Reasoning about multi-contextual ontology evolution

Maciej Zurawski

CISA (Centre for Intelligent Systems and their Applications)
School of Informatics, Edinburgh University
Appleton Tower, Crichton Street,
Edinburgh, EH8 9LE, Scotland, UK
m.zurawski@sms.ed.ac.uk

Abstract

In this paper we develop a formalization and algorithms that can manage the evolution of several ontologies from different contexts, using automated reasoning. It is in general difficult to maintain consistency between several ontologies, but we focus on developing computationally efficient ways of achieving this. Our formalization uses both the notions of several local contexts and of a sequence of states. We believe such a system can become a component in for example a distributed knowledge management system or some other knowledge infrastructure that requires semantic autonomy, i.e. lack of centralized semantics, but presence of a type of semantic coherence. In this paper version we summarize our approach.

Background and motivation

We envision that there will be a need for different kinds of systems that can support several ontologies, their individual evolution and maintain a type of coherence between them. For example, we would like to be able to build systems that will function as organizational knowledge infrastructures. The organizations using these will probably be decentralized and consist of separate divisions that have local autonomy in their knowledge-creating processes. Here we particularly mean semantic autonomy (see the partial definition in figure 1). Such an organization should act as a unified whole, because otherwise entities from outside (e.g. customers) interacting with the organization might be disappointed that it contradicts itself. Creating an organizational knowledge infrastructure is one application area (Zurawski, 2004), but there should exist other applications as well that also requires semantic autonomy. In both cases, this is modeled using several ontologies that can evolve, but where a kind of consistency is maintained between them.

Semantic autonomy requires (among others) these properties to hold at the same time:

1. The local contexts have the freedom to propose a change in their local ontology (i.e. the ontology of the local context).
2. The system does "in some way" maintain global ontological consistency (although it may be the case that the system doesn't have a global theory).
3. The ontological language is dynamic and open-ended (i.e. not confined by a pre-defined set) and there is an oracle (knowledge source) that can answer questions about this language.

Figure 1. In this paper we focus on developing algorithms that exhibit these three properties.

We mention here again very briefly a partial definition of semantic autonomy (for a full definition and detailed discussion about all the requirements, see Zurawski 2004). Semantic autonomy requires the properties in figure 1 to hold. Considering these requirements it is natural that our system should have an explicit notion of states. In the mentioned paper we discussed these requirements and why they make sense, and have to be possessed by distributed knowledge management system (DKM). In this paper we will instead focus on how to actually develop algorithms that have satisfied the requirements listed here.
This paper version is only a short summary of our approach. We don't present the full formalization, but focus more on the motivation.

Notions

We will first explain the basic notions we are using in order to design a system that has the above-mentioned properties.

By *multi-contextual* we mean that we have defined a finite set of subsets that all have their unique identifiers,

individual ontologies and represent a certain cognitive perspective, i.e. an individual way of representing a certain domain. Because different contexts describe the same domain (but using different ontologies) it will be natural that it is possible to create mappings between concepts in the different ontologies. We use the compose-and-conquer type of context-sensitivity (Bouquet et al. 2001).
By *ontology* we currently mean a subsumption hierarchy of logical concepts, that belongs to a certain context (we don't have any global ontology).

By *evolution* we mean that every individual ontology can change. We will particularly focus on the case when a new concept is added to an ontology together with a mapping within its home ontology and a mapping to another ontology. Because ontology evolution is so important, we have defined the notion of *states* that describe in which state a certain ontology is in. Every time an ontology changes it moves from one state to the next one, and all states are ordered.

The logical representation of ontology mappings

Because of limited space we don't define the model theory here, but we however summarize its characteristics. Now we will focus on how to formalize the ontology mappings themselves. The logic we are using has two fundamental dimensions that are used simultaneously: *contexts* and *states.* The basic entities that inhabit this logical space are *concepts*. Every concept belongs to a unique context, and there are no concepts outside the contexts. Every concept has to be created in a certain state and persists either forever or until it is deleted in some state in the future. Concepts can be applied to *instances* and then they act as logical predicates applied to constants. However, in the algorithms that we mention we will only focus on concepts and mappings – no instances will explicitly be present. Two concepts belonging to the same ontology can also be used be combined in these formulas: $R_i \wedge Q_i$ and $R_i \vee Q_i$ (and these can be nested). We have to some extent been inspired by intensional logic (see L. T. F. Gamut, 1991), although we will later focus only on the proof theory and rewrite rules.

State operators and their combination

In order to understand the ontology mapping notation we have understand its three main components, and the first component is the collection of state operators. We don't provide the formal definition here. However, informally speaking, G_r means that something will be true in all future states after r, that F_r means that something will be true at least once in the future after state r and N_r that something is true in state r. We also use a state c from which a state operator is evaluated (i.e. observed).
Moreover, we have defined a way of combining state operators, so that two state operators can be combined into one. The motivation is that this is needed when we want to combine two ontology mappings into one (and all mappings contain state operators as we will see). The reason for introducing the variables r and c is that this becomes practical later for talking about when an ontology mapping was created and in which states it is valid.

Quantifiers and their combination

The second component of ontology mappings is quantifier symbols, and there just two of them: α_1 and α_2. We don't provide the formal definitions here, but α_1 approximately means that we use a universal quantifier and α_2 an existential one. The reason why we have introduced these symbols is that they will be used in the ontology mappings, and can discern the difference between saying that a certain concept is true for all instances or for at least one instance.

Boolean functions and their combination

This is the third component of ontology mappings. We use the standard Boolean functions of two variables and they are represented in 2-DNF form. Two such Boolean functions can be combined by conjunction into a new Boolean function, using standard logical operations.

The ontology mapping notation

Here we show a part of the notation that is used for describing ontology mappings. The reason why we choose this kind of formalization is it that it seems to be good when doing efficient and automated proofs about ontology mappings. Let us call every mapping between two concepts m_i, where i is its unique identifier. A mapping m_i that holds between the concept C_1 in ontology j and the concept C_2 in the ontology k can always be expressed using on of the two following forms:

$$m_i(C_{1j}, C_{2k}) = op(\alpha(f(C_{1j}, C_{2k}))) \text{ or}$$
$$m_i(C_{1j}, C_{2k}) = op(\alpha(f(C_{1j}, C_{2k}))) \wedge op'(\alpha'(f'(C_{1j}, C_{2k})))$$

where
$op \in \{N_a, F_b, G_c\}$, $a, b, c \in S$ (the set of states)
$\alpha(\lambda) \in \{\alpha_1, \alpha_2\}$ and
$f(C_{1j}, C_{2k}) \in \{\langle e_1 \wedge e_2 \rangle, \langle e_1 \wedge \neg e_2 \rangle, \langle \neg e_1 \wedge e_2 \rangle, \langle \neg e_1 \wedge \neg e_2 \rangle\}$
where $e_1 = C_{1j}(x_j)$ and $e_2 = C_{2k}(y_k)$
(the notation of the Boolean function is the set of the conjunctions that a 2-DNF form would contain)

Combining ontology mappings by using the three kinds of rewrite or combination rules

Using the three kinds of rewrite or combination rules, we can now use them in a sequence and use them for

combining any two ontology mappings into one – that is their purpose. The first transformation is the application of rewrite rules for state operators in a way that combines two state operators into one. The second transformation is the application of rewrite rules for expressions with quantifiers in a way two combines to operators into one. The third transformation is the application of combination of Boolean functions in a way that combines two such functions into one.

Examples of mappings

The language mentioned above allows creating a huge variety of mappings. We can for example imitate the five proposed mapping types by Giunchglia (see for example Bouquet 2003) and restate them in this new concise language. Both formalisms use the notion of contexts, but the difference is that our definitions utilize the notion of states as well. The state when a mapping was created is denoted by *r*. Here are some examples:

CORRESPONDENCE – COR(C_{1j}, C_{2k})

$$N_r\alpha_1\{\langle e_1 \wedge e_2\rangle,\langle \neg e_1 \wedge \neg e_2\rangle\} \wedge G_r\alpha_1\{\langle e_1 \wedge e_2\rangle,\langle \neg e_1 \wedge \neg e_2\rangle\}$$

IS (C_{1j}, C_{2k})

$$N_r\alpha_1\{\langle e_1 \wedge e_2\rangle,\langle \neg e_1 \wedge \neg e_2\rangle,\langle \neg e_1 \wedge e_2\rangle\} \wedge$$
$$G_r\alpha_1\{\langle e_1 \wedge e_2\rangle,\langle \neg e_1 \wedge \neg e_2\rangle,\langle \neg e_1 \wedge e_2\rangle\}$$

DISJOINT (C_{1j}, C_{2k})

$$N_r\alpha_1\{\langle \neg e_1 \wedge e_2\rangle,\langle \neg e_1 \wedge \neg e_2\rangle,\langle e_1 \wedge \neg e_2\rangle\} \wedge$$
$$G_r\alpha_1\{\langle \neg e_1 \wedge e_2\rangle,\langle \neg e_1 \wedge \neg e_2\rangle,\langle e_1 \wedge \neg e_2\rangle\}$$

COMPATIBLE (C_{1j}, C_{2k})

$$F_r\alpha_2\{\langle e_1 \wedge e_2\rangle\}$$

(We should stress that for example $\{\langle e_1 \wedge e_2\rangle,\langle \neg e_1 \wedge \neg e_2\rangle,\langle \neg e_1 \wedge e_2\rangle\}$ is equivalent to to $e_1 \rightarrow e_2$, i.e. it is a DNF-form)

For example, the relationship Compatible means "There is at least one future state after r where there is at least a pair of instances (one from ontology j and one from ontology k) where the concept C_{1j} is true (when applied to its instance) at the same time as the concept C_{2k} is true (when applied to its instance)".

Algorithms for verifying consistency between ontology mappings

The problem we are trying to solve can be described as the following proof task. Given a set of existing ontology mappings $m_1(C_{ax}, C_{by})$, $m_2(C_{ax}, C_{by}) \ldots m_n(C_{ax}, C_{by})$, how can we prove if it is consistent the additional mapping $m_{n+1}(C_{ax}, C_{by})$ or not? The variables *x* and *y* refer to the ontologies of the concepts and *a* and *b* are unique concept identities (note that all these variables can be different for every mapping).

To be able to address this proof task we need to have operators that let us express the following things: $m_x \wedge m_y$, $m_x \Rightarrow m_y$ and $\neg m_x$ (and formulas that this can generate). Please note that this language is different from the one that was defined in the beginning (for talking about concepts). Now the basic entity is a mapping.

Then we need two algorithms (called A and B) that will build proof trees using refutation proofs and breath-first search (for proofs), for solving the proof task mentioned. Because of limited space we don't write them down here in full detail, but the algorithms returns an answer (yes/no) each to the following questions:

Algorithm A - "Is mapping G inconsistent with the current mappings?" *Output*: yes/no

Algorithm B - "Is mapping G valid, because it can be inferred from existing mappings?" *Output*: yes/no

This means that in both cases G is the newly proposed mapping, and there is a set of existing mappings (i.e. these are the inputs to algorithms). The algorithms are used in the following way. A newly proposed ontology mapping G is given and first we run algorithm A. If it answers "yes", then we know it is inconsistent with the existing ones. If it answers "no" we run algorithm B. If that algorithm answers "yes" then we know that the newly proposed is valid because it can be inferred from existing mappings (i.e. redundant in some way), and it answers "no" then the proposed mappings is consistent with the existing ones, but can't be inferred from them. Therefore, by using these two algorithms we have covered all three possible cases.

We don't provide here a proof of correctness and completeness. However, we just want to mention that our proof search procedure for refutation proofs using a breadth-first search, and the language used are horn clauses (since we use conjunction, implication and negation). So if the procedure finds a proof, it is valid, and if there is a proof, the procedure will find the shortest one.

Applying the algorithms to ontology evolution

Once both algorithms are in place, it is actually rather straightforward to use them for ontology evolution. An ontology transformation has to be translated to "one or more ontology mappings that are proposed to be added". For example, the addition of a new concept can be seen as inventing a new concept in an ontology and adding an internal mapping (within its home ontology) and a mapping to an external ontology. Then we run algorithms

A and B for both these proposed mappings, and only if there is no created inconsistency detected in neither of the cases, the evolutionary step is accepted and the ontology changes to a new state. Otherwise, the evolutionary step would be forbidden, and the ontology would remain unchanged.

Related research

Background to multi-context logic is give by (Giunchiglia 1993) and multiple languages and bridge rules are discussed. A description and motivation of cognitive context is given by (Giunchiglia et al. 1997) and the notions of locality and compatibility are discussed. In the interesting paper by (McGuiness et al. 2004) automated reasoning using SAT-solvers for class hierarchies is discussed. That is a separate case from the one we are investigating, because WordNet is not an ontology in the sense that there is a strict subsumption relationship between all connected terms. The paper by (Serafini et al., 2003) also investigates semantic matching using SAT and class hierarchies. Some of the inspiration how to design and formalize our logical representation comes from (Gamut, 1991) that describes intensional logic. A variety of different ontology-change operations are classified and described by (Noy & Klein, 2004). Much of the motivation why we need a system that can evolve multiple ontologies is given in (Zurawski 2004).

Conclusions

It will be important for many applications to be able to support many ontologies, that all can evolve at the same time as consistency is maintained between them. We have proposed an approach that uses a logical formalization that consists both of contexts and of states. Every local context has its own individual ontology, and it can evolve – this moves it into the next state. We have already implemented a part of the system (in Java) and when the whole system will be implemented we will evaluate the scalability by running some experiments. Our approach is an alternative to the model theoretical approach where SAT-solvers are used. Many of the systems described in the literature usually only allow for a few types of ontology mappings whereas our ontology mapping language is relatively rich. We have to investigate how this approach compares to other approaches (such as SAT-solving) and investigate how well it scales in cases when there are extensive amounts of ontology mappings that have to be taken into account. The problem of maintaining consistency between multiple evolving ontologies might seem to be intractable, but by adapting the reasoner to the unique properties of the problem, we might make the problem tractable (but experimental evaluation is needed as well). Finally, we believe that these methods could become one of the components in the design of an organizational distributed knowledge management system or some other knowledge infrastructure that will become valuable in the upcoming era of the knowledge society.

Acknowledgements

This research was funded by the Marcus Wallenberg Foundation for Education in International Industrial Enterprise. The author would like to thank Dave Robertson and Jessica Chen-Burger (both at CISA) for their valuable comments and feedback.

References

Bouquet, P., Ghidini, C., Giunchiglia, F., Blanzieri, E., "Theories and uses of context in knowledge representation and reasoning", IRST Technical Report 0110-28, Istituto Trentino di Cultura, October, 2001.

Bouquet, P., Giunchiglia, F., van Harmelen, F.,Serafini, L., Stuckenschmidt, H. "C-OWL: Contextualizing Ontologies", *Proceedings of the Second International Semantic Web Conference*, K. Sekara and J. Mylopoulis (Ed.), pp 164-179, LNCS. Springer Verlag, October, 2003.

Gamut, L. T. F. "Logic, Language, and Meaning, Volume 2: Intensional Logic and Logical Grammar". The University of Chicago Press, 1991.

Giunchiglia, F., "Contextual reasoning", In: *Epistemologia - Special Issue on I Linguaggi e le Macchine*, XVI, pp 345-364, 1993.

Giunchiglia F., Bouquet P., "Introduction to contextual reasoning. An Artificial Intelligence Perspective", In: *Perspectives on Cognitive Science*, B. Kokinov (ed.), 3, NBU Press, Sofia (Bulgaria), 1997.

McGuiness, D. L., Shvaiko, P., Giunchiglia, F., da Silva, P. P., "Towards explaining semantic matching". Technical Report DIT-04-019, Informatica e Telecomunicazioni, University of Trento, 2004.

Noy, N. F. and Klein, M., "Ontology evolution: Not the same as schema evolution". In: *Knowledge and Information Systems*, 6(4), pp 428-440, 2004.

Serafini, L., Bouquet, P., Magnini, B., Zanobini, S., "An algorithm for matching contextualized schemas via SAT". Technical Report DIT-03-003, Informatica e Telecomunicazioni, University of Trento, 2003.

Zurawski, M., "Towards a context-sensitive distributed knowledge management system for the knowledge organization", *Workshop on Knowledge Management and the Semantic Web*, 14th International Conference on Knowledge Engineering and Knowledge Management. EKAW 2004, Northamptonshire, UK, October, 2004.