POEM: an Ontology Manager based on Existence Constraints*

Nadira Lammari[1], Cédric du Mouza[1] and Elisabeth Métais[1]

[1] Lab. Cedric, CNAM
Paris, France
{lammari, dumouza, metais}@cnam.fr

Abstract. Whereas building and enriching ontologies are complex and tedious tasks, only few tools propose a complete solution to assist users. This paper presents POEM (Platform for Ontology crEation and Management) which provides a complete environment for ontology building, merge and evolution. POEM's algorithms are based on existence constraints and Boolean equations that allow to capture the semantic of concepts. POEM also integrates a semantic fusion that relies on WordNet and supports different storage formats. The different functionalities are implemented as a set of web services.

1 Introduction

Sharing data from multiple heterogeneous sources has been for several years a challenging issue in the area of Information Systems. From federated multi-sources Data Bases to the Semantic Web challenge, modern applications need to intensively exchange information. There is now evidence that ontologies are the best solution for dealing with semantically heterogeneous data. By adding sense to data sources, ontologies allow to "understand" their contents - and then to provide pertinent information to the users.

Nowadays, all experts recognize ontology's building and management as one of the most critical problem in their large scale utilization. Many tools have been proposed but they mainly lack in automatically taking into account the semantic aspects in building, merging and validating ontologies. Usually the semantic part of the work is done manually without guideline. Nowadays, many -small or large- ontologies have been developed around the world; building a new one has to take into account these legacy systems.

The POEM tool provides a complete environment for ontology building, merging, evolution, edition, transformation and storage. Due to space limitation, only two aspects will be particularly developed in this paper: the concepts of *existence constraints* and Boolean equations that help to capture the semantic of concepts, and the construction and reuse (*i.e.*, merging) algorithms.

* Work partially funded by Semweb project,*http://bat710.univ-lyon1.fr/∼semweb/*

Related work

Many research work had been dedicated for building and maintaining ontologies: (a) ontology learning methodologies from dictionary, from text, from XML documents, or from knowledge base [6, 10, 17], (b) ontology merging methods of existing ontologies [14, 13], (c) ontology re-engineering methods [8], and finally (d) ontology construction from scratch [7, 6, 9].

Moreover, several approaches were recently proposed to perform the merging. For instance [11] presents a merging based on WordNet and the Jaccard measure. However this technique does not take into account the attributes of the different concepts leading to the merging of non-similar concepts or redundancy in the final ontology. [16] introduces another approach called FCA-Merge that uses natural language techniques and formal concept analysis techniques. In [15] the authors propose an automatic merging algorithm named OM algorithm, based on the confusion theory for detecting concept matching. This algorithm is however syntactic and does not merge concepts that are semantically equivalent.

Besides methodologies for building ontologies many tools are now available like Protégé [4], Corporum [2], Text-To-Onto [5], Chimaera [1], OntoLingua [3], etc. All these tools have an editing functionality. Some of them also present merge and/or validation and/or reasoning functionalities. They however lack mechanisms and policies implementing methods rules and guidelines.

The rest of the paper is structured as follows: Section 2 introduces the existence constraints and associated algorithms, Section 3 details the ontology merging technique, Section 4 emphases implementation and Section 5 concludes.

2 Existence constraints and associated techniques

This section presents the concept of existence constraints and two of the basic algorithms (for normalization and extraction) that will be compound to build high level functionalities. These algorithms are detailed in [12].

2.1 Definitions of existence constraints

We distinguish three kinds of existence constraints: mutual, exclusive and conditioned existence constraints. A mutual existence constraint defined between two properties x and y of a concept c, denoted $x \leftrightarrow y$, describes the fact that any instance associated to c has simultaneously the properties x and y. An exclusive existence constraint defined between two properties x and y of a concept c, denoted $x \not\leftrightarrow y$, means that any instance associated to c that has a property x can not have y as another property and conversely. Finally, a conditioned existence constraint defined between two properties x and y of a concept c, denoted $x \mapsto y$, captures the fact that any instance of c described by the property x must also be described by the property y. The inverse in not always true. For example, let `vehicle` be a concept representing vehicles such as planes and boats, and

described by a set of properties: `move`, `transport`, `fly` and `float`. The fact that any vehicle moves and transports is translated by `transport`↔`moves`. Also, every vehicle that moves doesn't systematically fly. However, every vehicle that flies must move. These two assertions can be formalized using the conditioned existence constraint: `fly`↦`move`. We suppose in our example that only planes fly and there is no plane that floats. This hypothesis is translated into the exclusive existence constraint: `float`↛`fly`.

2.2 Normalization technique

The normalization technique enables to build ontologies from a set of properties and a set of existence constraints. It consists in automatically deducing valid property subsets that are compatible with the existence constraints. Let S be a subset of the property set of a given ontology O. S is said to be a valid subset of properties of O if and only if S satisfies the following three rules:

- **Homogeneity rule:** Any property of O linked to a property of S by a mutual existence constraint is in S.
- **Non-partition rule:** There is no exclusive existence constraint between properties of S.
- **Convexity rule:** Any property of O required by a group of properties of S is in S (taking into account the conditioned existence constraints).

Intuitively, the sets of coexisting properties are first deduced, by taking into account the mutual existence constraints. Then, by means of exclusive existence constraints, the sets of properties that may coexist are derived. Finally, sets that are compatible with the conditioned existence constraints are selected among the sets of properties that may coexist. These sets describe concepts represented by the ontology. These concepts are then organized into an Is_A inheritance graph.

2.3 Translation techniques

We present here two translation techniques described in [12] for the building of ontologies from scratch and that we implanted in POEM. The first one called `O_TO_BF`, transforms an ontology into a Boolean function while the second called `BF_TO_O` is the reverse operation. `O_TO_BF` consists in building a disjunctive Boolean function $\phi(x_1, \ldots, x_n)$ where each x_i corresponds to a property of the ontology and each maxterm T represents a type of instance represented by each concept C of the ontology. This maxterm is a conjunction of x'_i variables where each x'_i is either equal to x_i if the property associated to x_i describes T or equal to $\overline{x}_i$ otherwise.

`BF_TO_O` derives from a Boolean function the set of existence constraints and then using the normalization technique deduces the ontology. For the generation of the existence constraints from a Boolean function $\phi(x_1, \ldots, x_n)$ where each x_i corresponds to a property a_i, the technique transforms ϕ from a disjunctive form to a conjunctive one and then, by analyzing the different minterms of the transformed expression, it derives the existence constraints by applying the following rules to the conjunctive function:

- **Rule 1**: A minterm of ϕ of type x_i is translated into the non-null constraint
- **Rule 2**: A minterm of type $(\overline{x}_i + \ldots + \overline{x}_j + x_k)$ describes a conditioned existence constraint "$a_i, \ldots, a_j \mapsto a_k$".
- **Rule 3**: A minterm of type $(\overline{x}_i + \overline{x}_j)$ is translated into the exclusive existence constraint "$a_i \not\mapsto a_j$".
- **Rule 4**: Two attributes a_i and a_j coexist in R ($a_i \leftrightarrow a_j$) iff the two conditioned existence constraints $a_i \mapsto a_j$ and $a_j \mapsto a_i$ are derived from ϕ.

3 Ontology merging

POEM allows to build an ontology either from scratch or by reuse. For lack of space, we will focus on our building mechanism based on merging. The different ontology construction processes from scratch may be found in [12]. The merging process can basically be decomposed into two steps: the matching step and the integration step.

3.1 The matching step

The matching technique in POEM is realized for two normalized ontologies. It encompasses two phases, property merging and concept merging. Let O_1 and O_2 be these two ontologies. To achieve the matching between them, we first retrieve, for each property p from O_1 and O_2, its synset S_p from the general ontology WordNet and then we compute for any couple of properties (p_i, p_j), such that $p_i \in O_1$ and $p_j \in O_2$, the similarity degree (denoted $SimDegree(p_i, p_j)$) between them according to the following formula:

$$SimDegree(p_i, p_j) = \frac{|S_{p_i} \cap S_{p_j}|}{|S_{p_i} \cup S_{p_j}|}$$

According to this formula, the more numerous the synonyms of the two properties are, the higher the similarity degree is. Once the similarity degrees between properties measured, we perform the property matching. In POEM, we merge p_i and p_j iff they share at least one synonym, and there is no property in O_1 (resp. O_2) semantically closer to p_j (resp. p_i). More formally two properties p_i and p_j are matched if and only if the three following conditions are satisfied:

(i) $SimDegree(p_i, p_j) > 0$,
(ii) $\nexists p_k \in P_2,\ SimDegree(p_i, p_k) > SimDegree(p_i, p_j)$,
(iii) $\nexists p_k \in P_1,\ SimDegree(p_k, p_j) > SimDegree(p_i, p_j)$.

where P_1 (resp. P_2) denotes the set of properties from O_1 (resp. O_2).

When merging two properties, we choose as common name for them the most specific synonym syn_{ij} they shared, *i.e.* the first one that occurs in $(S_{p_i} \cap S_{p_j})$. Once a property merging is performed, we replace p_i and p_j by syn_{ij} in their respective ontology.

For the merging of concepts, our approach distinguishes itself from other existing techniques because it considers all the following cases for the similarity between concepts:

- **case 1**: a similarity both between concept names and between names of their respective properties,
- **case 2**: a similarity only between concept names,
- **case 3**: a similarity only between property names of the concepts.

More precisely, let c_i and c_j be two concepts from an ontology O and π_{c_i} (resp. π_{c_j}) be the property set of c_i (resp. c_j). To estimate the similarity between c_i and c_j, we refer to Jaccard measure defined by the ratio between the number of shared properties of these two concepts and the union of their properties, *i.e.*:

$$jaccard(c_i, c_j) = \frac{|\pi_{c_i} \cap \pi_{c_j}|}{|\pi_{c_i} \cup \pi_{c_j}|}$$

We assume the existence of a threshold value τ for the Jaccard measure to assert the similarity or not between two concepts. We also define a decreasing factor γ to attribute a lower similarity value when both concept names and property names are not similar (what is considered as the highest similarity). Our algorithm for estimating the similarity between two concepts c_i and c_j is:

---

if c_i and c_j share at least one synonym (their synsets are not disjoint) **then**
  if $jaccard(c_i, c_j) \geq \tau$ **then**
    c_i and c_j are similar with a similarity degree equal to $jaccard(c_i, c_j)$ **(case 1)**
  else
    if c_i and c_j have at least one parent $Parent(c_i)$ and $Parent(c_j)$ such that these two parents share at least one synonym (their synsets are not disjoint) and $(jaccard(Parent(c_i), Parent(c_j)) \geq \tau)$ **then**
    they are similar with a (lower) similarity degree equal to $\gamma \times jaccard(c_i, c_j)$ **(case 2)**
    else unable to evaluate the similarity; return the concepts to the ontology engineer
    endif
  endif
else
  if $jaccard(c_i, c_j) \geq \tau$ **then**
    they are similar with a (lower) similarity degree equal to $\gamma \times jaccard(c_i, c_j)$ **(case 3)**
  else the two concepts are not similar
  endif
end

After estimating all the similarity degrees between concepts, we merge two concepts c_i and c_j if they have a positive similarity degree, and if a higher value can not be found by replacing one of the two concepts by another one from the same ontology. The property set of the resulting concept c_{res} is the union of c_i and c_j property sets. To chose a name for c_{res}, we adopt the following strategy:

- in case 1, we choose for c_{res} the name of the concept that participates more in its building, that is to say the concept which has more properties in c_{res},
- in case 2, we choose the most specific name, *i.e.* the name that appears first in the common synset of the two concepts,

– in case 3: we compute the confusion value (see [15] for definition) between concept names according to their respective position in WordNet and we choose the name of the concept with the lowest value.

Finally, the matching technique allows us to give the same names to similar concepts and properties. The two ontologies O_1 and O_2 are transformed according to the result of the matching into respectively O'_1 and O'_2.

