Ontology Language Integration: A Constructive Approach

Heiner Stuckenschmidt, Jérôme Euzenat

Ontology Language Integration: A Constructive Approach

Heiner Stuckenschmidt[1] **and Jérôme Euzenat**[2]

[1]Center for Computing Technologies,University of Bremen
email: heiner@tzi.de

[2]INRIA Rhônes-Alpes, Grenoble
email: Jerome.Euzenat@inrialpes.fr

Aug 24th 2001

Abstract

The problem of integrating different ontology languages has become of special interest recently, especially in the context of semantic web applications. In the paper, we present an approach that is based on the configuration of a joint language all other languages can be translated into. We use description logics as a basis for constructing this common language taking advantage of the modular character and the availability of profound theoretical results in this area. We give the central definitions and exemplify the approach using example ontologies available on the Web.

1 Motivation

It has been widely recognized that information systems benefit from the use of formal ontologies. These ontologies are used to conceptualize and structure information as well as to provide intelligent search facilities and integration methods for remote information. While ontologies per se already support conceptualization and structuring, applications like intelligent search and information integration make it possible to reason about the knowledge specified in the ontologies. This requirement, in turn requires the ontology to be implemented in a machine processable language. Thus, the question of using ontologies in information systems is also a question of the language used to encode the ontologies.

The World Wide Web is the largest information system ever. Its size and heterogeneity makes ontology based search and integration even more important than in other information systems. In this context the "semantic web" [Berners-Lee et al., 2001] is mentioned. It is supported by the annotation of web pages, containing informal knowledge as we know it now, with formal knowledge. These documents can reference each other and depend on background knowledge. Taking advantage of the semantic web requires to be able to gather, compare, transform and compose the annotations. For several reasons (legacy knowledge, ease of use, heterogeneity of devices and adaptability, timelessness), it is not likely that this formal knowledge will be encoded in the very same language. The interoperability of formal knowledge languages must then be studied in order to interpret the knowledge acquired through the semantic web.

2 Language Construction

In the words of Tim Berners-Lee, the semantic web requires a set of languages of increasing expressiveness and anyone can pick up the right language for each particular semantic web application. This is what has been developed by the description logic community over the years. Our approach focuses on ontology languages that rely on these logics. The rationale for this choice is the following:

- The expressiveness and complexity of these languages has been studied thoroughly and well-founded results are available [Donini et al., 1991],[Donini et al., 1994]
- 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 the DAML language [McGuinness et al., 2001] are good examples.

A modular family of languages is a set of languages made from a set of operations (constituting an algebraic base) that can be combined. Since the languages have a similar kind of semantic characterization, it is easier to transform a representation from one language to another and one can take advantage of efficient provers or expressive languages.

2.1 Customized Languages

Relying on description logics we already get a notion of a special language from the combination of operators. 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 language patterns:

Reasoner Languages the languages that available reasoners are able to handle.

Legacy Languages the language a useful, already existing ontology is encoded in.

Acquisition Languages are languages needed to encode acquired knowledge.

The Goal Language describes a language that can act as an interlingua for the ontologies to be integrated. It represents a trade-offs between expressivity constraints of the legacy and acquisition languages and the complexity constraints of the reasoner languages.

In order to find the goal language, we have to find an optimal trade-off between the other languages involved. For this purpose we invent the notion of coverage for languages. A language L' is said to cover a language L, if there is a transformation from L to L' that preserves consequence. In particular, this is the case if all modeling primitives from L are also contained in L' or can be simulated by a combination of modeling primitives from L'. We denote the fact that L' covers L as $L \prec L'$. Using the notion of coverage we can now define the customization task.

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

$$\exists R \in \mathcal{R}(G \prec R) \wedge \forall P \in \mathcal{U} \cup \mathcal{A}(P \prec G) \tag{1}$$

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. We introduce different notions of coverage of increasing strength that is based on transformations in the next section.

2.2 A Transformation-Based Approach

The notion of transformability is a central one in our approach because it allows to define the coverage relation. The simplest transformation is the transformation from a logic to another which adds new constructors. The transformation is then trivial, but yet useful, because the initial representation is valid in the new language.

In the following we use L and L' to refer to languages. Languages are sets of expressions δ. Representations r are sets of expressions which are normally subsets of a language. $r_L \subseteq L$. Transformations are mappings $\tau : L \rightarrow L'$ from expressions in one language to expressions in a different language.

Definition: Syntactic Coverage This trivial form of transformation provides us with a first notion of coverage that reflects the situation, where a language is the subset of the other:

$$L \overline{\ll} L' \Leftrightarrow_{def} (L \subseteq L') \tag{2}$$

For this case, one can define a special case of the coverage relation as $L \overline{\ll} L'$ which means that one language is completely included in the other in a lexical sense.

