THE JUMP PROJECT: PRACTICAL USE OF SEMANTIC WEB TECHNOLOGIES IN EPSS SYSTEMS

Giovanni Semeraro, Ignazio Palmisano, Nicola Abbattista
Dipartimento di Informatica, Università degli Studi di Bari, Via Orabona 4, 70126 Bari, Italy

Silverio Petruzzellis
Cezanne Software S.p.A. - Via Amendola 172/C, I70126, Bari, Italy

Keywords: Semantic Web, Interoperability, SPARQL.

Abstract: The JUMP project aims at bringing together the knowledge stored in different information systems in order to satisfy knowledge and training needs in knowledge-intensive organisations. EPSS systems are meant to support a user in taking decisions, leveraging different information sources that are available. The JUMP framework is designed to offer multiple ways for the user to request information from a knowledge and document base itself as the result of the integration of indipendent systems. In order to simplify communication and maximize knowledge sharing and standardization of languages and protocols, Semantic Web languages and technologies are used throughout the framework to represent, exchange and query the knowledge stored in each part of the framework.

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 like Google, very often the same power is difficult to be achieved when the goal of those searches are information stored in various company databases, 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 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. No 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. The goal of an EPSS aiming 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, is to act as an agent gluing together the different sources by means of semantic connections, and providing the user with contextualized and personalized information tied to the task being accomplished and to his/her characteristics. On the basis of an accurate and formalized description of the user's features and of those of the software tool he/she is using, as well as the textual information describing the task being accomplished, like for example the text of an e-mail just received, the EPSS should select relevant material from the KBs, ranking them according to the user profile, and provide them in a list to the user who will eventually give his/her feedback related to the validity of the information provided. The JUMP system has been designed to achieve this goal by means of a centralized recommendation system

[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

Semeraro G., Palmisano I., Abbattista N. and Petruzzellis S. (2007).
THE JUMP PROJECT: PRACTICAL USE OF SEMANTIC WEB TECHNOLOGIES IN EPSS SYSTEMS.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - AIDSS*, pages 428-431
DOI: 10.5220/0002383304280431

that takes advantage of a common ontology describing the various knowledge bases.

2 DESIGN & IMPLEMENTATION

The system general architecture is depicted in Figure 1, where the central system (JUMP-EPSS) acts as a hub of many independent external systems. The involved systems in the current implementation are a Document Management system, an ERP, a Learning Management system and a HRM system, but the design of the platform is such that new systems can be easily added to the platform as they become available.

2.1 Modularized Design

The basic design idea of the JUMP project is to make the various portions of the framework as loosely coupled as possible, since one of the requirements of the framework is the ability to seamlessly add new information sources or external 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 each of the systems participating in JUMP has to implement.

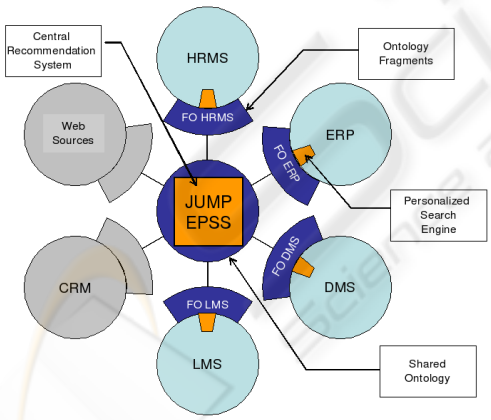

Figure 1: Sketch of the JUMP system architecture.

The communication level between JUMP and the ancillary systems 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 Gruber's widely accepted definition (Gruber, 1993), is a shared formalization of a conceptualization, which means that 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.

OWL was used as the representation language for the JUMP project and Description Logics (Baader et al., 2003) was consequently adopted as the underlying formalism.

2.2.1 Ontology Fragment Creation

Separate ontology fragments have been handcrafted using the Protégé editor[3] in order to represent the most important concepts inside each of the involved systems. By "most important" we mean here the concepts each system domain expert considered of paramount importance when representing the system internal knowledge, so that external queries could be answered by using the ontology fragment. 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.

One of the reasons for that is the necessity to find the right tradeoff between the effort spent in designing the ontologies and the advantages coming from their usage; for this reason, the line has been drawn so that all the relations and concepts that are relevant when querying the system could be expressed through ontologies, therefore excluding all those concepts and relations that are only used for implementation purposes or are not likely to be used in a search.

2.2.2 Ontology Fragment Integration

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

[2] http://www.w3.org/2004/OWL/

[3] http://protege.stanford.edu

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 JUMP component has therefore to interface with a generic service, sketched in Figure 2, knowing which subset of concepts and relations are understood by the service, and has to query it in order to discover which items that could be relevant for the user the service can return.

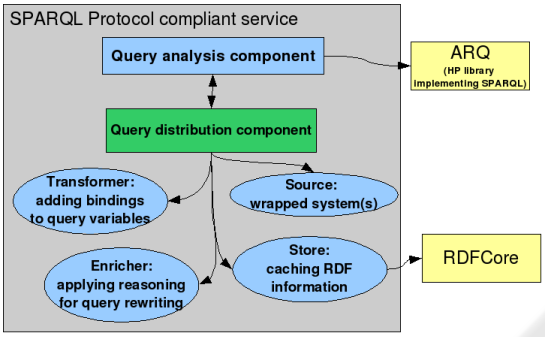

Figure 2: Sketch of a subsystem architecture.

