn represents an English sentence, given the previous productions.

E. Based on the ESGN, it is possible to represent English sentences using a new representation that can be obtained from an ESN. For instance, the sentences in Fig. 2 can be defined using the ESNG as: {(the) penguin is (a) bird}, {(the) canary is (a) bird}, {bird travels [by flying]} and {penguin travels [by walking]}. As we can see, the ESNG allows a simple representation of English sentences, adding useful information about the semantics, like the presence of noun modifiers (with the round brackets), complementary phrases (with the square brackets) and sentence boundary (with the curly brackets).

Taking a longer sentence, for instance, "The electron was identified as a particle in 1897 by J. J. Thomson and his team of British physicists", we have the ESN shown in Fig.3:

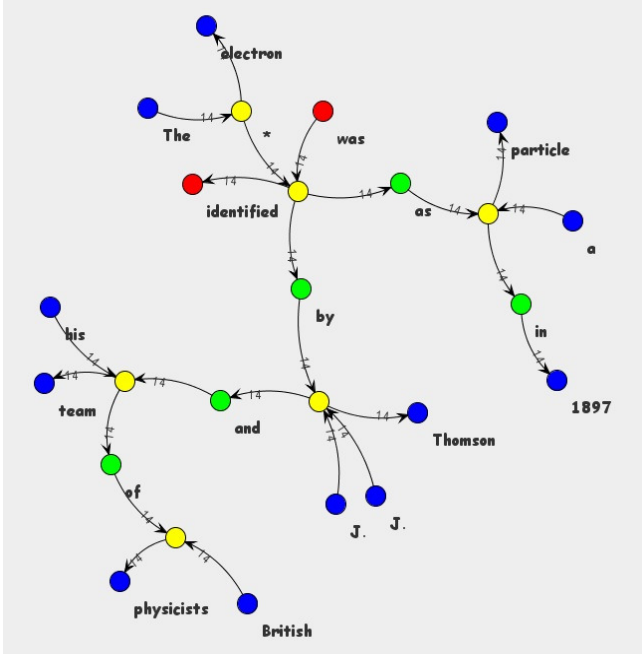


Fig. 3. An ESN representation of a longer sentence.

The corresponding ESNG representation of the ESN in Fig.3 is: {(The) electron (was) identified [as (a) particle [in 1897]] [by (J. J. Thomson) [and (his) team [of (British) physicists]]]}.

*F. Architecture*

The system is divided in two main modules: an input module (the *Writer module*) which is responsible for populating the database with knowledge gained by examining input sources; and an output module (the *Reader module*) which accesses the database in order to answer queries made by the user.

The general flow of the Writer module is outlined in the flowchart below.

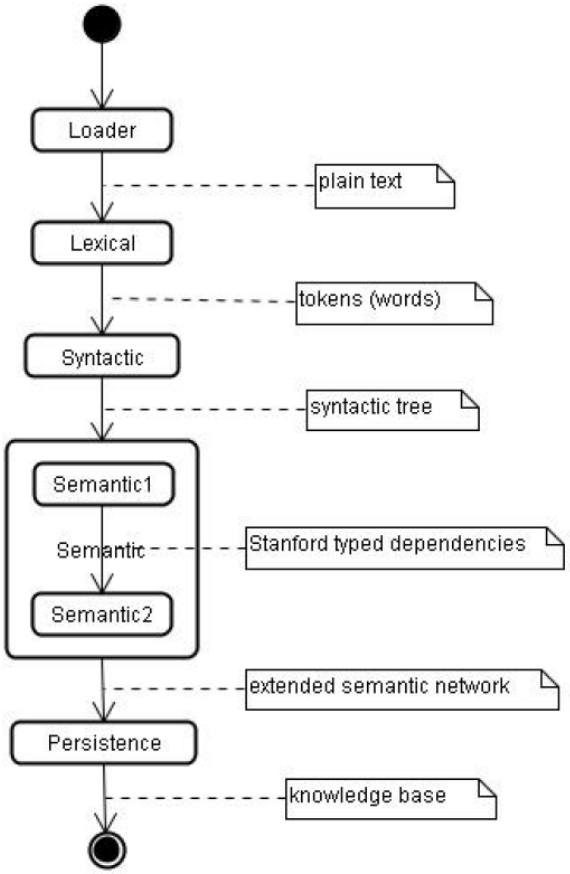


Fig. 4. Flowchart representing the Writer module of the Cube system.

The first component of the Writer module is the Loader. It loads the input texts, independently of the source types. This component isolates the source type from the system which, in turn, allows for different text sources to be processed in a similar way. This modular approach also simplifies the task of creating further functionalities to load data from other sources. Currently we are able to load text from PDF files and from HTML documents, so any web site may be processed. A special processor exclusively for Wikipedia was also developed. It includes the capability of processing links, treating the corresponding sites in a recursive manner. The depth to which this recursion is carried out can be controlled by the user.

The next three components (lexical, syntactic and semantic analyzers) have the usual functions. The lexical analyzer takes the text and splits it into tokens, which in our case represent words. This part is relatively simple and can be implemented using regular expressions. The syntactic analyzer transforms the tokens received from the lexical analyzer into a syntactic tree. A syntactic tree is a tree that represents the text in a sentence in a grammatically structured form. The words are characterized as being nouns, adjectives, etc. and are arranged on the tree based on their order in the sentence. Currently the Stanford Parser is being used to generate the syntactic tree.

The semantic analyzer converts the syntactic tree into the extended semantic network. To accomplish this goal, it uses the Stanford typed dependencies (8) generated by the parser. This functionality is implemented partly by the Stanford Parser and partly by a newly developed component in our software.

The last part of the Writer component is the Persistence module. It basically stores the ESN generated by the previous component in the knowledge base. Currently we use Hibernate (9) to store the ESN in a relational data base. Care must be taken when saving data to the database since the ESN generated for a given sentence must be joined with the existing database. It is important that when this is done, the vertices are not replicated in the database. In other words, there should only be one entity node for "canary", "penguin" and so on.

The Reader module allows the user to query the database. Currently the querying algorithms are relatively simple. We have not yet implemented algorithms for inferring knowledge from the ESN. However the system is already able to answer some simple queries. We have developed two user interfaces. One is in the form of a Java application that can be used to input information into the system as well as to query the database and obtain the answers and check the ESN in the database. The other is a web interface that allows querying and returns formatted answers. Fig.5 shows the Reader module components.

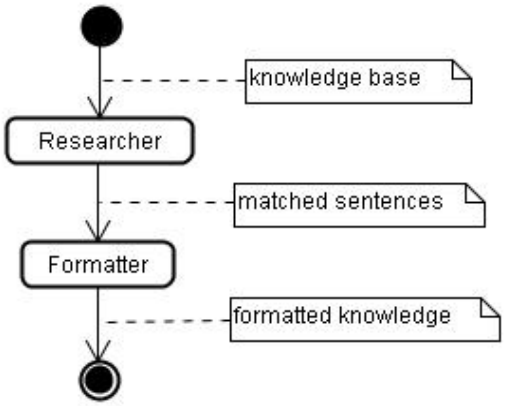


Fig. 5. A flowchart representing the Reader module.

The first Reader module component is the Researcher. This component receives the query provided by the user and then searches for sentences that contain the specified words. Since the ESNs contain data about the complete sentence, which consists of a unique identification for each sentence, we can return pieces of the stored ESN that contain only the desired sentences. Having found the sentences, this component reassembles the ESN in memory, creating the corresponding edge-vertices structure of the ESN.

The next component in the Reader module is the Formatter. This component recognizes knowledge in the sentences returned by the Researcher. For instance, imagine a user queries the word "electron". If the database contains an entity "electron" that is related to a composed entity "subatomic particle" through a relationship node that contains "is", as in Fig. 6, then the Formatter creates a Definition format, retrieving the term ("electron") and the definition ("subatomic particle") from the sentence. The resulting text, created to be displayed as a web page, is shown in Fig. 7.

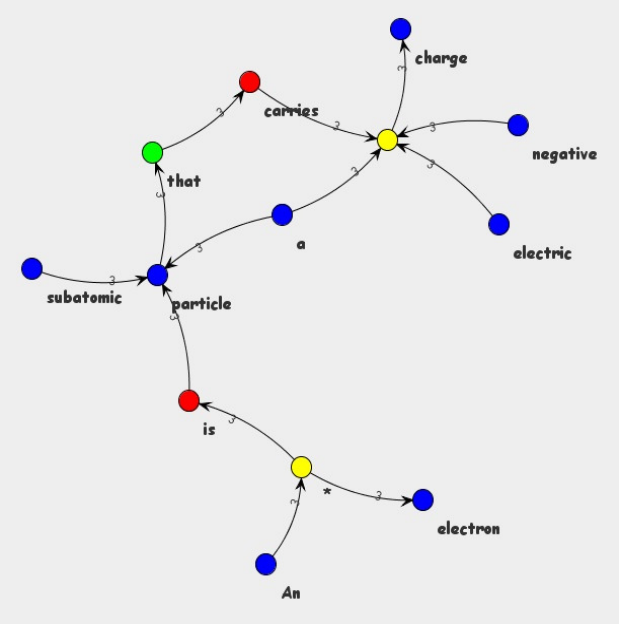


Fig. 6. ESN retrieved from the database for the query "electron".

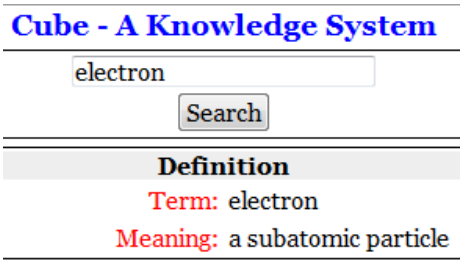


Fig. 7. The web result of a query for "electron".

A Property format was also developed. Take an ESN that contains an entity "color" connected to another entity "sky" by the complement "of" and to another entity "blue" by the relationship "is". This type of relationship describes a property: the main entity is "sky", which has a property "color" with value "blue". The Property format is returned as shown in Fig.8.

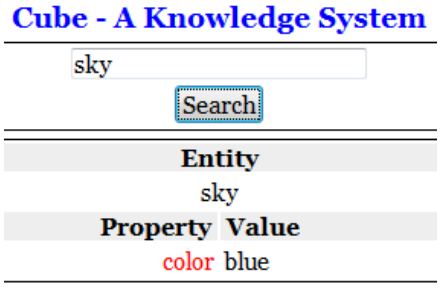


Fig. 8. The web result of a query for "sky".

The Formatter component is extensible and, therefore, other processing components can be incorporated in the system, amplifying its capabilities of finding knowledge from the ESN.

### *G. Cube as an application of adaptivity theory*

The Cube project has some aspects related with adaptivity theory (10). Its main component, the knowledge base, is a rule set. This rule set has a definition expressed by the Extended Semantic Network Grammar (ESNG). The system updates the ESN according to the sentences that it acquires from different possible sources, like PDF documents, web sites and so on. The updating process is not trivial and it has some details that require attention. First, entities can't be duplicated in the database. There must be a verification to merge a newly created ESN with the database ESN. Second, the algorithm must substitute pronouns and other indeterminations, because their presence in the knowledge base has no meaning. Finally, the system must take into account possible conflicts try to resolve them through modifications to the ESN. These last two requisites are not implemented yet on the current version of the program, but they are planned for future work.

The search and formatting process are based on the knowledge base. It is essentially a common application of adaptivity: when the system tries to retrieve some knowledge from the base, this process is done considering a structure that is modified dynamically, through a loading process from text documents. Beyond this, for the syntactic analyzer, we built a dynamic parser generator, that creates a parser (SLR or LR(1)) in execution time, given a grammar. This component permits updates on the grammar in execution time, without the need of recompiling the code in order to generate a new parser. This implementation is another application of adaptativity in this project.

However, developing this piece of software is not part of our current project, which is why we are using the Stanford Parser. In order to create our own complete solution with our own syntactic parser, however, a dynamic parser generator will be useful.

## IV. RESULTS

In order to test our system we loaded the Wikipedia web page for the entry "electron", using the previously described writer module. After the site has been processed, the database was updated with the sentences from that page. Searching for the term "electron" using our web interface, we obtain the results shown in Fig. 9.

