# The JUMP project: domain ontologies and linguistic knowledge @ work

P. Basile, M. de Gemmis, A.L. Gentile, L. Iaquinta, and P. Lops

Dipartimento di Informatica
Università di Bari
Via E. Orabona, 4 - 70125 Bari - Italia
{basilepp,degemmis,al.gentile,l.iaquinta,lops}@di.uniba.it

**Abstract.** The JUMP project aims at bringing together the knowledge stored in different information systems in order to satisfy information and training needs in knowledge-intensive organisations. Electronic Performance Support Systems provide help, advices, demonstrations, or any other informative support that a user needs to the accomplishment of job tasks in her day-to-day working environment. The paper describes the JUMP framework, which is designed to offer multiple ways for the user to query the knowledge base resulting from integration of autonomous legacy systems. Semantic Web languages and technologies are used throughout the framework to represent, exchange and query the knowledge, while Natural Language Processing Techniques are implemented to understand natural language queries formulated by the user and provide consistent and satisfying results.
**Keywords:** Semantic Web, Interoperability, Word Sense Disambiguation, Entity Recognition

## 1 Introduction

The JUMP project [1] aims at developing an EPSS *(Electronic Performance Support System)* capable of intelligent delivery of contextualized and personalized information to knowledge workers acting in their day-to-day working environment on non-routinary tasks. While generic queries can be easily fulfilled by means of standard information retrieval tools, such as general purpose search engines, the scenario is more difficult if the search goal concerns grey information stored in various forms and spread in different company knowledge bases, managed by different applications, all running within the company intranet. It is the case of users knowledgeable w.r.t. the IT infrastructure and that already have the background knowledge necessary to achieve most of the task they are involved in, but not being expert of all the domains in which the task to be achieved spans. Tasks of this kind are neither generally codified in corporate

[1] JUst-in-tiMe Performance support system for dynamic organizations, co-funded by POR Puglia 2000-2006 - Mis. 3.13, Sostegno agli Investimenti in Ricerca Industriale, Sviluppo Precompetitivo e Trasferimento Tecnologico

procedures nor completely new to the worker. Above all, those tasks are by no means solvable, in terms of information retrieval, by a standard Internet search. Any brute-force approach like Google desktop search can solve the problem in this case and, even if it could, the result would never take into account the connections existing between the various sources. An EPSS aims at supporting information needs spanning through multiple knowledge bases, namely all the available information systems in the company, be them formalized or not, including binary documents such as video or audio streams. It acts as an agent gluing together the different sources by means of semantic connections, and provides the user with contextualized and personalized information tied to both the task being accomplished and to her characteristics. On the basis of an accurate and formalized description of user's features and of those of the software tool she is using, as well as the textual information describing the task being accomplished (for example, the text of an e-mail just received), the EPSS should select relevant items from the KBs, ranking them according to the user profile, and provide them in a list to the user who will eventually give a feedback about the relevance of the provided information. The JUMP system has been designed to achieve this goal by means of a centralized recommendation system that takes advantage of a shared ontology describing the various knowledge bases and advanced Natural Language Processing (NLP) techniques to handle natural language requests.

The paper is organized as follows: after giving a general description of JUMP framework focusing on abstract layers of its architecture and the underlying shared ontology, in section 3 we give a detailed description of the Content Analyzer Module encapsulated in the framework, which provides NLP tools. In section 4 we argument the project prototype and conclusions to work, anticipating possible future work.

## 2 JUMP: Ontology-centric Architecture

The system general architecture is depicted in Figure 1, where the central component (JUMP-EPSS) acts as a hub of many autonomous peripheral systems. The involved systems in the current implementation are a Human Resources Managements system (HRMS), an Enterprise Resource Planning (ERP), a Document Management system (DMS) and a Learning Management system (LMS), but the design of the platform is such that new systems can be added as they become available.