3.2 The integration step

The final step of our process consists in building by a merge the ontology O_3 from the two ontologies O'_1 and O'_2 obtained after matching. For that purpose, we first translate the two ontologies into two Boolean functions ϕ_1 and ϕ_2 using the `O_TO_BF` mapping technique for both ontologies, then we gather the two resulting Boolean functions into one Boolean function ϕ_3 and finally, by applying the `BF_TO_O` mapping technique, we deduce the resulting ontology.

To gather the Boolean functions ϕ_1 and ϕ_2 into a Boolean function ϕ_3 we proceed to the following step-by-step algorithm:

---

let B the set of variables of ϕ_1 and ϕ_2
for each minterm T of ϕ_1 and ϕ_2 **do**
  for each variable x in B **do**
    if neither x nor $\overline{x}$ appears in T **then** $T = T.\overline{x}$ **endif**
  endfor
endfor
if T does not appear in ϕ_3 **then** $\phi_3 = \phi_3 + T$ **endif**

---

4 Implementation

We design POEM in accordance with four principles: (*i*) modularity (POEM is built as a set of seven modules, one for each basic functionality), (*ii*) reusability (a web service architecture), (*iii*) extensibility and (*iv*) guidance (three levels of guidance are proposed to give a cognitive support to human, expert or not).

The global architecture is basically a three-tier architecture with POEM, the Sesame system and a database. The services in POEM are implemented in Java JDK1.5 using MyEclipse 5.0 and the web service library XFire. Sesame acts as a middleware for gathering and storing ontologies and metadata. It is deployed with the Tomcat web server.

Figure 1 shows the interface of the POEM tool when merging two ontologies. The upper part of the interface is dedicated to the selection of ontologies and export parameters. You can for instance with this interface select a ntriples ontology and export it in RDF. You can also load it and proceed to concept extraction that are displayed in the window below. For instance here we have for the first ontology classes `Airplane`, `Boat` and `Vehicle` along with their properties. Finally POEM displays the ontologies loaded at bottom of the window. In

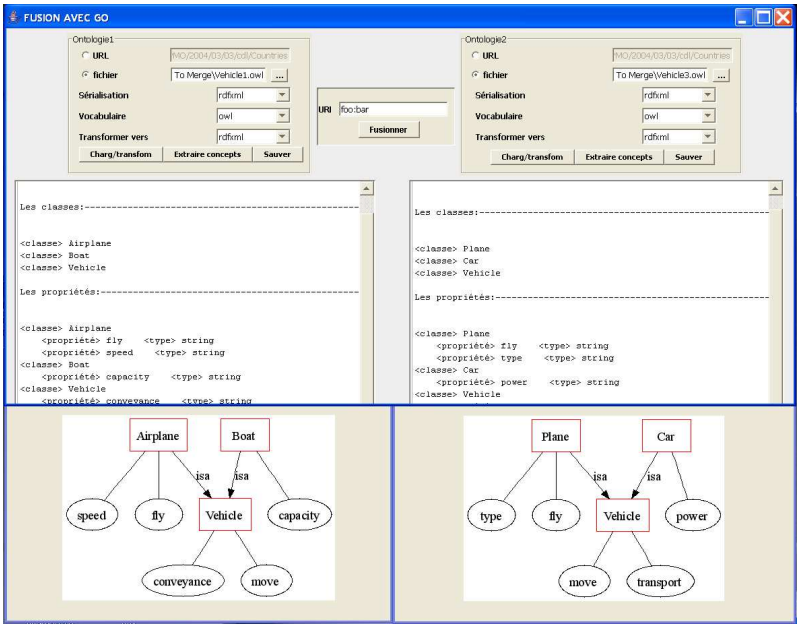
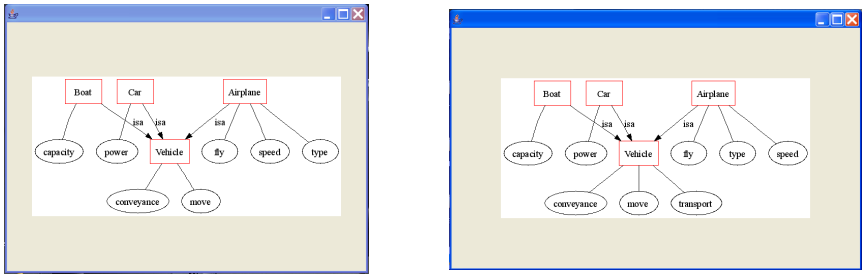