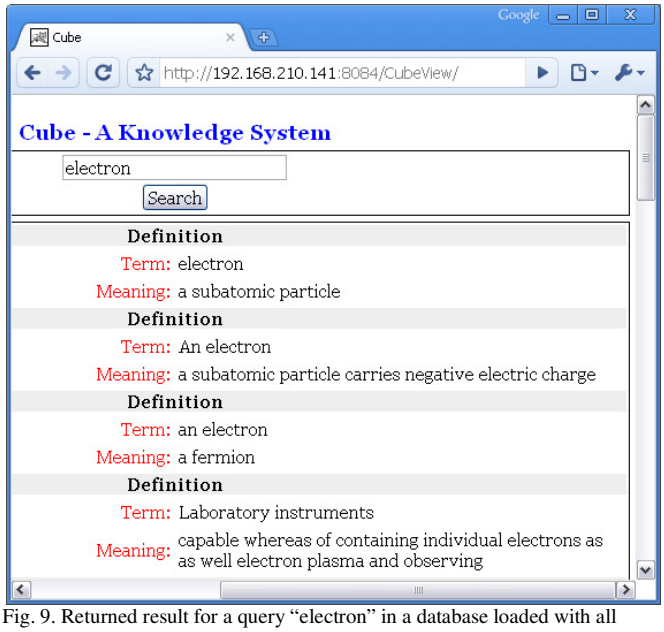


Fig. 9. Returned result for a query "electron" in a database loaded with all the information in Wikipedia web page for the entry "electron".

As can be seen, the system is capable of finding more than one definition for the word "electron". This occurs because the Formatter component analyzes all the sentences returned by the Researcher and verifies, for each of them, if it matches the expected format for a definition.

The process of reassembling the sentence is not yet completed and, therefore, some errors are present (the last definition in Fig.9 is an example). The conversion from an ESN to plain English is not trivial and it must follow English grammar rules. It is important to note that several of the observed mistakes are not caused by our system but are due to incorrect parsing of some sentences by the parser.

Although the Reader module is simple, it provides good results. The mechanisms to extract knowledge are very simple and currently lead to limited results. The describe Definition format is an example. Since not all occurrences of the verb "is" are used to define something, the system sometimes yields incorrect definitions. We believe however, that with little more logic this concept could yield to correct results in a more consistent manner.

The processing time to parse input data is still a bit high. For example, it takes a few minutes to process a Wikipedia page. This is mostly spent parsing the input data via the Stanford Parser.

## V. Conclusions

As was shown, the Cube system is still simple and in development. However it is already able to automatically extract knowledge from HTML and PDF documents. In order to store this knowledge, we developed a new semantic representation, the Extended Semantic Network (ESN). The ESN exhibits great flexibility and is capable of representing complex sentences written in English. The ESN has a small set of node types (entity, relationship, modifier, complement) but it is applicable to almost all the sentences in English, because of its generic concepts.

The entire system was intended to be as modular as possible, since this is very important for further development and adding new functionality. For example, adopting a new input source or a different parser, like the Link Grammar Parser (11), is relatively easy. We hope that by implementing the various features mentioned throughout this article and few others the Cube system will become even more useful. This means to generating better and faster results and incorporating new capabilities to provide more information and analyses. As part of our future work, we are studying the introduction of a timeline processing component (based on historical information and object hierarchy) and the addition of capabilities to process and interpret other languages, in special Portuguese.

## References

1. **Google.** Google Squared. [Online] [Cited: August 10, 2009.] http://www.google.com/squared.
2. **Wolfram Alpha**. [Online] [Cited: August 10, 2009.] http://www.wolframalpha.com.
3. **Feigenbaum, Edward.** The Age of Intelligent Machines: Knowledge Processing - From File Servers to Knowledge Servers. [Online] [Cited: November 30, 2009.] http://www.kurzweilai.net/meme/frame.html?main=/articles/art0098.html
4. **Stanford.** Stanford Parser. [Online] [Cited: September 21, 2009.] http://nlp.stanford.edu/software/lex-parser.shtml.
5. **Minsky, Marvin.** FRAMES. [Online] MIT, June 1974. [Cited: November 27, 2009.] http://web.media.mit.edu/~minsky/papers/Frames/frames.html.
6. **Sowa, John F.** *Semantic Networks.* s.l. : John Wiley and Sons, Inc., 1987.
7. **Wikipedia**. Wirth Syntax Notation. [Online] [Cited: November 30, 2009] http://en.wikipedia.org/wiki/Wirth_syntax_notation
8. **Manning, Marie-Catherine de Marneffe and Christopher D.** The Stanford typed dependencies representation. *COLING Workshop on Cross-framework and Cross-domain Parser Evaluation.* 2008.
9. **Hibernate.** Hibernate. [Online] [Cited: September 28, 2009.] https://www.hibernate.org/.
10. **Neto, J. J.** Adaptatividade e tecnologia adaptativa. *Revista IEEE América Latina.* 7, 2007, Vol. 5. (Special Edition – of Worshop de Tecnologia Adaptativa, WTA'2007)
11. **University, Carnegie Mellon.** *Link Grammar Parser.* [Online] [Cited: Novermber 30, 2009.] http://www.link.cs.cmu.edu/link/.