### 2.1 Modularized Design

The basic design idea of the JUMP project is to encourage the loosely coupling among framework components according to a Service Oriented Architecture perspective so that the framework has the ability to seamlessly add information sources or peripheral systems, each one based on different technologies, programming languages and knowledge representation metaphors. This has lead to adopting standard languages and protocols when designing the interfaces that

each of the systems participating in JUMP has to implement in order to expose search services. The communication level between JUMP and the ancillary sys-

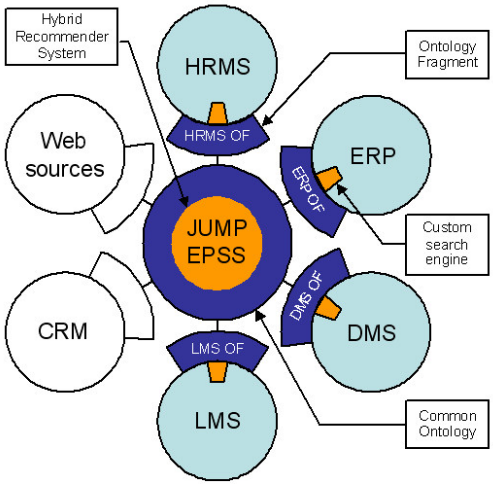


**Fig. 1.** Sketch of the JUMP system architecture

tems is designed to exchange both metadata about relevant items stored in the subsystems and the items themselves. While the items considered here (which are the results JUMP can present to the user) are generic binary objects ranging from email addresses to audio/video streams, the metadata about them are expressed through Semantic Web technologies; to make this possible, some OWL [2] ontologies about the items have been created, in order to structure specific domain knowledge and instantiate resources to describe the stored items.

### 2.2 Ontologies in the JUMP project

An ontology, following Grubers widely accepted definition [6], is a shared formalization of a conceptualization. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. In order to define an ontology it is necessary to choose a formalism, to use this formalism to encode the conceptualization that the applications are going to use, and to make this conceptualization shared, i.e. ensure that the ontology is used consistently by all the systems involved. To define the JUMP ontology we adopted OWL as representation language and Description Logics [1] was consequently adopted as the underlying formalism. Separate ontology fragments have been handcrafted using the Protege editor [3]

[2] http://www.w3.org/2004/OWL/
[3] http://protege.stanford.edu

in order to represent the most important concepts inside each of the involved systems. Ontologies have been then designed bottom-up in order to reflect the semantics of the underlying databases and coded functional processes as much as possible, but not aiming at a total ontological replication of the knowledge bases. After developing the single Ontology Fragments (OF), they have been divided into system specific ontologies and upper ontologies; these upper ontologies are the part of the OFs that the JUMP system should use when formulating queries for the subsystems. The Shared Ontology (SO) is therefore the union of all the upper OFs plus all the relations and concepts that are specific to the JUMP system; since some concepts are repeated across systems, the creation of the SO is the point in which alignment techniques have to be used in order to simplify and generalize the query writing phase of the search. The concept in SO are annoted using lexical concept that are exploited by Word Sense Disambiguaiton algorithm described in Section 3.1. The Ontology Fragments are aligned manually using the concepts in the Shared Ontology. The single Ontology Fragments are populated automatically creating a mapping between each legacy system's DBMS and each fragment.

### 2.3 JUMP EPSS Core: External and Internal Interactions

JUMP architecture has two abstract layers, as showed in figure 2, logically organized as a stack:

- *User Interface Layer*
- *Application Domain Layer*

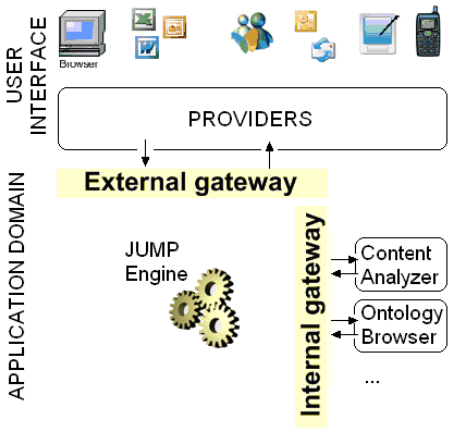


**Fig. 2.** JUMP Engine and Framework Layers

The *User Interface Layer* has been designed to support both online and offline client. It supports several devices: we handle both *untrusted devices*, such

