Lessons Learned From Ontology Design

Jean-André Benvenuti[1] Laure Berti–Équille[2]
Éric Jacopin[1]

[1]Macclia, Écoles de Saint-Cyr Coëtquidan, France
{jean.benvenuti,eric.jacopin}@st-cyr.terre.defense.gouv.fr

[2]IRISA, Université de Rennes 1, France
berti@irisa.fr

Introduction Since 2001, we have been involved in the development of a system named SABRE [1, 2, 3, 4] for supporting the training of military students of the French Army; we expressed the knowledge of the environment and concepts of military work as ontologies using Protégé. We consider that the ontology contains explicit descriptions of the concepts (represented by words: e.g. "to serve one's homeland") used in our military domain. The knowledge is also at stake in the military training and then must be understandable by the students. Finally, not only our ontology must produce usable data structures, but it also must be accepted by the French Army instructors. The SABRE system must consequently possess the necessary data structures so that this knowledge can be expressed, edited, completed, presented, used in a training session to check the current knowledge of the trained students, and stored for future references. However, building ontologies and designing the data structures for such a knowledge-based system are not easy tasks; we recently found that some set-theory-based characteristics extracted from our "in progress" ontology could considerably help the ontology builder. This paper describes our experience on building the ontology of the French Military training using Protégé and how this step-by-step process can be supported and guided on-the-fly by systematically checking the theoretical properties of the instance sets of the ontology.

During the design of an ontology for military training [1, 2], we partially solved the ambiguity (getting different interpretations for identical values of different slots), completeness (adding interpretations for all values) and minimality (avoiding intractability) problems with extra design and rules [4]. We here present several (polynomial-time) functions which can automatically detect where *i)* to add relevant interpretation rules or *ii)* to complete our ontology (adding classes and slots) for assisting the ontology design.

Illustrative Example This paragraph illustrates the problem we encoutered in the case of ontology design in the domain of military training with the help of a geometrical problem (chosen for the sake of simplicity). Then we present a set of routines which aims at pointing to the classes, slots and instances which are either incomplete or need rules for further interpretation.

On the left of Figure 1 is an UML class diagram (a) where Polygon and Point are two classes; at least 3 ordered Points can be the vertices of 1 Polygon (adapted from [8, page 68]). The object diagram (b) is correct with respect to the UML class diagram (a); `Triangle1` and `Square` are instances of Polygon and `Triangle2` is an instance of Polygon whose vertices are made from vertices of `Triangle1` and `Square`. `Triangle1` has vertices {`P1`,`P2`,`P3`}, `Square` has vertices {`P4`,`P5`,`P6`,`P7`} and `Triangle2` has vertices {`P1`,`P5`,`P7`}. One interesting question is : May {`P1`,`P5`,`P7`}, instances of Point and vertices of `Triangle2`, be also vertices of both `Triangle1` and `Square`? The correctness of Figure 1(b) with respect to the UML class diagram (a) entails a positive answer to this question[1].