Fig. 1. Merging two ontologies with POEM

Figure 2(a) we see the result got by proceeding to the fusion with POEM. We notice that POEM, based on semantical similarity of the concept names and of their properties, decided to merge the concepts `Airplane` and `Plane` and the resulting concept `Airplane` beholds the union of the properties that are semantically different. The concept `Vehicle` has only two properties, since the former `conveyance` and `transport` properties were identified as similar. Figure 2(b) represents the merge with Protégé 3.3.1 (one of the most frequently used ontology manager tool). If the concepts were, in this case, merge correctly, we see that the merge did not apply to properties. Similarly, our experiments show that concepts whose names are not synonyms in WordNet but have numerous similar properties are also not merge in Protégé.

5 Conclusion

In this paper we present the POEM existence constraints based platform to assist ontology creation and maintenance with a guidance module. POEM takes into account the semantic, via semantic constraints for the building tool and via WordNet access for ontology merging. In this paper we focused on the building and the merging algorithms, although other algorithms presented in [12] are also implemented. For the next step we plan to integrate the detection of similarities between properties in the ontology construction processes. Another perspective is to extract properties and constraints from textual sources.



(a) with POEM (b) with PROTÉGÉ+PROMPT

Fig. 2. Result of the merging

References

1. CHIMAERA. http://www ksl.stanford.edu/software/chimaera/.
2. CORPORUM-ONTOEXTRACT. http://www.Ontoknowledge.org.
3. ONTOLINGUA. http://ontolingua.stanford.edu/.
4. PROTEGE. http://protege.stanford.edu/.
5. TEXT-TO-ONTO. http://webster.cs.uga.edu/ mulye/SemEnt/.
6. E. Agirre, O. Ansa, E. H. Hovy, and D. Martínez. Enriching Very Large Ontologies Using the WWW. In *Proc. Intl. ECAI Workshop on Ontology Learning (OL)*, 2000.
7. J. Davies, D. Fensel, and F. van Harmelen. *Towards the Semantic Web: Ontology-driven Knowledge Management*. Wiley, January 2003.
8. A. Gómez-Pérez and D. Rojas-Amaya. Ontological Reengineering for Reuse. In *Eur. Workshop on Knowledge Acquisition, Modeling and Management (EKAW)*, pages 139–156, 1999.
9. M. Grüninger and M. S. Fox. Methodology for the Design and Evaluation of Ontologies. In *Proc. Intl. IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*, pages 1–10, 1995.
10. L. Khan and F. Luo. Ontology Construction for Information Selection. In *Proc. Intl. Conf. on Tools with Artificial Intelligence (ICTAI)*, pages 122–127, 2002.
11. H. Kong, M. Hwang, and P. Kim. Efficient Merging for Heterogeneous Domain Ontologies Based on WordNet. *Jour. of Advanced Computational Intelligence and Intelligent Informatics (JACIII)*, 10(5):733–737, 2006.
12. N. Lammari and E. Métais. Building and Maintaining Ontologies: a Set of Algorithms. *Data Knowl. Eng. (DKE)*, 48(2):155–176, 2004.
13. D. McGuinness, R. Fikes, J. Rice, and S. Wilder. An Environment for Merging and Testing Large Ontologies. In *Proc. Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, pages 483–493, 2000.
14. N. Noy and M. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proc. Nat. Conf. on Artificial Intelligence and on Innovative Applications of Artificial Intelligence*, pages 450–455, 2000.
15. A. Rasgado and A. Guzman. A Language and Algorithm for Automatic Merging of Ontologies. In *Proc. Intl. Conf. on Computing (CIC)*, pages 180–185, 2006.
16. G. Stumme and A. Maedche. FCA-MERGE: Bottom-Up Merging of Ontologies. In *Proc. Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 225–234, 2001.
17. H. Suryanto and P. Compton. Discovery of Ontologies from Knowledge Bases. In *Proc. Intl. Conf. on Knowledge Capture (K-CAP)*, pages 171–178, 2001.