as web browser or Office tools (Word/Excel), for which we predisposed a login procedure and *trusted devices*, such as mobile devices (e.g. smart phone) or email client for which the system is able to identify the user without any login procedure. The application domain layer has a software layer, namely *external gateway*, that communicates with all user devices through their specialized provider. The *JUMP Engine* elaborates all requests making use of specialized modules to resolve specific requests (e.g. plain text is sent to *Content Analyzer Module*, described in Section 3, in order to capture its semantics). Communication between the *JUMP Engine* and all specialized modules is implemented through a software layer, namely *internal gateway*.

## 3 NLP Processes in the Content Analyzer Module

Since user's requests are formulated by using natural language, Natural Language Processing (NLP) techniques are adopted in order to covert the original requests into an internal representation processable by the JUMP system. The *Content Analyzer Module* is devoted to this task: it extracts relevant concepts from the text describing the request and build an internal data structure called Bag-Of-Concepts (BOC). The goal is to include semantics in the process and to overcome well-known problems in text processing, such as polysemy, due to the use of keywords. The BOC structure contains two type of concepts:

1. relevant *linguistic* concepts recognized in the text by a Word Sense Disambiguation process [8] exploiting external linguistic knowledge-bases such as the WordNet lexical database [9];
2. relevant *domain* concepts extracted by a Named Entity Recognition (NER) process. The NER process is guided by JUMP ontology.

The implemented NLP process also includes operations preliminary to the BOC extraction step, such as:

- Text normalization: the original text is modified to prepare it for the following steps (for example, all formatting characters are removed);
- Tokenization: it is the process of split up input a string into tokens;
- Stop words elimination: all commonly used words are deleted;
- Stemming: it is the process of reducing inflected (or sometimes derived) words to their stem. In our project we adopt the *Snowball stemmer* [4];
- POS-tagging: it is the process of assign a part-of-speech to each token. We develop a JAVA version of *ACOPOST tagger* [5] using Trigram Tagger T3 algorithm. It is based on Hidden Markov Models, in which the states are tag pairs that emit words;
- Lemmatization: it is the process of determining the lemma for a given word. We use WordNet Default Morphological Processor (included in the WordNet

[4] http://snowball.tartarus.org/
[5] http://acopost.sourceforge.net/

distribution) for English. For the Italian language, we have built a different lemmatizer that exploits the *Morph-it!* morphological resource [6].

As final output each word in the original document is enriched with syntactic and semantic information collected during all the steps. In the two following subsections (3.1 and 3.2) we provide more details about the process of BOC extraction process.

### 3.1 JIGSAW: Word Sense Disambiguation

The goal of a WSD algorithm consists in assigning a word $w_i$ occurring in a document $d$ with its appropriate meaning or sense $s$, by exploiting the *context* $C$ in where $w_i$ is found. The context $C$ for $w_i$ is defined as a set of words that precede and follow $w_i$. The sense $s$ is selected from a predefined set of possibilities, usually known as *sense inventory*. In the proposed algorithm, the sense inventory is obtained from WordNet. JIGSAW is a WSD algorithm based on the idea of combining three different strategies to disambiguate nouns, verbs, adjectives and adverbs. The main motivation behind our approach is that the effectiveness of a WSD algorithm is strongly influenced by the POS tag of the target word. An adaptation of Lesk dictionary-based WSD algorithm has been used to disambiguate adjectives and adverbs [2], an adaptation of the Resnik algorithm has been used to disambiguate nouns [10], while the algorithm we developed for disambiguating verbs exploits the nouns in the *context* of the verb as well as the nouns both in the glosses and in the sentence examples that WordNet utilizes to describe the usage of a verb. JIGSAW takes as input a document $d = (w_1, w_2, \ldots, w_h)$ and returns a list of WordNet synsets $X = (s_1, s_2, \ldots, s_k)$ in which each element $s_i$ is obtained by disambiguating the *target word* $w_i$ based on the information obtained from WordNet about a few immediately surrounding words. We define the *context* $C$ of the target word to be a window of $n$ words to the left and another $n$ words to the right, for a total of $2n$ surrounding words. The algorithm is based on three different procedures for nouns, verbs, adverbs and adjectives, called $JIGSAW_{nouns}$, $JIGSAW_{verbs}$, $JIGSAW_{others}$, respectively. A short description of procedures $JIGSAW_{nouns}$ and $JIGSAW_{verbs}$ follows, more details about all the procedures and experiments are reported in [3].