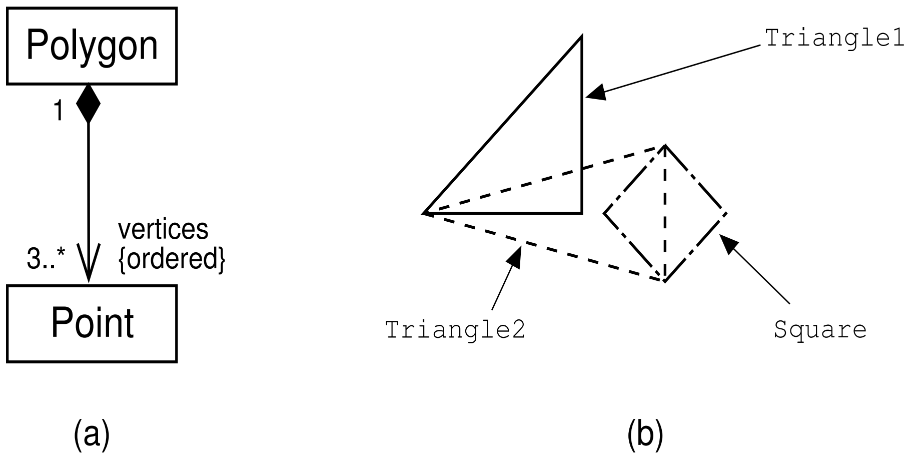

Figure 1: A Geometrical Example Illustrating the Need for Disambiguization

Now consider the following set of coordinates of instances of `Point` that can be associated to Figure 1(b): $\{(0, 0), (0, 2), (2, 2), (0, 3), (0, 5)\}$; which of these coordinates are that of the three vertices of `Triangle2`? Or, in other words, which of these coordinates activate `Triangle2`? As `Triangle2` shares all its `vertices` with both `Triangle1` and `Square`, it should be obvious that the answer to this question is impossible when some information is missing (and may not be available) such as the orientation of the axes and the position of their origin: interpreted only from the coordinates, Figure 1(b) is ambiguous. Could we modify Figure 1(a) to avoid this ambiguity, and how?

As illustrated by this geometrical discussion, disambiguization and completeness are the problems we encountered when editing the knowledge of the French Military Training and designing our ontology with Protégé [1]. Our objective is to activate this knowledge so as to educate military staff to the recently designed French soldier's code [5, 6, 7]; the proposed pedagogical method [5] is to confront the learner to a concrete case, describing a real-life military situation where the learner is demanded to act as an actor of the concrete case and play his rôle throughout. In the context of the French Military Training ontology designed from several referential documents [6, 7], the word "discipline" may have different interpretations depending on the articles studied in the current pedagogical training session: for example, in the context of the 10th article of the French Soldier's Code, "discipline" should be interpreted as close as possible as "reserve" (*i.e.*, "secrecy"). But in the 2nd article, "discipline" should be interpreted as close as possible as "courage". The previous question now stands as follows: which of the learner's behaviours activate the articles and themes of the current pedagogical session? To answer this question, we first designed a set of rules to interpret the behaviours along the pedagogical session [2, 3, 4]. However, as we came to discover that some parts of our military ontology did not need rules, we came to design a set of functions to automatically detect our need for disambiguization rules (*i.e.*, for enriching our ontology). The next paragraph formally describes these functions.

Towards automatic detection of ambiguities in ontologies We begin by an analogy with linear algebra, where eigenvalues are the coordinates of the eigenvectors defining the main axes of a vector space (all vectors of a vector space can be written as a combination of these eigenvectors). We thus use the word eigen together with: *(i)* values of a slot of a class, *(ii)* a slot of a class, *(iii)* all instances of a class and finally *(iv)* a class, to denote that the values of the slot of an instance of a class can uniquely determine this instance.

[1] Note that object-oriented design answers "no" [8, page 68]: "The general rule is that, although a class may be a component of many other classes, any instance must be a component of only one owner. [...] The "no sharing" rule is the key to composition. Another assumption is that if you delete [a] polygon, it should automatically ensure that any owned Points also are deleted."

Table 1: Algorithm for Determining Eigen Values, Eigen Instances and Eigen Classes of a Given Ontology

```
let $\mathcal{V}$ be a set;
let $c$ be a class with a slot s taking as value a subset of $\mathcal{V}$ (from $\emptyset$ to $\mathcal{V}$ itself);
let $\mathcal{I}(c)$ be the set of instances of class $c$;
let $\mathcal{V}$(s,i) be the set of values of the slot s of the instance i of class $c$;

function GetEigenValues(s, i):
    let V ← $\mathcal{V}$(s,i);
    for = all j∈ $\mathcal{I}$(GetClass(i)) with j≠i do
        V ← V \ $\mathcal{V}$(s,j)
    end for all;
    return V;

function EigenSlotQ(s, i):
    return (GetEigenValues(s, i)≠ $\emptyset$);

let $\mathcal{S}(c)$ be the set of slots of class $c$;
function EigenInstanceQ(i):
    for = all $s \in \mathcal{S}$(GetClass(i)) do
        when EigenSlotQ(s,i) throw true
    end for all;
    return false;

function EigenClassQ($c$):
    return ($\bigwedge_{i \in \mathcal{I}(c)}$EigenInstanceQ(i));
```

