Involving Non Knowledge Base Experts With the Development of Ontologies

Vlad Nicolicin-Georgescu*, Vincent Bénatier
SP2 Solutions
La Roche Sur Yon, France
vladgeorgescun@sp2.fr, vbenatier@sp2.fr

Rémi Lehn, Henri Briand
*Ecole Polytechnique de l'Université de Nantes
Nantes, France
remi@fc.univ-nantes.fr, henri.briand@univ-nantes.fr

***Abstract* - The paper presents an approach to ontology development with the help of regular, non technical users. The specific objective is the construction of a software ontology with a much higher level of detail (e.g., patch version or software version compatibility) than existing propositions like OpenCyc. To this end, we need the feedback of software experts and users. The problem is that these are not knowledge experts with a background in working with ontology concepts, as required by the actual ontology development solutions. Our strategy is to provide an intuitive online platform through which users can provide feedback about their software configurations without the perquisites of ontology modeling. The platform, called *TimSys*, is linked with the ontology model via mapped data bases and it represents a bridge between the technical and non technical knowledge base worlds.**

Keywords – Ontology, Information System, Software, Decision Support System

I. INTRODUCTION

The evolution of *information systems* (IS) lead to complex description of their architectures, from hardware resources to installed software. As the number of software vendors increased exponentially, so did the number of offered functionalities and services. It is assessed that up to 90% of the requested functionalities is already available with existing applications [1]. The variety of software products implies an increased number of problems, from bugs to product incompatibility. These are referenced in different non or semi-structured sources, such as readme documents or technical forums. Integration propositions such as Microsoft's *knowledge base* (KB) articles for driver development [2] are very specific and are addressed to expert users.

In this context of problem resolution, whenever an issue occurs the user searches for answers with several sources, among which: the available documentation, call centers or technical forums and discussion lists. This way of functioning poses two major problems.

First, it requires a perfect knowledge of the used software configuration (vendor, name, version, patch, OS etc.). For example, if an interactive reporting software crashes constantly while using a specific data spreadsheet program, the user reports at best the reporting software version. He/she has no knowledge of the Java JRE version, which is actually the cause of the malfunction. As there is no complete description of the installed software, it will take several exchanges, e.g., with the support line, to determine that there is a third element at the root of the crash.

Second, each time an issue occurs, there is a repetitive and confusing process of software description. For instance, help desks employ three levels of competencies [3]. At each level, you are asked for your software configuration. If the problem isn't solved, each time you are in contact with a person from the help desk, you have to re-specify the software and the problem. This translates to frustrating repetitive operations and increased times for problem resolution. Moreover, it relates to the first problem as a non technical user is asked for detailed technical pieces of information.

With the expansion of the Semantic Web, ontologies have become a standard to model complex IS. Proposition of ontology usage for software models [4] or for semantic help desks [5] have proven that this may be a valid path to explore. Building and managing an ontology is not a trivial task, and is based on the collaboration of a specific user community. The problem is that this implies an expertise in working with ontologies and KBs, thus being reserved to a 'closed' category of users.

In consideration with the problems mentioned above, our objective is to build a software ontology, which should provide a reference point for system software description. For the initial ontology, we have chosen a restrained software perimeter, related to our expertise: *decision support systems* (DSS). To this end, we propose a semantic collaborative online platform, *TimSys*, which enables the description of user software environments starting from the ontology software concepts. The main idea behind *TimSys* is to help the evolution of the software ontology by integration of non technical user feedback. This way, everyone contributes to the development of the ontology, even if they are not KB experts.

The remainder of the paper is organized as follows. Section 2 presents the main software types with DSS and the problems of software configuration description. Section 3 shows how ontologies are used for knowledge modeling, and some of the advantages and drawbacks of using them. Section 4 introduces the *TimSys* platform, with the data

 ISBN: 978-1-61208-130-4

model and the use case scenarios. Finally, Section 5 sums up the conclusion and the future directions for this work.

II. SOFTWARE CONFIGURATION DESCRIPTION

First, this section introduces the main DSS software types, and then the existent problems and solutions with software description.

A. Decision Support Systems Software

DSSs represent the use case of our proposition. They are a type of IS that supports business and organizational decision-making activities. They have been thoroughly described by Inmon [6], with focus on data warehouse architectures. Software environments of DSSs include the following four major components:

(i) A data provider which contains the data that is integrated with the data warehouses. This data can be structured (e.g., DBs) or non-structured (e.g., technical documentation). The most often met solution is relational DBs (e.g., SQL Server, Oracle DB).

(ii) ETL (Extract, Transform, Load) software is in charge of transforming the provided data and loading it into the data warehouses. ETLs are usually developed by the data provider software editors (e.g., Oracle DW Builder, Data Integrator & Data Services by SAP).

(iii) The *data warehouse*, which stores the aggregated analytical data. Examples include the Oracle Hyperion Essbase or SAP Business Objects.

(iv) The *use interfaces* that provide access to the data from the data warehouse, usually for reporting (e.g., Hyperion Interactive Reporting , Microsoft Excel).

As decisional experts, we have been faced with the need of describing the software products above. Usually, the enterprises maintain this information in plain text documents, or eventually semi-structured ones (e.g., office documents with templates). This implies that every reference to the software configuration is based on a specific document, which must be provided each time. Moreover, version control has to be investigated for the documentation and for the software configuration. We have met several situations where software patches were applied without proper documentation (e.g. undocumented software migration). If the initial configuration specification is not updated, inconsistencies and false information occur.

B. Software Configuration Description

Software description offers many modeling alternatives. With the development of modeling tools such as UML or *Architecture Description Languages* (ADL), companies understood that integration and easy access are key factors for fast problem resolution.

In [7], the authors present an overview of the usage of UML with software architecture description. There is an extensive area of research over this subject, at a very detailed and technical level. Although they provide standardization with the description language, the complexity of such solutions is in most cases a ‘deal-breaker’ when facing simpler needs.

Another solution, less complex and simpler to use is system information software (e.g., Belarc Advisor [8]). For example, on Windows machines, the SOFTWARE registry keys contain reference to the installed software. Nevertheless, this solution has several drawbacks. First, it requires the installation of a specific program on each machine. Second, there is no complete view of the system (i.e., number of physical machines, how they are connected). Third, there is a problem with information availability, as the software list is not managed collaboratively; its sharing requires each time a duplication of the description file.

Our proposition is elaborated over the two modeling aspects presented above, taking the benefits of both. First, it uses a model complex enough, which enables the description of machines, software and the links that exist between them, but not too complex to enter the ADL world, while providing an intuitive interface for non-technical users. Second, by using ontologies, it overcomes the issues of availability and synchronization. Each software, configuration and system has its own unique URI, while assuring a complete system overview. Moreover, as the data model is opened, users benefit by adding feedback and continually improving it.

III. LINKED DATA AND ONTOLOGIES

With the development of the web and the expansion of the Internet, linked data is specified as the future of information throughout online environments. Developed by Berners-Lee, linked data is founded over the collaborative efforts of the Web 2.0 and the semantics of the future Web 3.0 [9]. The proposition states that the entire information on the web is part of a single global KB.

The formalization of the linked data concept is made through ontologies. Introduced by Gruber [10], an ontology defines a set of representational primitives able to model a domain knowledge or discourse [11]. An ontology allows the definition of three types of concepts: (i) classes (type of concept), (ii) individuals (instances of classes), and (iii) properties (links between classes and/or individuals). A sentence in an ontology is represented under the form of a triplet (subject, predicate, object), e.g., (Windows2003SRV, isA, Windows2003). Ontology expression languages are XML based, such as the W3C standards RDFS and OWL [12]. Additionally, SPARQL enhances SQL-like data query to retrieve information from ontologies.

Relating to the problem of software configuration, in [13], the authors provide an overview over how ontologies mix with UML. Moreover, some of our previous works with ontology models for managing DSSs [14] have shown the advantages and inconveniences of ontology modeling.

The benefits of using ontologies come from the dynamics of the data model, high expressivity and inference support. Dynamics refers to the fact that the information model is prone to constant changes (unlike DBs implementations), as collaboration is the key to building an ontology. High expressivity indicates that any matter or facts can be expressed within the ontology (from where the three levels of expressivity with the OWL). Last, inference allows the deduction of new knowledge from the existing knowledge by using axioms and rules.

 ISBN: 978-1-61208-130-4

On the contrary, the main drawbacks of ontologies are the novelty of the technology, information retrieval performance and high technical competences requirement. Only recently the industry has shown its interest towards this technology (i.e., Oracle 10g semantic module). Data retrieval performance for large scale ontologies proves to be a problematic point, from where the recommendation that for high number of concepts, relational DBs are preferred for faster access [15]. The last major inconvenient is that working with ontologies (as with any new technology) requires a formal technical preparation. As ontologies aim to sustain a large collaborative online usage, this sends aback a good part of its 'target' users.

Our proposition is implemented with regards to these disadvantages. We use a combined data model (DB + OWL), to assure high expressivity and collaboration, while providing fast data access.

IV. TIMSYS

TimSys [16] is our proposition for a collaborative, semantic, online and non-technical software configuration description platform. Collaborative expresses the fact that its users play an active role in the evolution of the software KB. Semantic indicates the usage of ontologies for KB formalization. Online sends to the standardization of the software concepts. Last, non-technical underlines that anyone can contribute to the development of the KB, regardless his background in working with ontologies.

TimSys is composed of two main modules: (i) the software and systems KB and (ii) the user interface .

A. The Knowledge Base

The KB contains the totality of available software, configurations and systems. As mentioned earlier, the data model is a combination between relational DBs and OWL ontologies.

OWL is used to implement the software ontology with a very high granularity. All the details from editor to patch version are described. Additionally, it allows the dynamic description of relations between software such as compatibility or functionalities. Inference rules play a very important role, as to establish software version dependencies. Our prototype software ontology contains 902 individuals, 25 classes, 13 object properties and 14 data type properties with the OWL DL expressivity.

An example of a software concept from the ontology (Windows Server 2003 R2) is shown in the following table

Table 1 – Software ontology concept example (triple)

Subject	***Predicate***	***Object***
Win2k3SRV_R2	rdf:type	Win2k3SRV
	hasMajorVersion	"5"^^xsd:string
	hasVersion	"5.2"^^xsd:string
	isCompatibleWith	Essbase_9.2.1
	hasPrevious	Win2k3SRV_SP1
	hasEditor	Microsoft

First, we notice the inclusion of the *Win2k3SRV_R2* individual the general *Windows2k3SRV* class via the *rdf:type* property. Then, two string data properties indicate the version of the software. The object property *isCompatibleWith* specifies the list of software with which it is compatible. The *hasPrevious* property links this specific version with the Win2k3 server products timeline. Last, the *hasEditor* property links it with the *Microsoft* concept, which is obtained by querying the DBPedia SPARQL endpoint [17] as an already existing concept.

Relational DBs are used for the representation of the systems and the corresponding software configurations. We define a *configuration* as the totality of (interesting) software installed on a single machine (the OS and the installed software). A *system* is defined as a combination of one or several configurations; which we consequently call a *timsys*.

The choice of DBs for the implementation comes from the fact that, unlike the software ontology (where there is a reduced number of concepts), the number of configurations and systems may reach billions. In order for the DB model to be as fast as possible, a mapping of the software ontology concepts is done such that they are also formalized in the DB model. This means that the data backend is fully assured by relational DBs. Any evolution in the ontology model results in its remapping into the DB model. This permits a constant availability of the software lists, while new changes to the software ontology are processed.

B. Evoloving the Ontology - The User Interface

The graphical user interface is the front end of the TimSys platform. It includes two major different sub interfaces: (i) access and (ii) management.

The access interface enables the online view of a system by an unique hash link (e.g., http://www.timsys.org/hjJKuyJ8). Using this link, anyone at anytime can have a look at the configurations and list of software. This greatly helps the problematic of ambiguity, duplication and synchronization while offering access to an opened software KB.

The management administration interface proposes the creation of new systems or the modification of existent ones, similarly via an unique link. The interface is build with regards to non-technical users, thus remaining as simple as possible. Alongside permitting users to describe their configurations, it provides the possibility of feedback in the cases where a required software is not found in the KB.

This is the *key aspect* towards the evolution of the software ontology. Whenever a software is not found, the user fills a three field form with the software editor, name and version (only the last 2 are compulsory). This feedback is stored temporarily in the DB, becoming instantly available to the user and with the destination of ontology integration. At this point, the KB experts are in charge of updating the software ontology accordingly. Once the ontology updated (e.g., once per day), the remapping of the new concepts to the DBs is made, and the new software becomes permanently available. We note that the intervention of a KB expert is always needed for integration.

Summing up, Figure 1 shows the general functioning of the *TimSys* platform, with the presented modules and use cases.

 ISBN: 978-1-61208-130-4

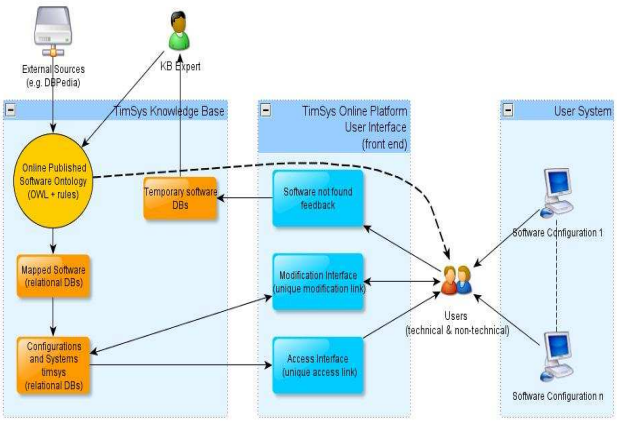

Figure 1 – *TimSys* Overall Architecture

The arrows indicate the direction in which the data flows. We notice that this is both ways for the management modification interface. Moreover, the ontology is published online and available to users for direct access of its concepts.

V. CONCLUSIONS

In this paper, we presented a proposition for building an ontology by involvement of non KB experts. Specifically, the objective was to develop a software ontology by integrating a maximum of user feedback. To this end, we proposed an online platform for describing software configurations. Consequently, we proposed solutions to overcome the issues of collaboration, synchronization and availability when it comes to describing the software environments, with the use case of DSSs.

The state of the art offered a view over software configuration modeling approaches and over the semantic web technologies. With our proposition, *TimSys*, we have presented a combined DB/OWL data model and an intuitive interface for access and management. Thus, we have seen how non-technical users feedback contributes to the development of the software ontology.

Nevertheless, the work presented here is at an early stage. Our future works will detail the aspects of: DB/OWL mapping with the data model, usage of existing software KB (more than DPBedia), validation of the ontology and its publication as a recognized reference. As it is an open source project, we aim at building an active community around *TimSys*, for both technical and non-technical feedback.

REFERENCES

[1] P. Oreizy, “Decentralized software evolution,” in The International Conference on the Principles of Software Evolution (IWPSE 1), 1998. Last accessed November 2010. Available: http://www.ics.uci.edu/~peymano/-papers/iwpse98/

[2] Microsoft. Knowledge base articles for driver development. Last accessed November 2010. Available: http://www.microsoft.com/whdc/driver/-kernel/kb-drv.mspx

[3] T. N. I. A. System. Help desk level competencies. Last accessed November 2010. Available: http://-www.nitas.us/docs/-Help%20Desk%20Level%20Competencies.pdf

[4] M. Brauer and H. Lochmann, “An ontology for software models and its practical implications for semantic web reasoning,” in The 5th European semantic web conference on The semantic web: research and applications, ESWC’08, 2008.

[5] Nepomuk. Mandriva community case study first prototype of a social semantic help desk. Last accessed November 2010. Available: http://-nepomuk.semanticdesktop.org/xwiki/bin/download/-Main1/D11-2/-D11.2_v10_NEPOMUK_1st%20Prototype%20of%20Social%20Semantic%20Helpdesk.pdf

[6] W. H. Inmon, Building the data warehouse, fourth edition, W. Publishing, Ed. Wiley Publishing, 2005.

[7] P. Avgeriou, N. Guelfi, and N. Medvidovic, “Software architecture description and uml,” UML MODELING LANGUAGES AND APPLICATIONS, vol. 3297, pp. 23–32, 2005.

[8] Belarc. The belarc advisor. Last accessed January 2011. Available: http://www.belarc.com/free_download.html

[9] C. Bizer, T. Heath, and T. Berners-Lee, “Linked data - the story so far,” International Journal on Semantic Web and Information Systems (IJSWIS), vol. 5, no. 3, pp. 1–22, 2009.

[10] T. Gruber, What is an ontology? Academic Press Pub., 1992.

[11] L. Liu and M. T. Özsu, Encyclopedia of Database Systems, L. Liu and M. T. Özsu, Eds. Springer-Verlag, 2008. Available: http://tomgruber.org/writing/ontology-definition-2007.htm

[12] W3C. World Wide Web consortium. W3C. Last accessed July 2010. Available: http://www.w3.org/

[13] J. Savolainen, “The role of ontology in software architecture,” in OOPSLA Workshop on How to Use Ontologies and Modularization to Explicitly Describe the Concept Model of a Software Systems Architecture, 2003.

[14] V. Nicolicin-Georgescu, V. Benatier, R. Lehn, and H. Briand, “Ontology-based autonomic computing for decision support systems management,” in The First International Conference on Models and Ontology-based Design of Protocols, Architectures and Services, MOPAS 2010, pp. 233–236, 2010

[15] N. K. Irina Astrova and A. Kalja, “Storing owl ontologies in sql relational database,” International Journal of Electrical, Computer, and Systems Engineering, vol. 1, no. 4, pp. 242–247, 2007.

[16] This is my System (TimSys). Last accessed January 2011. Available: www.timsys.org

[17] DBpedia. DBpedia SPARQL endpoint. Last accessed January 2011. Available: http://dbpedia.org/sparql

 ISBN: 978-1-61208-130-4