***JIGSAW*$_{nouns}$** The procedure is obtained by making some variations to the algorithm designed by Resnik for disambiguating noun groups. Given a set of nouns $W = \{w_1, w_2, \ldots, w_n\}$, obtained from document $d$, with each $w_i$ having an associated sense inventory $S_i = \{s_{i1}, s_{i2}, \ldots, s_{ik}\}$ of possible senses, the goal is assigning each $w_i$ with the most appropriate sense $s_{ih} \in S_i$, according to the *similarity* of $w_i$ with the other words in $W$ (the context for $w_i$). The idea is to define a function $\varphi(w_i, s_{ij})$, $w_i \in W$, $s_{ij} \in S_i$, that computes a value in $[0, 1]$ representing the confidence with which word $w_i$ can be assigned with sense $s_{ij}$. $JIGSAW_{nouns}$ differs from the original algorithm by Resnik in several ways.

[6] http://sslmitdev-online.sslmit.unibo.it/linguistics/morph-it.php

First, in order to measure the relatedness of two words we adopted a modified version of the Leacock-Chodorow measure [7], which computes the length of the path between two concepts in a hierarchy by passing through their *Most Specific Subsumer* (MSS). In our version, we introduced a constant factor *depth* which limits the search for the MSS to *depth* ancestors, in order to avoid "poorly informative MSSs". Moreover, in the similarity computation, we introduced both a Gaussian factor $G(pos(w_i), pos(w_j))$, which takes into account the distance between the position of the words in the text to be disambiguated, and a factor $R(k)$, which assigns $s_{ik}$ with a numerical value, according to the frequency score in WordNet (more importance is given to the synsets that are more common than others). This algorithm considers the words in $W$ pairwise. For each pair ($w_i$,$w_j$), the most specific subsumer $MSS_{ij}$ is identified, by reducing the search to *depth* ancestors, at the most. Then, the similarity $\textsc{Sim}(w_i, w_j, depth)$ between the two words is computed. $MSS_{ij}$ is considered *as supporting evidence* for those synsets $s_{ik}$ in $S_i$ and $s_{jh}$ in $S_j$ that are descendants of $MSS_{ij}$. The amount of support contributed by the pairwise comparison is the similarity value $\textsc{Sim}(w_i, w_j, depth)$, weighted by a gaussian factor that takes into account the position of $w_i$ and $w_j$ in $W$ (the shorter is the distance between the words, the higher is the weight). The value $\varphi(i, k)$ assigned to each candidate synset $s_{ik}$ for the word $w_i$ depends on both the amount of support it received and a factor that takes into account rank of $s_{ik}$ in WordNet, i. e. how common sense $s_{ik}$ is for the word $w_i$. The synset assigned to each word in $W$ is the one with the highest $\varphi$ value. More details about both the procedure and the computation of the similarity value are reported in [3].

***JIGSAW*$_{verbs}$** We define the *description* of a synset as the string obtained by concatenating the gloss and the sentences that WordNet uses to explain the usage of a synset. First, $JIGSAW_{verbs}$ includes, in the context $C$ for the target verb $w_i$, all the nouns in the window of $2n$ words surrounding $w_i$. For each candidate synset $s_{ik}$ of $w_i$, the algorithm computes $nouns(i, k)$, that is the set of nouns in the description for $s_{ik}$. Then, for each $w_j$ in $C$ and each synset $s_{ik}$, the following value is computed:

$$max_{jk} = max_{w_l \in nouns(i,k)} \{\texttt{sim(}w_j\texttt{,}w_l\texttt{,}depth\texttt{)}\} \tag{1}$$

where $\texttt{sim(}w_j\texttt{,}w_l\texttt{,}depth\texttt{)}$ is the same similarity measure in $JIGSAW_{nouns}$. In other words, $max_{jk}$ is the highest similarity value for $w_j$ wrt the nouns related to the $k$-th sense for $w_i$. Finally, an overall similarity score among $s_{ik}$ and the whole context $C$ is computed:

$$\varphi(i, k) = R(k) \cdot \frac{\sum_{w_j \in C} G(pos(w_i), pos(w_j)) \cdot max_{jk}}{\sum_h G(pos(w_i), pos(w_h))} \tag{2}$$

where both $R(k)$ and $G(pos(w_i), pos(w_j))$, that gives a higher weight to words closer to the target word, are defined as in $JIGSAWnouns$. The synset assigned to $w_i$ is the one with the highest $\varphi$ value.

### 3.2 Named Entity Recognition Step

The Named Entity Recognition (NER) task has been defined in the context of the Message Understanding Conference (MUC) as the capability of identifing and categorizing entity names, defined as instances of the three types of expressions: entity names, temporal expressions, number expressions [5]. Further specializations of these top level classes have been proposed [11] and general purpose lists of Named Entities are publicly available and incorporated e.g. within well-known Text Processing Software, as GATE (General Architecture for Text Engineering) [4], to give a popular example. However, for the aim of JUMP project we cannot rely on general purpose gazetteers to perform the step of Named Entity Recognition, due to specificity of categories and their instances for this particular project. For this reason we developed a simple algorithm to recognize entities using as gazetteers a domain ontology: we tag each token in the original document with the ontology class value if it represents an instance of that class in the domain ontology. The idea of the algorithm follows. Given $C = \{C_1, C_2, \ldots, C_n\}$ the set of classes in the domain ontology, for each class $C_k$ we consider the set $P = \{p_1, p_2, \ldots, p_m\}$ of properties belonging to $C_k$. Given $T = \{t_1, t_2, \ldots, t_s\}$ the list of tokens obtained from document $d$, for each token $t_j$ we consider a window of $h$ following tokens. The algorithm checks for each $C_k$ if value of any combination of $t_j, \ldots, t_{j+h}$ matches with the value of any $p_m$, for all instances of $C_k$, and assigns to $t_j$ the correspondent label. The search is done beginning from longer combinations of tokens and in the worst case it ends without any class annotation for the single token $t_j$. Taking advantage of semantic information provided by ontology, we can simply obtain relations between all entities found in the text, exploiting the object properties defined by the ontology, without added computational cost.

### 3.3 Concept-based Text Representation

Both the WSD procedure and the NER process are fundamental to obtain a concept-based text representation that we called Bag-Of-Concepts (BOC). In this model, a vector of concepts (WordNet synsets or named entities) corresponds to a document, instead of a vector of keywords. Therefore, each document (for example, an email text) representing the user request is converted in a BOC structure in order to be processed by the JUMP Engine. A more formal description of the BOC text representation follows. Assume that we have a document $d_n$ (corresponding to a user's request $n$)processed by the *Content Analyzer Module*. The document is converted into the following BOC structure:

$$d_n = \langle (t_{n1}, w_{n1}), (t_{n2}, w_{n2}), \ldots, (t_{n|V|}, w_{n|V|}) \rangle$$

where:

- $t_{nk}$ is the $k$-th token (synset or named entity) recognized in document $d_n$ by NLP procedures;
- $V$ is the set of distinct tokens recognized in $d_n$;

– $w_{nk}$ is the weight representing the informative power of token $t_{nk}$ in document $d_n$.

In other words, a text is represented by a vector of pairs *(token, weight)*, where tokens are recognized from keywords in the text by NLP procedures which assign each token with a numerical score representing the discriminatory power of that token in the text. Weights can be computed in different ways for synsets and named entities. For example, we consider a user who is going to prepare a technical report for a project. She has technical skills but no idea about how to compile such kind of document. She queries the Jump System, using the following expression $q$: *"I have to prepare a technical report for the Jump project"*. The Jump Engine passes the user's request $q$ to the Content Analyzer module that processes the document in order to both disambiguate the task to be accomplished, by using the WSD procedure, and to recognize entities potentially useful for the task. The final output of this stage is $q$ represented according to the BOC model:

$$q = \langle (01704982, 0.75), (06775158, 0.85), (Jump, 0.99), (00746508, 0.80) \rangle$$

where both synset identifiers of the concepts recognized in the text and named entities are used in the BOC structure. For example, the verb "prepare" has been disambiguated as the synset reported below:

`01704982 (verb.creation) prepare -- (to prepare verbally, either for written or spoken delivery; ‘‘prepare a report’’`

which identifies the task to be accomplished. This structure is used to query the Shared Ontology. As a result, the Jump Engine provides all istances of the concepts *technical report* and *project* and relations intercourring among these istances and istances of different classes of the ontology. In the example, the system returns the list of partners and documents related to the Jump project, and the list of technical reports already written for other projects.

## 4 Conclusions and Future Work

In this paper we presented the design and initial implementation of a framework for knowledge sharing within knowledge-intensive organizations by personalized information retrieval. The user current task and background knowledge are used to fulfill informative request. NLP and shared domain ontology are exploited to semantic interpretation of user request in order to query legacy knowledge bases. The JUMP project is an ongoing project; so far, a prototype implementing what presented in this paper has been developed as an internal proof of concept to verify that interfacing systems through the JUMP framework is feasible and useful even outside the project scope itself.Other features currently under development in the prototype are related to the different possible interfaces that the user can exploit in order to query the system. In particular, the possible interactions that have been depicted so far include support to Microsoft IBF (Information Bridge Framework) smart tags (in PUSH mode, i.e. without the user explicitly requesting services), and SMS and email support (in PULL mode, i.e. as answer to a

user explicit request). Since the user is likely to be a fairly experienced computer user and not a computer programmer, the query is expected to be a simple text query, not different from a normal query that could be issued against a standard query engine such as Google or Yahoo. As future work, we intend to improve the WSD algorithm and the NER process by including the shared ontology, enriched with external links to WordNet, into these processes.

## Acknowledgements

This work has been co-funded by Regione Puglia, Italy, through the research funding program named POR Puglia 2000-2006 - Mis. 3.13, Sostegno agli Investimenti in Ricerca Industriale, Sviluppo Precompetitivo e Trasferimento Tecnologico.

## References

1. F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. S. Banerjee and T. Pedersen. An adapted lesk algorithm for word sense disambiguation using wordnet. In *CICLing '02: Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing*, pages 136–145, London, UK, 2002. Springer-Verlag.
3. P. Basile, M. de Gemmis, A.L. Gentile, P. Lops, and G. Semeraro. Jigsaw algorithm for word sense disambiguation. In *SemEval-2007: 4th Int. Workshop on Semantic Evaluations*, pages 398–401. ACL press, 2007.
4. H. Cunningham, Y. Wilks, and R.J. Gaizauskas. Gate: a general architecture for text engineering. In *Proceedings of the 16th conference on Computational linguistics*, pages 1057–1060, Morristown, NJ, USA, 1996. Association for Computational Linguistics.
5. R. Grishman and B. Sundheim. Message understanding conference- 6: A brief history. In *COLING*, pages 466–471, 1996.
6. T. R. Gruber. A translation approach to portable ontology specifications. In *Knowledge Engineering, 5(2)*, pages 199–220. Academic Press, 1993.
7. C. Leacock and M. Chodorow. *Combining local context and WordNet similarity for word sense identification*, pages 305–332. MIT Press, 1998.
8. C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*, chapter 7: Word Sense Disambiguation, pages 229–264. MIT Press, Cambridge, US, 1999.
9. G. A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.
10. P. Resnik. Disambiguating noun groupings with respect to WordNet senses. In *Proc. of the 3th Workshop on Very Large Corpora*, pages 54–68. ACL, 1995.
11. S. Sekine, K. Sudo, and C. Nobata. Extended named entity hierarchy. In *Proceedings of the LREC-2002*, 2002.