When EigenSlotQ(s,i) returns **true**, the EigenValues of slot s uniquely determine the instance i; which means, in our case of ontology for military training, that (at least) one behaviour the learner is asked to play corresponds to only one article of the soldier's code or else only one pedagogical theme, thus avoiding the need for interpretation of the behaviour in the context of the article or the theme of the pedagogical session.

When EigenInstanceQ(i) returns **true**, the instance i has at least one EigenSlot, which, in our case of ontology for military training, means that an article has at least one behaviour as unique representative.

Finally, EigenClassQ(c) returns **true** when all the instances of class c have at least one unique representative.

Here is what these functions return in the case of Figure 1(b):

Table 2: Characterization of the Example

GetEigenValues(vertices, Triangle1)	{P2,P3}
GetEigenValues(vertices, Triangle2)	$\emptyset$
GetEigenValues(vertices, Square)	{P4,P7}
EigenSlotQ(vertices, Triangle1)	true
EigenSlotQ(vertices, Triangle2)	false
EigenSlotQ(vertices, Square)	true
EigenInstanceQ(Triangle1)	true
EigenInstanceQ(Triangle2)	false
EigenInstanceQ(Square)	true
EigenClassQ(Polygon)	false

As vertices is not an EigenSlot of Triangle2, Polygon is not an EigenClass. We either need to add rules to interpret the coordinates correctly, or add at least one slot to Polygon (e.g. barycenter) or further add classes to the ontology (e.g. subclasses of Polygon, e.g. Triangle, Square, Pentagon, . . .) to distinguish between polygons.

Conclusion The set of functions presented in this paper greatly helped to automatically detect the need to design rules to interpret ambiguous values in our ontology (in particular, for setting up the use of this ontology in a pedagogical session) and ultimately, to complete the ontology design. These functions run in polynomial time in the number of values of the same slot across all the instances of the same class.

References

[1] Jean-André Benvenuti, Laure Berti–Équille & Éric Jacopin, *Ontological Parsing of XML Documents: A Use Case in the Domain of Training Military Staff*, in: Proceedings of the 21^{st} International Conference on Distance Education (ICDE'04), Hong-Kong (February 2004), 10 pages.

[2] Jean-André Benvenuti, Laure Berti–Équille & Éric Jacopin, *From Ontology to Inference*, in: Poster Proceedings of Ingénierie des Connaissances (IC'04), Lyon (May 2004) (in french), 2 pages.

[3] Jean-André Benvenuti, Laure Berti–Équille & Éric Jacopin, *The* Sabre *Project: an Intelligent Tutoring System for Military Behaviours*, Poster at the CNRS Summer School, Autrans (Juillet 2004) (in french), 2 pages.

[4] Jean-André Benvenuti, Laure Berti–Équille & Éric Jacopin, *When Protégé and Rules become Parsing for Learning*, Workshop on Protégé with Rules, Madrid (July 2005), 3 pages.

[5] Commandement de la Formation de l'Armée de Terre, *Guide pour l'enseignement des principes de l'Exercice du Métier des Armes et du Code du Soldat*, Saint-Maixent l'École (2002), 92 pages.

[6] État-Major de l'Armée de Terre, *l'Exercice du Métier des Armes dans l'Armée de Terre : Fondements et Principes*, SIRPA Terre (1999), 41 pages.

[7] État-Major de l'Armée de Terre, *Code du Soldat et Guide du Comportement*, Directive n^{o} 004496/DEF/-EMAT/CAB du 28 juin 1999, 21 pages.

[8] Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley (3^{rd} Edition, 2004), 175 pages.