Statement of Interest: Towards Ontology Language Customization

Heiner Stuckenschmidt
Center for Computing Technologies
University of Bremen

1 Motivation

It has been argued that intelligent applications benefit from the use of ontologies encoding background knowledge about the structure of a domain and the meaning of terms occurring therein. Prominent examples can be found in the following application areas:

Systems Engineering: The use of ontologies for the description of information and systems has many benefits. The ontology can be used to identify requirements as well as inconsistencies in a chosen design. It can help to acquire or search for available information. Once a systems component has been implemented its specification can be used for maintenance and extension purposes.

Information Integration: An important application area of ontologies is the integration of existing systems. The ability to exchange information at run time, also known as interoperability, is an important topic. In order to enable machines to understand each other we also have to explicate the vocabulary of each system in terms of an ontology.

Information Retrieval: Common information-retrieval techniques either rely on a specific encoding of available information (e.g. fixed classification codes) or simple full-text analysis. Both approaches suffer from severe shortcomings. Using an ontology in order to explicate the vocabulary can help overcome some of these problems. When used for the description of available information as well as for query formulation an ontology serves as a common basis for matching queries against potential results on a semantic level.

These application areas come with completely different requirements concerning the modeling and reasoning abilities of the ontology language used. In turn, existing ontology languages are rather generic, because they aim at providing modeling facilities independent of a concrete application. In principle, being generic is an advantage, because a generic language covers broader range of applications. However, the use of generic languages turns out to produce problems in real-life applications. These problems include the following:

- *Natural distinctions of an application domain are not supported by the language:* In the design of knowledge-based systems the distinction between tasks and methods (and their ontologies) is an important one.
- *The use of modeling constructs in a concrete application is not obvious:* A re-occurring discussion in ontological modeling is whether to represent a domain item as a class or an instance.
- *Small languages are not used, because implicitly representable constructs are overlooked:* The concept of disjointness of a set of classes can be modeled by negation and Implication.
- *Unused language features lead to unnecessary complexity of the language:* Transitive slots are a modeling construct not used too often that is hard to handle with respect to inference.

We argue that ontology language can gain practical relevance if they would address these problems. A promising way is to provide a framework that allows to customize an ontology language with respect to a given application. A customized language should cover the natural distinctions of a domain and provide guidance for the use of language constructs. Further it should be designed as an optimal trade off between reasoning expressiveness and reasoning complexity.

2 The Representation-Reasoning Trade-Off of Ontology Languages

We exemplify the representation-reasoning trade-off of ontology languages and its impact on the application of the language using ontology languages that are based on description logics. The rational for this choice is:

- The expressiveness and complexity of these languages has been studied thoroughly and well-founded results are available [Donini *et al.*, 1991]
- It has been shown that description logics provide a unifying framework for many class-based representation formalisms [Calvanese *et al.*, 1999].
- Description logic-based languages have become of interest in connection with the semantic web. the languages OIL [Fensel *et al.*, 2000] and DAML-ONT [McGuinness *et al.*, 2000] are good examples.

We compared three description logic languages that have been used to build ontologies, i.e. CLASSIC, LOOM and OIL. The results of the comparison are depicted in figure 1.

	CLASSIC	OIL	LOOM
Logical Operators			
conjunction	×	×	×
disjunction		×	×
negation		×	×
Slot-Constraints			
slot values	×		×
type restriction	×	×	×
range restriction	×	×	×
existential restriction	×	×	×
cardinalities	×	×	×
Assertions			
entities	×	(×)	×
relation-instances	×	(×)	×

Figure 1: Expressiveness of some description logic based ontology languages

The comparison reveals an emphasis on highly expressive concept definitions. The languages compared are capable of almost all common concept forming operators. An exception is CLASSIC that does not allow the use of disjunction and negation in concept definitions. The reason for this shortcoming is the existence of a sound and complete subsumption algorithm that support A-box reasoning [Borgida and Patel-Schneider, 1994]. LOOM on the other hand is a very expressive language containing all language constructs used in the comparison. The price for this high expressiveness is a loss in reasoning support: Soundness and completeness of the subsumption algorithms cannot be guaranteed [Horrocks, 1995].

The OIL approach is a first attempt to overcome the problems that arise from the representation-reasoning trade-off by defining a family of languages of different complexity. While the purpose of the smallest language of the OIL family (*Core OIL*) is to define a well-founded semantics for schemas of the emerging web standard RDF. This language is rather small and therefore allows efficient reasoning. The main language *Standard OIL* is tailored to have efficient reasoning support for consistency checking and for automatic construction of inheritance hierarchies for an extremely expressive logic. However this language does not include assertional language, because this would disable the reasoning support. For applications where instances are required, the OIL defines the language *Instance-OIL* that includes instances, but has no reasoning support.

3 The Design Space of Ontology Languages

The OIL framework allows a user to select between languages of different expressive power, however it does not address the problem of tailoring a language to a given application. Our main objective is that the current architecture of the OIL framework does only allow for strict extensions excluding the possibility to define alternative language that only partially overlap.

In order to allow more flexible variations we have to investigate the design space of ontology languages. There are many options to be taken into account. We could rely on previous work on comparing frame-based and terminological knowledge representation systems [Karp, 1993; Heinsohn *et al.*, 1994]. As our concerns are rather application driven than of a theoretical nature, we have to abstract from the technical details of the languages that are mainly concerned in the work mentioned above. We therefore concentrate on the following questions:

- What kinds of knowledge have to be modeled ?
- Which reasoning tasks have to be performed ?
- Which level of complexity is acceptable ?

The answers to these questions depend on the purpose of the language. They constitute dimensions of the design space: Different types of knowledge can be used for different kind of reasoning tasks. Further different kinds of reasoning methods result in different levels of reasoning complexity. In order to customize a language, we have to locate it with respect to each of these dimensions. Possible locations are further restricted by the needs and possibilities of the application environment. Examples for further design constraints are:

- The conceptualization of the application domain as well as pre-existing models implies the existence of certain knowledge types. The designed language must at least implicitly support these knowledge types.
- The role of the ontology in the overall application implies a certain task type. The design space is therefore restricted to variations of this task type.
- The availability of reasoners for the given task does not only have impact on the reasoning complexity, but also on the types of knowledge that can be used to define the ontology.

These restrictions have to be taken into account when the design space is explored. The situation becomes complicated, because the dimensions are not independent of each other. We already mentioned the interrelation of modeling primitives and reasoning support. In order to resolve such conflicts an engineering method is needed to guide the search process and validate the result.

4 A Pattern-Based Approach

We propose an engineering approach for customizing ontology languages that is based on the notion of ontology patterns. A pattern denote a language construct with special properties with respect to structure, semantics and inference capabilities. In a related approach Staab an colleagues propose the use of *semantic patterns* to support ontology engineering and propose a set of such patterns [Staab and Maedche, 2000]. The idea of providing a set of simple patterns that can be combined to form more complex patterns on which languages can be based is essential, because it makes different customized languages comparable and

provides a basis for translations across these languages. As already mentioned, we restrict ourselves to description-logic based languages. Therefore, our set of modeling primitives to start with are the concept forming operators of these kinds of languages.

Relying on description logics we already get a notion of more complex patterns in terms of special logics. These logics result from the combination of operators. The name of the language and therefore the pattern is a combination of the identifiers of the operators included. One of the most well known patterns is $\mathcal{ALC}$ the description logic containing Boolean operations on class expressions as well as universal and existential restrictions on slot fillers. The pattern used to resemble different class-based representation formalisms in [Calvanese *et al.*, 1999] is $\mathcal{ALUNI}$ which contains the corresponding operators: conjunction, disjunction, negation, universal restrictions on slot fillers, quantified number restrictions and inverse slots. $\mathcal{SHIQ}$, the logic underlying OIL also contains existential restrictions, transitive slots and conjunction of slot definitions. Theoretical results from the field of description logics provide us with the knowledge about decidable combinations of modeling primitives and their complexity with respect to subsumption reasoning. Consequently, every decidable combination of operators is a potential pattern that can be used to build the ontology for a certain application. In the course of the engineering process we have to handle different patterns:

Reasoner Patterns describe the language a certain reasoner is able to handle.

Reuse Patterns describe the language a useful, already existing ontology in encoded in.

Acquisition Patterns describe the language needed to encode acquired knowledge.

The Goal Pattern describes the language that will be designed.

In order to find the goal pattern, we have to find an optimal trade-off between the other patterns involved. For this purpose we invent the notion of coverage for ontology patterns. A Pattern P_1 is said to cover a pattern P_2, if all modeling primitives from P_2 are also contained in $P2_1$ or can be simulated by a combination of modeling primitives from P_1. We denote the fact that P_1 covers P_2 as $P_2 \prec P_1$. Using the notion of coverage we can now define the customization task.

Definition: Customization Task. A customization task is defined by a three tuple $\langle \mathcal{R}, \mathcal{U}, \mathcal{A} \rangle$ where $\mathcal{R}$ is a set of reasoner patterns, $\mathcal{U}$ a set of reuse patterns and $\mathcal{A}$ a set of acquisition patterns. The pattern G is a solution of the customization task if it is the minimal pattern that is covered by a reasoner pattern and covers all reuse and acquisition patterns, or formally:

Suitability of the goal pattern:

$$suitable(G) \iff \exists R \in \mathcal{R}(G \prec R) \wedge \forall P \in \mathcal{U} \cup \mathcal{A}(P \prec G)$$

Minimality of the goal pattern:

$$minimal(G) \iff \neg \exists G'(suitable(G') \wedge G' \prec G)$$

Solution:

$$solution(G) \iff suitable(G) \wedge minimal(G)$$

This definition provides us with an idea of the result of the customization process. However there are still many technical and methodological problems. We have to investigate the nature of the covering predicate and develop an algorithm for deciding whether one pattern covers the other. Further, the customization process has to be implemented. It is quite likely that the acquisition pattern will not be completely available in the beginning. Therefore we have to incorporate user interaction and revisions of previous decisions. Finally, results have to be generalized beyond the scope of description logics which will be difficult, because there are less theoretical results to build upon.

References

[Borgida and Patel-Schneider, 1994] A. Borgida and P.F. Patel-Schneider. A semantics and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research*, 1(2):277–308, 1994.

[Calvanese *et al.*, 1999] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Unifying class-based representation formalisms. *Journal of Artificial Intelligence Research*, 11:1999–240, 1999.

[Donini *et al.*, 1991] F.M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen Sandewall, R. Fikes, and E., editors, *2nd International Conference on Knowledge Representation and Reasoning, KR-91*. Morgan Kaufmann, 1991.

[Fensel *et al.*, 2000] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein. Oil in a nutshell. In *12th International Conference on Knowledge Engineering and Knowledge Management EKAW 2000*, Juanles-Pins, France, 2000.

[Heinsohn *et al.*, 1994] J. Heinsohn, D. Kudenko, B. Nebel, and H.-J. Profitlich. An empirical analysis of terminological representation systems. *Artificial Intelligence*, 68(2):367–397, 1994.

[Horrocks, 1995] Ian Horrocks. *A Comparison of Two Terminological Knowledge Representation Systems*. Master's thesis, University of Manchester, 1995.

[Karp, 1993] Peter D. Karp. The design space of frame knowledge representation systems. Technical Note 520, AI Center SRI International, May 5 1993.

[McGuinness *et al.*, 2000] D. McGuinness, R. Fikes, D. Connolly, and L.A. Stein. Daml-ont: An ontology language for the semantic web. *IEEE Intelligent Systems*, 2000. Submitted to Special Issue on Semantic Web Technologies.

[Staab and Maedche, 2000] Steffen Staab and Alexander Maedche. Ontology engineering beyond the modeling of concepts and relations. In *Proceedings of he ECAI'2000 Workshop on Applications of Ontologies and Problem-Solving Methods*, Berlin, Germany, 2000.