If $L \overline{\not\ll} L'$, the transformation is more difficult. The initial representation r can be restricted to what is (syntactically) expressible in L'. However, this operation (which is correct) is incomplete because it can happen that a consequence of a representation expressible in L' is not a consequence of the expression of that representation in L'. The preceding proposal is restricted in the sense that it only allows in the target language, expressions expressible in the source language, while there are equivalent non-syntactically comparable languages. This is the case of the description logic languages $\mathcal{ALC}$ and $\mathcal{ALUE}$ which are known to be equivalent while being defined by different operators. For that purpose, one can define $L \overline{\overline{\ll}} L'$ if and only if the models are preserved.

Definition: Semantic Coverage Transformations that simulate some operators of the transformed language using combinations of operators of the goal language imply a notion of coverage that is based on the semantics of languages. Let I be the a Tarskian style interpretation defining the model-theoretic semantics of expressions, then we get

$$L \overline{\overline{\ll}} L' \Leftrightarrow_{def} \forall I : I \models_L \delta \Rightarrow I \models_{L'} \tau(\delta) \tag{3}$$

Another possibility is to define $\widetilde{\ll}$ as the existence of an homomorphism between the models of the original and the translated language. This property guarantees that inconsistency of an expression in the target language implies inconsistency of the expression in the source language.

Definition: Model-Theoretic Coverage Transformations that preserves inconsistency which is an important property with respect to automated reasoning by guaranteeing that for every model in L there also is a model in L' define a special case of semantic coverage we refer to as model-theoretic coverage:

(4) $$L\widetilde{\ll}L' \Leftrightarrow_{def} \forall I \exists I' : I \models_L \delta \Rightarrow I' \models_{L'} \tau(\delta)$$

Summarizing, the syntactic and semantic structure of a language family provides us with different criteria for coverage all based on the notion of transformability. These notions of coverage do not only give us the possibility to identify and prove coverage, it also specifies a mechanisms for transforming the covered into the covering language. Therefore we not only show that a suitable language can be generated, but also how the generation is being performed. In the next section we present an implementation of this approach.

3 Transformation-based Ontology Integration

The notion of semantic interoperability is a very broad one since it covers almost all application of the semantic web. Therefore we can only give evidence for the usefulness of the 'family of languages' approach by example.

3.1 An Example Problem

We chose a scenario where two existing ontologies should be integrated to establish a semantic model of an application domain. The library of the DAML (DARPA Agent Markup Language) contains an ontology describing a technical support application (http://www.daml.org/ontologies/69). It is encoded in the DAML-ONT language.

```
<Class ID="ProductInfo">
    <subClassOf
     resource="#IncommingTechSupIncident"/>
    <comment>
       Technical Product Information
    </comment>
```

```
</Class>

<Property ID="operatingSystem">
    <domain resource="#ProductInfo"/>
    <comment>Product's Operating System</comment>
    <default resource="MSWindows98"/>
</Property>

<Class ID="OperatingSystem">
    <oneOf parseType="daml:collection">
       <OperatingSystem ID="MSWindows2000"/>
       <OperatingSystem ID="MSWindowsNT"/>
       <OperatingSystem ID="MSWindows98"/>
       <OperatingSystem ID="MSWindows95"/>
    </oneOf>
    <comment>
       Available Operating Systems
    </comment>
</Class>

<Property ID="productVersion">
    <domain resource="#ProductInfo"/>
    <comment>Product's Version</comment>
    <default resource="#PersonalEdition"/>
</Property>

<Class ID="ProductVersion">
    <oneOf parseType="daml:collection">
       <ProductVersion ID="EnterpiseEdition"/>
       <ProductVersion ID="DeveloperEdition"/>
       <ProductVersion ID="ProfessionalEdition"/>
       <ProductVersion ID="SmallBusinessEdition"/>
       <ProductVersion ID="PersonalEdition"/>
    </oneOf>
    <comment>Available Product Versions</comment>
</Class>
```

Since the DAML language borrows from description logics [McGuinness et al., 2001, Horrocks, 2000] these constructs can easily be mapped on operators available in the description logic markup language DLML. Operators used in this specific ontology are: atomic names, primitive classes, primitive roles, domain restrictions for assigning properties to classes and the one-of operator for defining classes by enumeration.

We assume that the technical support should be extended to include hardware as

well. For this purpose definitions of existing hardware products have to be integrated into the ontology. As an example product we use the printer ontology that can be found at http://www.ontoknowledge.org/oil/case-studies/. This ontology in turn is encoded in the OIL language.

```
<oil:DefinedClass rdf:ID="HPLaserJet1100Series">
    <rdfs:subClassOf>
       <oil:And>
          <oil:hasOperand>
             <rdfs:Class
              rdf:about="#HPLaserJetPrinter"/>
          </oil:hasOperand>
          <oil:hasOperand>
             <rdfs:Class
              rdf:about="#PrinterForPersonalUse"/>
          </oil:hasOperand>
       </oil:And>
    </rdfs:subClassOf>
    <oil:hasPropertyRestriction>
       <oil:HasValue>
          <oil:onProperty
           rdf:resource="#PrintingSpeed"/>
          <oil:toClass>
             <rdfs:Class
              rdf:about="#&amp;quot;8 ppm&amp;quot;"/>
          </oil:toClass>
       </oil:HasValue>
    </oil:hasPropertyRestriction>
    <oil:hasPropertyRestriction>
       <oil:HasValue>
          <oil:onProperty
           rdf:resource="#PrintingResolution"/>
          <oil:toClass>
             <rdfs:Class
              rdf:about="#&amp;quot;600 dpi&amp;quot;"/>
          </oil:toClass>
       </oil:HasValue>
    </oil:hasPropertyRestriction>
</oil:DefinedClass>
```

The semantics of the OIL language is completely specified in terms of description logics. Consequently, we can directly map OIL constructs. Operators used in the model are the following: atomic names primitive concepts, primitive roles, existential

restrictions on slot values as well as conjunction for multiple inheritance and multiple slot constraints.

3.2 Integrating the Specifications

Using the family of languages approach, we can integrate the two specifications in a three step process. First, we have to analyze the language patterns (i.e. combinations of operators) at hand then we customize a joint language. Finally, we define and implement transformations between the language patterns and the customized language.

Step 1: Identify Language Patterns The languages used in the specifications from our example, i.e. DAML and OIL are legacy language in the sense of the language customization task. As these languages are very expressive, however for our purpose we only have to care about the part of the languages that are really used in the specifications (see last section).

The second kind of language patterns involved are defined by the aim of providing reasoning support for the integrated specifications. A potential reasoner is the FaCT system that supports two different languages, $\mathcal{SHF}$ and $\mathcal{SHIQ}$ [Horrocks et al., 1999]. Figure 1 illustrates the expressiveness of these languages.

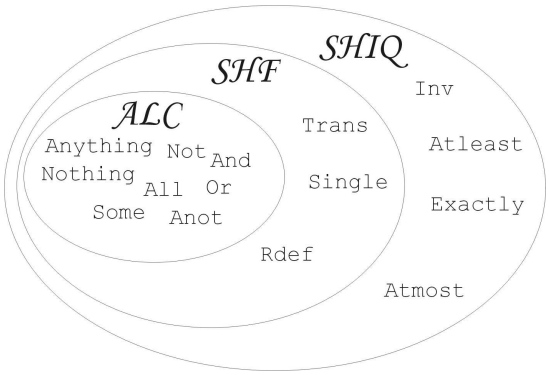

Figure 1: Expressiveness of languages supported by the FaCT reasoner

Step 2: Customize Integration Language The patterns identified in step 1 act as an input for the language customization step. We denote the language pattern used for the technical support ontology as L_{DAML}, the one used for the printer ontology as L_{OIL}. In the example case, we can simply merge the two language patterns into a language that consists of all operators found in both models. The resulting integration language denoted as $L_G = L_{DAML} \overline{\vee} L_{OIL}$ simply consists of the union of

the operators present in the two ontologies described above.

We now have to test the suitability of the pattern. For the suitability we have to check, whether the language covers the legacy languages, i.e. whether $L_G \prec L_{DAML} \wedge L_G \prec L_{OIL}$ holds. In this case this is obvious, because L_G extends both languages ($L_{OIL} \overline{\ll} L_G$ and $L_{DAML} \overline{\ll} L_G$). Additionally we have to make sure that L_G is covered by at least one reasoner language (denoted as $L_{\mathcal{SHF}}$ and $L_{\mathcal{SHIQ}}$). We can show that $L_{\mathcal{SHIQ}}$ covers L_G: Concept and role definitions as well as conjunction and existential restriction are already directly contained in $\mathcal{ALC}$ while we have to model `one-of` and `domain` using other operators of the languages supported. This can be done in the very same way as it is done from OIL to the FaCT reasoner [Horrocks, 2000]:

one-of can be simulated using `or`, `not` and `cdef`, because the transformation from $(\texttt{one} - \texttt{of}\ C_1\ C_2)$ to $(\texttt{or}\ C_1\ C_2) \wedge (\texttt{cdef}\ C_1(\texttt{not}\ C_2)))$ preserves inconsistency checking by guaranteeing that for every model for the original expression there is a model in the transformed one. We obtain consequence preservation for this transformation.

domain can be simulated using `all` and `inv`, because

$$R \leq (\texttt{domain}\ C) \Rightarrow \top \leq (\texttt{all}\ (\texttt{inv}\ R)\ C)$$

Another way is to use general concept implications or the form:

$$(\texttt{some}\ R\ \top) \leq C$$

We omit the proofs due to the limited space.

Performing the first transformation we have to use the $\mathcal{SHIQ}$ reasoner, because inverse roles are needed to simulate domain constraints. If we do the second transformation, we can even rely on the $\mathcal{SHF}$ reasoner which supports a smaller language and therefore is able to provide faster reasoning service.

To summarize, we can use the language L_G created by the transformations as a language for the integrated model, because there is a reasoner language (i.e. $L_{\mathcal{SHIQ}}$) that syntactically covers L_G.

Step 3: Implement Languages and Transformations This can be refined in three sub-steps:

1. Translating from DAML and OIL to L_{DAML} and L_{OIL};
2. Providing the transformation from L_{DAML} and L_{OIL} to L_G;

3. Translating from $L_{\mathcal{SHIQ}}$ which syntactically covers L_G to $\mathcal{SHIQ}$.

The implementation has been carried out by transforming representations within the DLML (Description Logic Markup Language [Euzenat, 2001]) framework. It encodes many description logics in XML in a coherent way (same operators have the same name) but does not offer CGI. Transformations are written in the XSLT language for transforming XML documents.

The second one is more related to description logics. It first involves merging both ontologies. This is easily achieved with a straightforward transformation, thanks to the unified vocabulary provided by DLML [Euzenat, 2001](i.e. whatever the logic, the syntax is the same). The resulting logic ($L_{DAML}\overline{\vee}L_{OIL}$) being syntactically stronger than L_{DAML} and L_{OIL} preserves the content of the ontologies as well as the consequence relation.

Then, the resulting merged ontology, which cannot be directly translated into $\mathcal{SHIQ}$ is converted by applying successive transformations (again written in XSLT). The first one eliminates the `domain` constructor and the second one eliminates the `one-of` constructor in exactly the way put forth above. Because the first transformation preserves the models and the second one preserves unsatisfiability, then, the whole chain of transformation preserve consequence.

4 Discussion

We introduced an approach for ontology language integration that is based on the construction of a joint language and the use of semantics-preserving transformations. We outlined the idea of the approach and gave evidence for its suitability using a real-life example.

The approach presented still has several shortcomings implying needs for further research. First of all the nature of different kinds of transformation needs further investigation. We envision a formal framework for proving special properties of transformation in order to guarantee formal properties of the constructed language. When thinking of a web of trust, it is also beneficial to annotation complete proofs to transformed language as a guarantee that no information has been lost.

Another very important related problem which is completely out of the scope of this paper is the problem of translating not only between different representation languages, but also between different terminologies. An approach able to perform translations between different ontologies on the language and the terminology level would be a big step forward.

References

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific Amercan*, 284(5):35–43.

[Calvanese et al., 1999] Calvanese, D., Lenzerini, M., and Nardi, D. (1999). Unifying class-based representation formalisms. *Journal of Artificial Intelligence Research*, 11:1999–240.

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

[Donini et al., 1994] Donini, F., Lenzerini, M., Nardi, D., and Schaerf, A. (1994). Deduction in concept languages: from subsumption to instance checking. *Journal of logic and computation*, 4(4):423–452.

[Euzenat, 2001] Euzenat, J. (2001). Preserving modularity in xml encoding of description logics. In McGuinness, D., Patel-Schneider, P., Goble, C., and Mller, R., editors, *Proc. 14th workshop on description logics (DL), Stanford (CA US)*, pages 20–29.

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

[Horrocks, 2000] Horrocks, I. (2000). A denotational semantics for Standard OIL and Instance OIL. http://www.ontoknowledge.org/oil/downl/semantics.pdf.

[Horrocks et al., 1999] Horrocks, I., Sattler, U., and Tobies, S. (1999). Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, pages 161–180. Springer-Verlag.

[McGuinness et al., 2001] McGuinness, D., Fikes, R., Connolly, D., and Stein, L. (2001). Daml-ont: An ontology language for the semantic web. *IEEE Intelligent Systems*. Submitted to Special Issue on Semantic Web Technologies.