2.3 Communication Protocols

Being each service based on an existing system that can potentially use any technology or programming paradigm, our design choice was to model services as Web Services or HTTP services, depending on the specific necessity. More in detail, we identified two possible requests to a service:

- Find all relevant metadata for a request, i.e. answer a SPARQL query, using the system specific ontology where possible in order to refine the query
- Return a specified item upon request (items can be binary objects in any format)

The first request, in practice, can be answered through the implementation of the SPARQL Protocol[4] (which amounts to the ability to answer a SPARQL query through an HTTP or SOAP request), while the second request can be easily filfilled through the use of a simple HTTP service, that will answer to a GET operation with suitable arguments; the reason to use HTTP instead of a Web Service interface is that, as already said, the items to be presented can be generic textual or binary objects, in some cases quite large in size (w.r.t the E-Learning system, the size of a resource, say a video tutorial for example, can reach hundreds of megabytes), and therefore the additional overload of SOAP serialization can be a serious performance bottleneck, not providing any actual advantage over the HTTP solution.

The internal architecture of an ancillary system is responsibility of the single developer; however, the JUMP project aims at integrating the four abovementioned systems in the platform proposing and sharing a common architectural design. This architecture is sketched in Figure 2.

The RDFCore component represented in the subsystem architecture is a RDF storage system built on top of the Jena Toolkit; it offers a SPARQL Protocol implementation that, depending on the specific subsystem architecture, can be directly used to implement the subsystem interface to the JUMP framework or can be used to simplify data access. The solution is described in more detail in (Stoermer et al., 2006).

2.4 Dataflow Design Metaphor

The architecture described here is based on the metaphor of data pipes: by reusing an idea presented in (Palmisano et al., 2006), we model the answer to the input query as a data flow coming out of the conjunction of many data streams, each one originating into a *Store* and corresponding to a URI in the query *Dataset*. Since it is possible that different parts of the query refer to different portions of the knowledge base (i.e. some variables of the SPARQL query can be fully binded in a subset of the dataset), we wrap the query execution in a component called Query Analyzer, which has responsibility to split or rewrite the query through the different portions of the knowledge base, which are then queried independently.

The knowledge necessary to redirect a pattern to a specific portion of the dataset must be available for the Query Analyzer component, in order to rewrite the query for each specific *Locator* that will feed the results. This information can be represented as a list of URIs associated to each section of the knowledge base, where each URI corresponds to one of the predicates of the statements contained in that section of the knowledge base, or to the types of the individuals described therein; moreover, it is possible to do the same thing with a set of namespaces instead of full URIs, but this implies that different namespaces are to be used for different sections of the knowledge base; to prevent clashing, it is possible to just think of joining two pipes with the same set of namespaces or URIs (or where one has a superset of the other),

[4] http://www.w3.org/TR/rdf-sparql-query/

however this diminishes the effectiveness of the optimizations presented so far. It is therefore important to choose the partitioning of the knowledge base in order to simplify these issues.

3 PROTOTYPE

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 (one of the side effects of the explicit knowledge representation, in fact, is the ability of the systems to implement interfaces to other frameworks, e.g. frameworks based on HR-XML[5] standard for Human Resource management systems).

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[6] or Yahoo[7]. However, we are exploring the possibility of offering the user a more expressive language, e.g. giving the ability to specify the class of results to which he/she is interested, like in:

```
Deliverable(FP 6 EU Project)
```

which carries the extra information that the resource we are looking for, besides being related to the keywords *European Project 6th Framework Programme*, belongs to the class Deliverable or one of its subclasses.

4 CONCLUSIONS

In this paper we presented the design and initial implementation steps of a framework for knowledge and document sharing between different systems, aimed at satisfying user information needs on the base of his/her current task and background knowledge, through the use of Sematic Web technologies. We also presented some design issues and the solutions we adopted to overcome the problems found so far. An initial prototype and some current and future developments of the design were also presented.

[5]http://www.hr-xml.org/

[6]www.google.com

[7]www.yahoo.com

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

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., editors (2003). *The Description Logic Handbook*. Cambridge University Press.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. In *Knowledge Engineering, 5(2)*, page 199220. Academic Press.

Palmisano, I., Redavid, D., Iannone, L., Semeraro, G., Degemmis, M., Lops, P., and Licchelli, O. (2006). A RDF-Based Framework for User Profile Creation and Management. In Gabrys, B., Howlett, R. J., and Jain, L. C., editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 4253 of *Lecture Notes in Artificial Intelligence*, pages 606–613. Springer.

Stoermer, H., Palmisano, I., Redavid, D., Iannone, L., Bouquet, P., and Semeraro, G. (2006). Contextualization of a RDF Knowledge Base in the VIKEF Project. In *Proceedings of the 9th International Conference on Asian Digital Libraries, Shiran Kaikan (Kyoto, Japan), 27-30 November (to appear)*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg.