A Layered Model for Building Ontology Translation Systems

Oscar Corcho, Intelligent Software Components, Spain*

Asunción Gómez-Pérez, Universidad Politécnica de Madrid, Spain

ABSTRACT

In this paper we present a model for building ontology translation systems between ontology languages and/or ontology tools, where translation decisions are defined at four different layers: lexical, syntax, semantic, and pragmatic. This layered approach provides a major contribution to the current state of the art in ontology translation, since it makes ontology translation systems easier to build and understand and, consequently, to maintain and reuse. As part of this model, we propose a method that guides in the process of developing ontology translation systems according to this approach. The method identifies four main activities: feasibility study, analysis of source, and target formats, design, and implementation of the translation system, with their decomposition in tasks, and recommends the techniques to be used inside each of them.

Keywords: ontologies; semiotics; transformation languages; transformation models

INTRODUCTION

An ontology is defined as a "formal explicit specification of a shared conceptualization" (Studer et al., 1998); that is, an ontology must be machine readable (it is formal), all its components must be described clearly (it is explicit), it describes an abstract model of a domain (it is a conceptualization), and it is the product of a consensus (it is shared).

Ontologies can be implemented in varied ontology languages, which are usually divided in two groups: classical and ontology markup languages. Among the classical languages used for ontology construction, we can cite (in alphabetical order): CycL (Lenat & Guha, 1990), FLogic (Kifer et al., 1995), KIF (Genesereth & Fikes, 1992), LOOM (MacGregor, 1991), OCML (Motta, 1999), and Ontolingua (Gruber, 1992). Among the

ontology markup languages used in the context of the Semantic Web, we can cite (in alphabetical order): DAML+OIL (Horrocks and van Harmelen, 2001), OIL (Horrocks et al., 2000), OWL (Dean & Schreiber, 2004), RDF (Lassila & Swick, 1999), RDF Schema (Brickley & Guha, 2004), SHOE (Luke & Hefflin, 2000), and XOL (Karp et al., 1999). Each of these languages has its own syntax, its own expressiveness, and its own reasoning capabilities provided by different inference engines. Languages also are based on different knowledge representation paradigms and combinations of them (frames, first order logic, description logic, semantic networks, topic maps, conceptual graphs, etc.).

A similar situation applies to ontology tools: several ontology editors and ontology management systems can be used to develop ontologies. Among them, we can cite (in alphabetical order): KAON (Maedche et al., 2003), OilEd (Bechhofer et al., 2001), OntoEdit (Sure et al., 2002), the Ontolingua Server (Farquhar et al., 1997), OntoSaurus (Swartout et al., 1997), Protégé-2000 (Noy et al., 2000), WebODE (Arpírez et al., 2003), and WebOnto (Domingue, 1998). As in the case of languages, the knowledge models underlying these tools have their own expressiveness and reasoning capabilities, since they are also based on different knowledge representation paradigms and combinations of them. Besides, ontology tools usually export ontologies to one or several ontology languages and import ontologies coded in different ontology languages.

There are important connections and implications between the knowledge modeling components used to build an ontology in such languages and tools, and the knowledge representation paradigms used to represent formally such components. With frames and first order logic, the knowledge components commonly used to build ontologies are (Gruber, 1993) classes, relations, functions, formal axioms, and instances; with description logics, they are usually (Baader et al., 2003) concepts, roles, and individuals; with semantic networks, they are: nodes and arcs between nodes; etc.

The *ontology translation problem* (Gruber, 1993) appears when we decide to reuse an ontology (or part of an ontology) with a tool or language that is different from those where the ontology is available. If we force each ontology-based system developer to commit individually to the task of translating and incorporating the necessary ontologies to the developer's system, the developer will need a lot of effort and time to achieve his or her objectives (Swartout et al., 1997). Therefore, ontology reuse in different contexts will be boosted highly, as long as we provide ontology translation services among those languages and/or tools.

Many ontology translation systems can be found in the current ontology technology. They are aimed mainly at importing ontologies implemented in a specific ontology language to an ontology tool, or at exporting ontologies modeled with an ontology tool to an ontology language. A smaller number of ontology translation systems is aimed at transforming ontolo-

gies between ontology languages or between ontology tools.

Since ontology tools and languages have different expressiveness and reasoning capabilities, translations between them are neither straightforward nor easily reusable. They normally require many decisions at different levels, which range from low layers (i.e., how to transform a concept name identifier from one format to the other) to higher layers (i.e., how to transform a ternary relation among concepts to a format that only allows representing binary relations between concepts).

Current ontology translation systems usually do not take into account such a layered structure of translation decisions. Besides, in these systems, translation decisions usually are hidden inside their programming code. Both aspects make it difficult to understand how ontology translation systems work.

To ameliorate this problem, in this chapter we propose a new model for building and maintaining ontology translation systems, which identifies four layers where ontology translation decisions can be made: lexical, syntax, semantic, and pragmatic. This *layered architecture* is based on existing work in formal languages and the theory of signs (Morris, 1938).

The following section describes the four layers where ontology translation problems may appear, with examples of how transformations have to be made at each layer; then we describe an ontology translation method based on the previous layers, which is divided into four main activities; finally, we present the main conclusions of our work and related work.

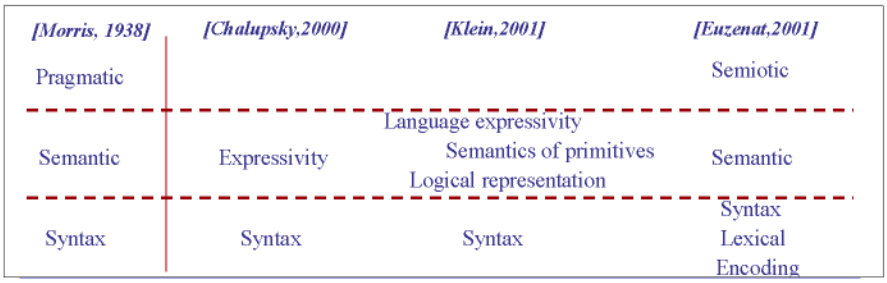
ONTOLOGY TRANSLATION LAYERS

As discussed previously, our ontology translation model proposes to structure translation decisions in four different layers. The selection of layers is based on existing work on formal languages and the theory of signs (Morris, 1938), which consider the existence of several levels in the definition of a language: syntax (related to how the language symbols are structured), semantics (related to the meaning of those structured symbols), and pragmatics (related to the intended meaning of the symbols; that is, how symbols are interpreted or used).

In the context of semantic interoperability, some authors have proposed classifications of the problems to be faced when managing different ontologies in possibly different formats. We will enumerate only the ones that are due to differences between the source and target formats[1]. Euzenat (2001) distinguishes the following non-strict levels of language interoperability: encoding, lexical, syntactic, semantic, and semiotic. Chalupsky (2000) distinguishes two layers: syntax and expressivity (aka semantics). Klein (2001) distinguishes four levels: syntax, logical representation, semantics of primitives, and language expressivity; the last three levels correspond to the semantic layer identified in the other classifications. Figure 1 shows the relationship between these layers.

The layers proposed in our model are based mainly on Euzenat, the only one in the context of semantic interoperability

Figure 1. Classifications of semantic interoperability problems and relationships between them



who deals with pragmatics (although he uses the term semiotics for it). However, we consider it unnecessary to split the lexical and encoding layers when dealing with ontologies and consider them as a unique layer, called lexical.

In the next sections we describe the types of translation problems that usually can be found in each of these layers and will show some examples of common transformations performed in each of them.

Lexical Layer

The lexical layer deals with the ability to segment the representation in characters and words (or symbols) (Euzenat, 2001). Different languages and tools normally use different character sets and grammars for generating their terminal symbols (i.e., ontology component identifiers, natural language descriptions of ontology components, and attribute values). This translation layer deals with the problems that may arise in these symbol transformations.

Therefore, in this layer, we deal with the following types of transformations:

- **Transformations of ontology component identifiers**. For instance, the source and target formats use different sets of characters for creating identifiers; the source and target format use different naming conventions for their component identifiers, or their components have different scopes; hence, some component identifiers cannot overlap with the identifiers of other components.
- **Transformations of pieces of text used for natural language documentation purposes**. For instance, specific characters in the natural language documentation of a component must be escaped since the target format does not allow them as part of the documentation.
- **Transformations of values**. For instance, numbers must be represented as character strings in the target format, or dates must be transformed according to the date formulation rules of the target format.

From a lexical point of view, among the most representative ontology languages

and tools we can distinguish three groups of formats:

- *ASCII-based formats*. Among these formats, we can cite the following classical languages: KIF, Ontolingua, CycL, LOOM, OCML, and FLogic. Also in this group, we can include the ontology tools related to some of these languages (Ontolingua Server, OntoSaurus, and WebOnto). These languages are based on ASCII encodings, and hence, the range of characters allowed for creating ontology component identifiers and for representing natural language texts and values is restricted to most of the characters allowed in this encoding.
- *UNICODE-based formats*. Among these formats, we can cite the following ontology tools: OntoEdit, Protégé-2000, and WebODE. These formats are based on the UNICODE encoding, which is an extension of the ASCII encoding and, thus, allows using more varied characters (including Asian and Arabic characters, more punctuation signs, etc.).
- *UNICODE&XML-based formats*. Among these formats we can refer to the ontology markup languages: SHOE, XOL, RDF, RDFS, OIL, DAML+OIL, and OWL, and some of the tools that are related to them, such as KAON and OilEd. These formats are characterized not only for being UNICODE compliant, as the previous ones, but also for restricting the use of some characters and groups of characters in the component identifiers and in the natural language documentation and values, such as the use of tag-style pieces of text (e.g., <*example*>) inside documentation tags. An important restriction is the compulsory use of qualified names (*QNames*) as identifiers of ontology concepts and properties, since they are used to construct tags when dealing with instances.

The easiest lexical transformations are usually those to be done from the first and third group of formats to the second one, which is the most unrestricted one. In other cases, the specific features of each format do not allow us to generalize the types of transformations to be done, which mainly consist in replacing non-allowed characters with others that are allowed, or in replacing identifiers that are reserved keywords in a format with other identifiers that are not. Obviously, there are also differences among the languages and tools inside each group, although the transformations needed in those cases are minimal.

Special attention deserves the problem related to the scope of the ontology component identifiers in the source and target formats, and to the restrictions related to overlapping identifiers. These problems appear when, in the source format, a component is defined inside the scope of another and, thus, its identifier is local to the latter, while the correspondent component has a global scope in the target format. As a consequence, there could be clashes of identifiers if two different components have the same identifier in the source format.

Table 1 shows examples of how some ontology component identifiers can be transformed from WebODE to

Ontolingua, RDF(S), OWL and Protégé-2000, taking into account the rules for generating identifiers in each format and the constraints about the scope and possible overlap of some ontology component identifiers.

As previously expressed, inside this layer, we also deal with the different naming conventions that exist in different formats[2]. For instance, in Lisp-based languages and tools such as Ontolingua, LOOM, OCML, and their corresponding ontology tools, compound names usually are joined together using hyphens (e.g., Travel-Agency). In tools like OntoEdit, Protégé, and WebODE, words are separated with blank spaces (e.g., Travel Agency). In ontology markup languages, the convention used for class identifiers is to write all the words together, with no blank spaces or hyphens, and with the first capital letter for each word (e.g., TravelAgency).

Syntactic Layer

This layer deals with the ability to structure the representation in structured sentences, formulas or assertions (Euzenat, 2001). Ontology components in each language or tool are defined with different grammars. Hence, the syntactic layer deals with the problems related to how the symbols are structured in the source and target formats, taking into account the derivation rules for ontology components in each of them.

In this layer, the following types of transformations are included:

- **Transformations of ontology component definitions** according to the grammars of the source and target formats. For instance, the grammar to define a concept in Ontolingua is different than that in OCML.
- **Transformations of datatypes.** For instance, the datatype date in WebODE must be transformed to the datatype &xsd;date in OWL.

Figure 2 shows an example of how a WebODE concept definition (expressed in XML) is transformed into Ontolingua and OWL. In this example, both types of translation problems are dealt with.

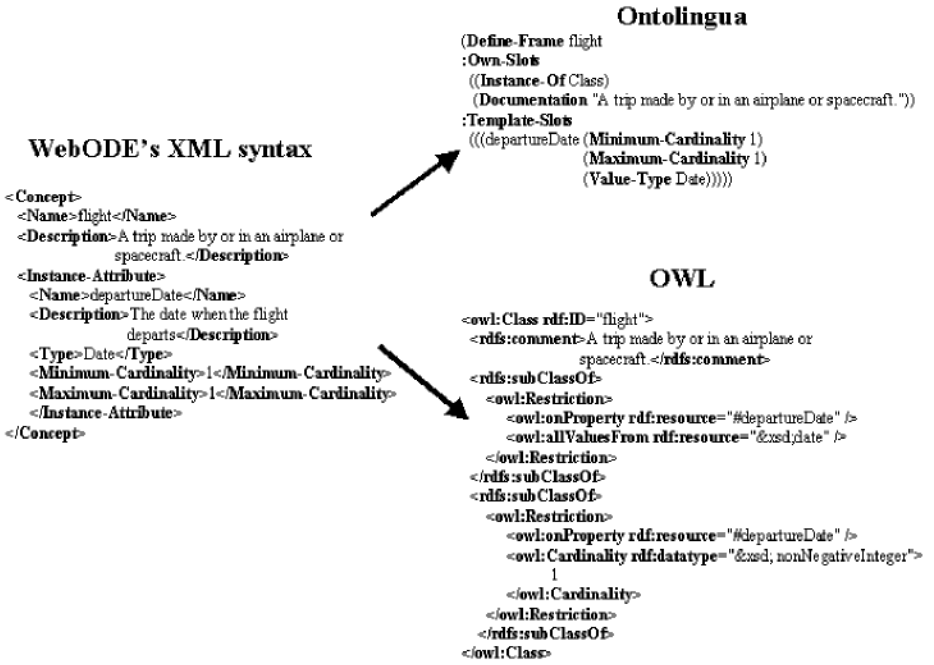
Among the most representative ontology languages and tools, we can distinguish the following (overlapping) groups of formats:

- **Lisp-based formats.** The syntax of several classical ontology languages are based on the Lisp language; namely, KIF and Ontolingua, LOOM, and OCML, together with their corresponding ontology tools (Ontolingua Server, OntoSaurus, and WebOnto, respectively).
- **XML-based formats.** Ontology markup languages are characterized by being represented in XML syntax. Among them, we can cite SHOE, XOL, RDF, RDFS, OIL, DAML+OIL, and OWL. In addition, ontology tools such as OntoEdit, Protégé-2000, and WebODE also provide ad hoc XML backends to implement their ontologies.
- **Ad hoc text formats.** There are other ontology languages that do not provide any of the previous syntaxes, but they

Table 1. Examples of transformations at the lexical layer

WebODE Identifier	Target	Result	Reasons for Transformation
`Business Trip`	Ontolingua	`Business-Trip`	Blank spaces in identifiers are not allowed in Ontolingua
`1StarHotel`	RDF(S)	`OneStarHotel`	Identifiers cannot start with a digit in RDF(S). They do not form valid QNames
Concepts `Name` and `Name`	Ontolingua	classes `Name` and `Name_1`	Ontolingua is not case sensitive
Concept `Room` attribute `fare` Concept `Flight` attribute `fare`	OWL	classes `Room`, `Flight` datatypeProperty `roomFare` datatypeProperty `flightFare`	WebODE attributes are local to concepts. OWL datatype properties are not defined in the scope of OWL classes, but globally
Concept `Name` attribute `Name`	Protégé-2000	class `Name`; slot `name`	The identifiers of classes and slots cannot overlap in Protégé-2000

Figure 2. Examples of transformations at the syntactic layer



provide their own ad hoc formats. These languages are F-Logic, the ASCII syntax of OIL, and the Notation-3 (N3) syntax used to represent ontologies in RDF, RDFS, and OWL. Except for F-Logic, these syntaxes are alternative and mainly intended for human consumption.

- **Ontology management APIs.** Finally, several ontology languages and tools provide ontology management APIs. These APIs are included here because they can be considered as another form of syntax; the expressions used to access, create, and modify ontology components in the programming language

in which these APIs are available have to be created according to the specification provided by the API. Among the languages with an ontology management API, we have all the ontology markup languages, where ontologies can be created using available XML Java APIs such as DOM, SAX, and so forth; and, more specifically, RDF, RDFS, DAML+OIL, and OWL, for which there are specific APIs that resemble the knowledge models of the ontology languages, such as Jena, the OWL API, and so forth. Among the tools, we have KAON, OntoEdit, Protégé-2000, and WebODE.

There are other aspects to be considered in this layer, such as the fact that some ontology languages and tools allow defining the same component with different syntaxes. For example, Ontolingua provides at least four different ways to define concepts using KIF, using the Frame Ontology or using the OKBC-Ontology exclusively, or embedding KIF expressions inside definitions that use the Frame Ontology. This variety adds complexity both for the generation of such a format (we must decide what kind of expression to use[3]) and for its processing (we have to take into account all the possible syntactic variants for the same piece of knowledge).

Inside this layer, we also must take into account how the different formats represent datatypes. Two groups can be distinguished:

- *Formats with their own internal datatypes*. Among these formats, we can refer to most of the ontology languages except RDF, RDFS, and OWL, and most of the ontology tools.
- *Formats with XML Schema datatypes*. These datatypes have been defined with the aim of providing datatype standardization in Web contexts (e.g., in Web services). They can be used in the ontology languages RDF, RDFS, and OWL, and in the ontology tool WebODE, which allows using both types of datatypes (internal and XML Schema).

Therefore, with regard to datatypes, the problems to be solved will consist mainly of finding the relationships between the internal datatypes of the source and target formats (not all the formats have the same group of datatypes) or finding relationships between the internal datatypes of a format and the XML Schema datatypes, and vice versa.

Semantic Layer

This layer deals with the ability to construct the propositional meaning of the representation (Euzenat, 2001). Different ontology languages and tools can be based on different KR paradigms (frames, semantic networks, first order logic, conceptual graphs, etc.) or on combinations of them. These KR paradigms do not always allow expressing the same type of knowledge, and sometimes the languages and tools based on these KR paradigms allow expressing the same knowledge in different ways.

Therefore, in this layer, we deal not only with simple transformations (e.g.,

WebODE concepts are transformed into Ontolingua and OWL classes), but also with complex transformations of expressions that usually are related to the fact that the source and target formats are based on different KR paradigms (e.g., WebODE disjoint decompositions are transformed into subclass-of relationships and PAL[4] constraints in Protégé-2000, WebODE instance attributes attached to a class are transformed into datatype properties in OWL and unnamed property restrictions for the class).

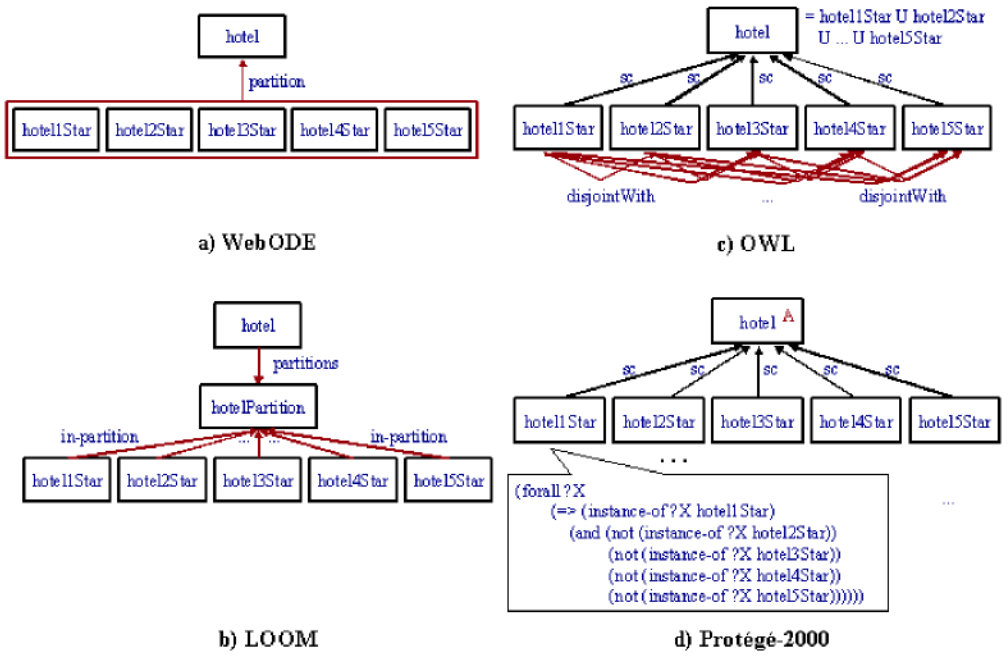
As an example, Figure 3 shows how to represent a concept partition in different ontology languages and tools. In WebODE and LOOM, there are specific built-in primitives for representing partitions. In OWL the partition must be represented by defining the *rdfs:subClassOf* relationship between each class in the partition and the parent class, by stating that every possible pair of classes in the decomposition is disjoint, and by defining the parent class as the union of all the classes in the partition. In Protégé-2000, the partition is represented like in OWL, with *subclass-of* relationships between all the classes in the partition and the parent class, with several PAL constraints that represent disjointness between all the classes in the partition, and with the statement that the parent class is abstract (that is, it cannot have direct instances).

Most of the work on ontology translation done so far has been devoted to solving the problems that arise in this layer. For example, in the literature, we can find several formal, semi-formal, and informal methods for comparing ontology languages and ontology tools' knowledge models (Baader, 1996; Borgida, 1996; Corcho & Gómez-Pérez, 2000; Euzenat & Stuckenschmidt, 2003; Knublauch, 2003), which aim at helping to decide whether two formats have the same expressiveness or not, so that knowledge can be preserved in the transformation. Some of these approaches also can be used to decide whether the reasoning mechanisms present in both formats will allow inferring the same knowledge in the target format.

Basically, these studies allow analyzing the expressiveness (and, in some cases, the reasoning mechanisms) of the source and target formats, so that we can know which types of components can be translated directly from a format to another, which types of components can be expressed using other types of components from the target format, which types of components cannot be expressed in the target format, and which types of components can be expressed, although losing part of the knowledge represented in the source format.

Therefore, the catalogue of problems that can be found in this layer are related mainly to the different KR formalisms in which the source and target formats are based. This does not mean that translating between two formats based on the same KR formalism is straightforward, since there might be differences in the types of ontology components that can be represented in each of them. This is specially important in the case of DL languages, since many different combinations of primitives can be used in each language, and, hence, many possibilities exist in the transformations between them, as shown in Euzenat and Stuckenschmidt (2003).

Figure 3. Examples of transformations at the semantic layer



However, the most interesting results appear when the source and target KR formalisms are different.

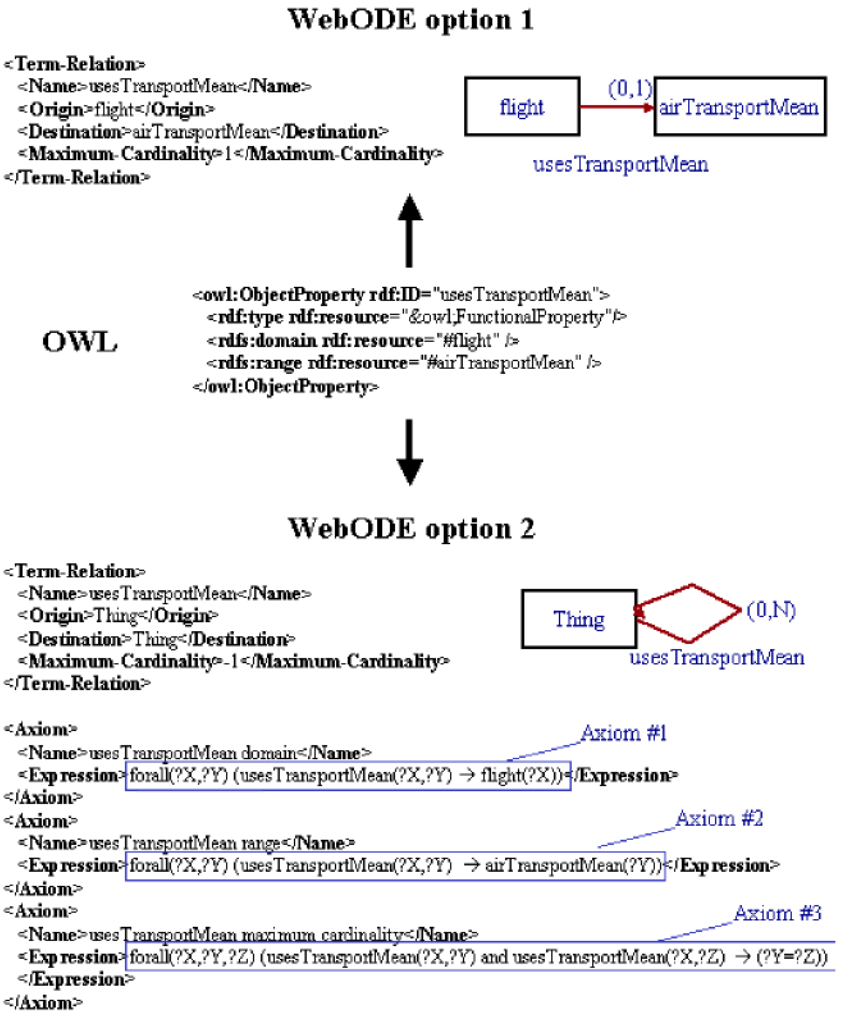
Pragmatic Layer

This layer deals with the ability to construct the pragmatic meaning of the representation (or its meaning in context). Therefore, in this layer we deal with the transformations to be made in the ontology resulting from the lexical, syntactic, and semantic transformations, so that both human users and ontology-based applications will notice as few differences as possible with respect to the ontology in the original format, either in one-direction transformations or in cyclic transformations.

Therefore, transformations in this layer will require the following: adding special labels to ontology components in order to preserve their original identifier in the source format; transforming sets of expressions into more legible syntactic constructs in the target format; hiding completely or partially some ontology components not defined in the source ontology but that have been created as part of the transformations (such as the anonymous classes discussed previously); and so forth.

Figure 4 shows two transformations of the OWL functional object property usesTransportMean to WebODE. The object property domain is the class flight, and its range is the class airTransportMean.

Figure 4. Examples of transformations at the pragmatic layer



The figure shows two of the possible semantically equivalent sets of expressions that can be obtained when transforming that definition. In the first one, the object property is transformed into the ad hoc relation usesTransportMean that holds between the concepts flight and airTransportMean, with its maximum cardinality set to one. In the second one, the object property is transformed into the ad hoc relation usesTransportMean, whose domain and range is the concept Thing (the root of the ontology concept taxonomy), with no restrictions on its cardinality, plus three formal axioms expressed in first-order logic, the first one stating that the relation domain is flight, the second one that its range is airTransportMean, and the third one imposing the maximum cardinality constraint[5].

From a human user's point of view, the first WebODE definition is more legible; at first glance, the user can see that the relation usesTransportMean is defined between the concepts flight and airTransportMean, and that its maximum cardinality is one. In the second case, the user must find and interpret the four components (the ad hoc relation definition and the three formal axioms) to reach the same conclusion.

A similar conclusion can be obtained from an application point of view. Let us suppose that we want to populate the ontology with an annotation tool. The behavior of the annotation tool is different for both definitions. With the first definition, the annotation tool will easily understand that its user interface cannot give users the possibility of adding more than one instance of the relation, and that the drop-down lists used for selecting the domain and range of a relation instance will show only direct or indirect instances of the concepts flight and airTransportMean, respectively. With the second definition, the annotation tool will allow creating more than one relation instance from the same instance and will display all the ontology instances in the drop-down lists instead of just presenting instances of flight and airTransportMean, respectively. After that, the annotation tool will have to run the consistency checker to detect inconsistencies in the ontology.

Relationships Between Ontology Translation Layers

Figure 5 shows an example of a transformation from the ontology platform WebODE to the language OWL DL. In this example, we have to transform two ad hoc relations with the same name (*usesTransportMean*) and with different domains and ranges (a *flight* uses an *airTransportMean*, and a *cityBus* uses a *bus*). In OWL DL, the scope of an object property is global to the ontology, and thus we cannot define two different object properties with the same name. In this example, we show that translation decisions have to be taken at all layers, and we also show how the decision taken at one layer can influence the decisions to be made at the others, hence showing the complexity of this task.

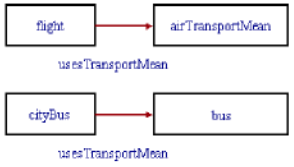
Option 1 is driven by semantics; to preserve semantics in the transformation, two different object properties with different identifiers are defined. Option 2 is driven by pragmatics; only one object property is defined from both ad hoc relations, since we assume that they refer to the same meaning, but some knowledge is lost in the transformation (the one related to the object property domain and range). Finally, Option 3 also is driven by pragmatics, with more care on the semantics; again, only one object property is defined, and its domain and range is more restricted than in Option 2, although we still lose the exact correspondence between each domain and range.

A LAYERED ONTOLOGY TRANSLATION METHOD

Once we have described the four layers where ontology translation decisions have to be made, we will present our

WebODE

```
<Term-Relation>
  <Name>usesTransportMean</Name>
  <Origin>flight</Origin>
  <Destination>airTransportMean</Destination>
</Term-Relation>
<Term-Relation>
  <Name>usesTransportMean</Name>
  <Origin>cityBus</Origin>
  <Destination>bus</Destination>
</Term-Relation>
```



OWL (1)

```
<owl:ObjectProperty rdf:ID="flight_usesTransportMean">
  <rdfs:domain rdf:resource="#flight"/>
  <rdfs:range rdf:resource="#airTransportMean"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="cityBus_usesTransportMean">
  <rdfs:domain rdf:resource="#cityBus"/>
  <rdfs:range rdf:resource="#bus"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="flight">
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty
          rdf:resource="#flight_usesTransportMean" />
     <owl:allValuesFrom
          rdf:resource="#airTransportMean" />
   </owl:Restriction>
 </rdfs:subClassOf>
   ...
</owl:Class>
<owl:Class rdf:ID="cityBus">
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty
          rdf:resource="#cityBus_usesTransportMean" />
     <owl:allValuesFrom rdf:resource="#bus" />
   </owl:Restriction>
 </rdfs:subClassOf>
   ...
</owl:Class>
```

OWL (2)

```
<owl:ObjectProperty rdf:ID="usesTransportMean"/>

<owl:Class rdf:ID="flight">
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#usesTransportMean" />
     <owl:allValuesFrom rdf:resource="#airTransportMean" />
   </owl:Restriction>
 </rdfs:subClassOf>
   ...
</owl:Class>

<owl:Class rdf:ID="cityBus">
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#usesTransportMean" />
     <owl:allValuesFrom rdf:resource="#bus" />
   </owl:Restriction>
 </rdfs:subClassOf>
   ...
</owl:Class>
```

OWL (3)

```
<owl:ObjectProperty rdf:ID="usesTransportMean">
 <rdfs:domain>
   <owl:Class>
     <owl:unionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#flight"/>
       <owl:Class rdf:about="#cityBus"/>
     </owl:unionOf>
   </owl:Class>
 </rdfs:domain>
 <rdfs:range>
   <owl:Class>
     <owl:unionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#airTransportMean"/>
       <owl:Class rdf:about="#bus"/>
     </owl:unionOf>
   </owl:Class>
 </rdfs:range>
</owl:ObjectProperty>
```

```
<owl:Class rdf:ID="flight">
 <rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty
          rdf:resource="#usesTransportMean" />
    <owl:allValuesFrom
          rdf:resource="#airTransportMean" />
  </owl:Restriction>
 </rdfs:subClassOf>
   ...
</owl:Class>

<owl:Class rdf:ID="cityBus">
 <rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty
          rdf:resource="#usesTransportMean" />
    <owl:allValuesFrom rdf:resource="#bus" />
  </owl:Restriction>
 </rdfs:subClassOf>
   ...
</owl:Class>
```

Different identifiers for each object property	The same identifiers for both object properties	The same identifiers for both object properties	***Lexical layer***
RDF/XML Abbrev	RDF/XML Abbrev	RDF/XML Abbrev	***Syntactic layer***
No losses of expressiveness	Some expressiveness lost: object property can be applied to any class	Some expressiveness lost: the exact correspondance between domain and range is lost	***Semantic layer***
Both properties are interpreted as different things	Both properties are interpreted as the same. By reading the object property definition, it is not easy to know where it is applied	Both properties are interpreted as the same. By reading the object property definition, it is easier to know where it is applied	***Pragmatic layer***

Figure 5. Example of translation decisions to be taken at several layers

method for building ontology translation systems, based on these layers. This method consists of four activities: feasibility study, analysis of source and target formats, design and implementation of the translation system. As we will describe later, these activities are divided into tasks, which can be performed by different sets of people and with different techniques.

Ontology translation systems are difficult to create, since many different types of problems have to be dealt with. Consequently, this method recommends developing ontology translation systems following an iterative life cycle. It proposes identifying a first set of expressions that can be translated easily from one format to another, so that the first version of the ontology translation system can be developed and tested quickly; then, it proposes refining the transformations performed to analyze more complex expressions and to design and implement their transformations, and so forth. The reason for such a recommendation is that developing an ontology translation system is usually a complex task that requires taking into account too many aspects of the source and target formats, and many different types of decisions on how to perform specific translations. In this sense, an iterative life cycle ensures that complex translation problems are tackled once the developers have a better knowledge of the source and target formats and once they have tested simpler translations performed with earlier versions of the software produced.

The feasibility activity is performed at the beginning of the development project. If this study recommends starting with the ontology translation system development, then for each cycle, the other three activities will be performed sequentially, although developers always can go back to a previous activity using the feedback provided by the subsequent ones, as shown in Figure 6, which summarizes the proposed development process.

As a summary, Table 2 lists the activities that the method proposes and the tasks to be performed inside each activity. The design and implementation activities take into account the four translation layers described in the previous section.

The method does not put special emphasis on other activities that usually are related to software system development, either specific to the software development process, such as deployment and maintenance, or related to support activities, such as quality assurance, project management, and configuration management. Nor does it emphasize other tasks usually performed during the feasibility study, analysis, design, and implementation activities of general software system development. It only describes those tasks that are specifically related to the development of ontology translation systems and recommends performing such additional activities and tasks that will be beneficial to their development.

In the following sections, we will describe briefly the objective of each of these activities, the techniques that can be used to perform them, and their inputs and outputs.

Feasibility Study

The objective of this activity is to analyze the ontology translation needs, so

Figure 6. Proposed development process of ontology translation system

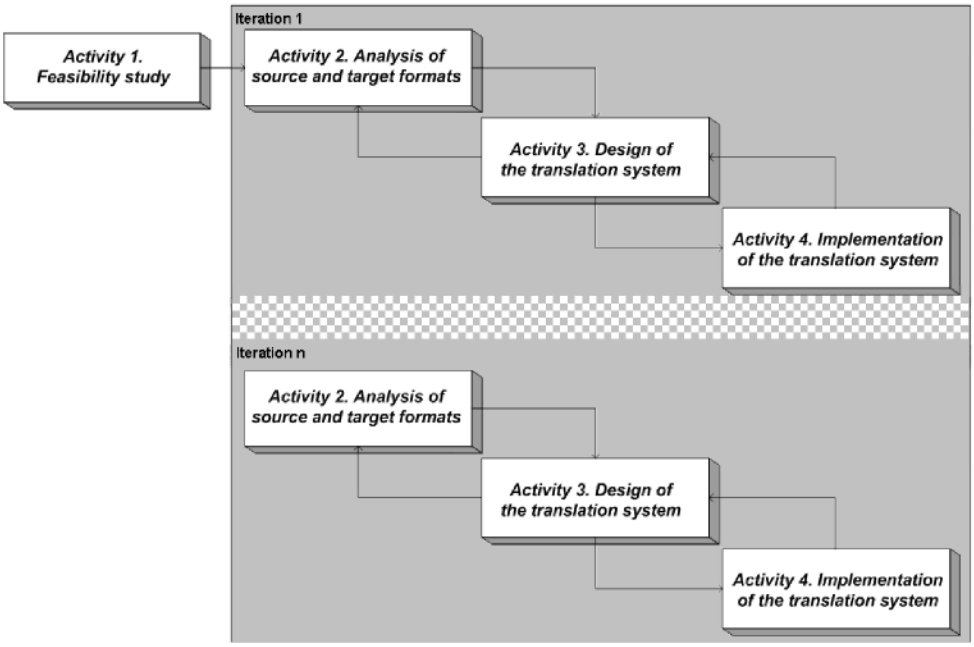
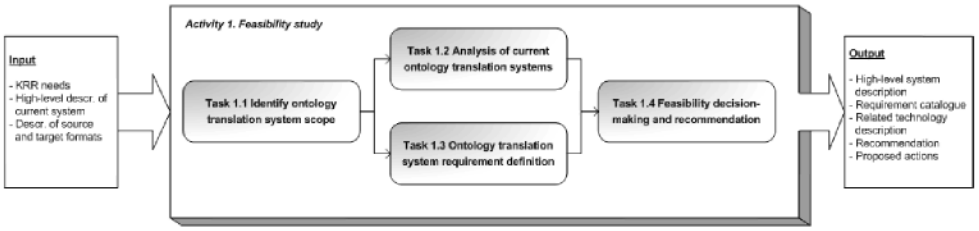


Table 2. List of activities and tasks of the method for developing ontology translation systems

Activity	*Task*
1. Feasibility study	1.1. Identify ontology translation system scope 1.2. Analysis of current ontology translation systems 1.3. Ontology translation system requirement definition 1.4. Feasibility decision-making and recommendation
2. Analysis of source and target formats	2.1. Describe source and target formats 2.2. Determine expressiveness of source and target formats 2.3. Compare knowledge models of source and target formats 2.4. Describe and compare additional features of source and target formats 2.5. Determine the scope of translation decisions 2.6. Specify test plan
3. Design of the translation system	3.1. Find and reuse similar translation systems 3.2. Propose transformations at the pragmatic level 3.3. Propose transformations at the semantic level 3.4. Propose transformations at the syntax level 3.5. Propose transformations at the lexical level 3.6. Propose additional transformations
4. Implementation of the translation system	4.1. Find translation functions to be reused 4.2. Implement transformations in the pragmatic level 4.3. Implement transformations in the semantic level 4.4. Implement transformations in the syntax level 4.5. Implement transformations in the lexical level 4.6. Implement additional transformations 4.7. Declarative specification processing and integration 4.8. Test suite execution

Figure 7. Task decomposition of activity 1 (feasibility study)



that the proposed solution takes into account not only the technical restrictions (technical feasibility), but also other restrictions related to the business objectives of an organization (business feasibility) and to the project actions that can be undertaken successfully (project feasibility). As a result of this activity, the main requisites to be satisfied by the ontology translation system are obtained, and the main costs, benefits, and risks are identified. The most important aspect of this feasibility study regards the technical restrictions, which can determine whether it is recommended or not to proceed with the ontology translation system development.

The techniques (and documents) used in the execution of these tasks are inspired by knowledge engineering approaches (Gómez-Pérez et al., 1997; Schreiber et al., 1999) and based mainly on the CommonKADS worksheets.

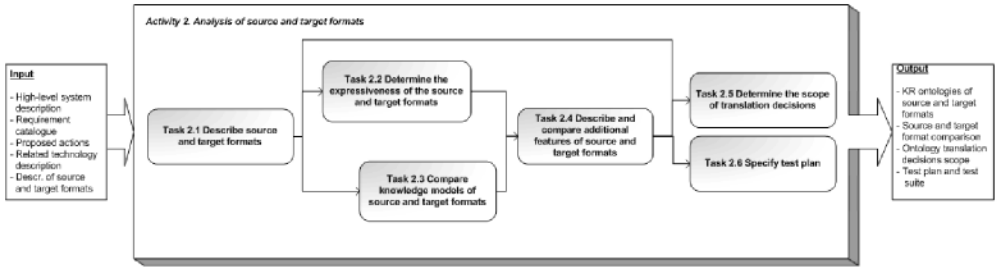
As shown in Figure 7, we first propose to determine the scope of the ontology translation system that will be implemented, its expected outcome, the context where it will be used, and so forth. We then propose to analyze current translation systems that are available between the source and target formats and determine the requisites of the new system. Finally, we propose to fill in a checklist where the three dimensions identified are considered (technical, business, and project feasibility), allowing us to make a decision on the feasibility of the system and to propose a set of actions and recommendations to be followed.

Consequently, the input in this activity consists of some preliminary high-level information about current systems, the KRR needs, and the source and target formats. The results consist in a deeper description of the current ontology translation systems available for the origin and target formats, a preliminary catalogue of requisites for the system to be developed, and the recommendation about its feasibility, including the main costs, benefits, and risks involved.

Analysis of the Source and Target Formats

The objective of this activity is to obtain a thorough description and comparison of the source and target formats of the ontology translation system. We assume that this will allow us to gain a better understanding of the similarities and differences in expressiveness, which will be useful in designing and implementing

Figure 8. Task decomposition of activity 2 (analysis of source and target formats)



the translation decisions in the subsequent activities. Moreover, in this activity we refine the catalogue of requirements already obtained as a result of the feasibility study, and we identify the test suite that will be used to test the translation system validity after each iteration in its development process. A summary of the tasks to be performed and the input and outputs of this activity is shown in Figure 8.

Many techniques can be used to describe the source and target formats of the translation system. Among them, the method recommends describing their KR ontologies (as shown in Broekstra et al., 2000 or Gómez-Pérez et al., 2003), which provide a good overview of the ontology components that can be used to represent ontologies with them.

For the comparison tasks, we can use either formal, semi-formal, or informal approaches, such as the ones identified in Section 2.3, which show good examples of the results that should be obtained. Once the two formats have been described, evaluated, and compared, we recommend focusing on other additional features that might be needed in the translation process. They may include reasoning mechanisms or any other specific details that could be interesting for the task of translation.

The information gathered in the previous tasks is used to determine the scope of the translation decisions to be made; that is, which components map to each other, which components of the source format must be represented by means of others in the target format, which components cannot be represented in the target format, and so forth. As a result, we obtain a refinement of the requirement catalogue obtained during the feasibility study, which serves as the basis for the next activities (design and implementation of the translation system).

Finally, we propose to define the test plan, which consists of a set of unitary tests that the translation system must pass in order to be considered valid. The test suite must consider all of the possible translation situations that the translation system must cover. These ontologies will be available in the source format and in the target format, which should be the output of the translation process. The test execution will consist of comparing the output obtained and the output expected. For each iteration of the software development process, we will define different sets of ontologies.

This activity receives as an input all the results of the feasibility study, together with the description of the source and target formats (also used as an input for that activity). It outputs a comparison of both formats; the scope of the translation decisions to be performed, with a refined requirements catalogue; and a test plan with its corresponding test suite.

Design of the Translation System

The design activity aims at providing a detailed specification of the transformations to be performed by the ontology translation system. From this specification, we will be able to generate the implementation of the translation decisions at each layer, which will be used in its turn to generate the final ontology translation system. The tasks, inputs, and outputs of this activity are shown in Figure 9.

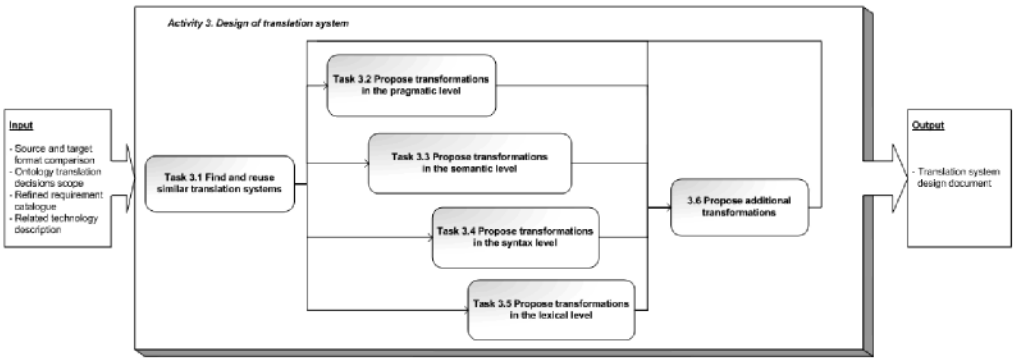
The objective of the first task is to analyze similar ontology translation systems and to detect which of their translation decisions actually can be reused. We assume that by reusing existing translation decisions, we will be able to minimize the sources of errors in our translation proposals. Furthermore, we will benefit from work already known, for which we already know its properties (namely, how they preserve semantics and pragmatics). We must remember that the potential reusable systems were already identified and catalogued during the feasibility study.

The second group of tasks deals with the four layers of translation problems described in Section 2. We propose to design transformations at different interrelated levels, using different techniques for each layer. All these tasks should be performed mainly in parallel, and the decisions taken at one task provide feedback for the others, as shown in the figure. We propose to start with the translation decisions at the pragmatic and semantic levels, leaving the syntax and lexical transformations for the last steps. The pragmatic and semantic translation decisions are proposed mainly by knowledge engineers, while the syntax and lexical transformations can be proposed jointly by knowledge and software engineers, since they have more to do with general programming aspects rather than with the complexity of transforming knowledge. The method proposes to represent these translation decisions mainly with tables and diagrams, such as the ones proposed in Table 3 and Figure 10 for transformations between WebODE and OWL DL.

Finally, the objective of the last task is to propose any additional transformations or design issues that have not been covered by the previous tasks, because they could not be catalogued as lexical, syntax, semantic, or pragmatic transformations, which are necessary for the correct functioning of the ontology translation system. These transformations include design issues such as the initialization and setting up of parameters in the source and target formats, any foreseen integration needs of the generated system in the case of transformations where ontology tools or specific libraries are used, and so forth.

As shown in Figure 9, we may need to come back to the second group of activities after proposing some additional transformations. This is a cyclic process

Figure 9. Task decomposition of activity 3 (design of the ontology translation system)



until we have determined all the transformation to be performed in the corresponding development iteration. All the output results obtained from the tasks in this activity are integrated in a single document called "translation system design document," as shown in the figure.

Implementation of the Translation System

The objective of the implementation activity is to create the declarative specifications of the transformations to be performed by the ontology translation system, which will be used to generate its final code. The method proposes to implement these translations using three different formal languages—ODELex, ODESyntax, and ODESem—which correspond to the lexical, syntax, and semantic/pragmatic ontology translation layers, respectively. The same language (ODESem) is used for implementing semantic and pragmatic transformations, because the translation decisions at both layers are similar. The description of these languages is out of the scope of this chapter and can be found in Corcho and Gómez-Pérez (2004) and Corcho (2005). We can say that the ODELex and ODESyntax languages are similar to the lex (Lesk, 1975) and yacc (Johnson, 1975) languages used for compiler construction, and that ODESem is based on common rule-based systems.

As in the design activity, the tasks inside this implementation activity are divided in groups—four, in this case—as shown in Figure 11.

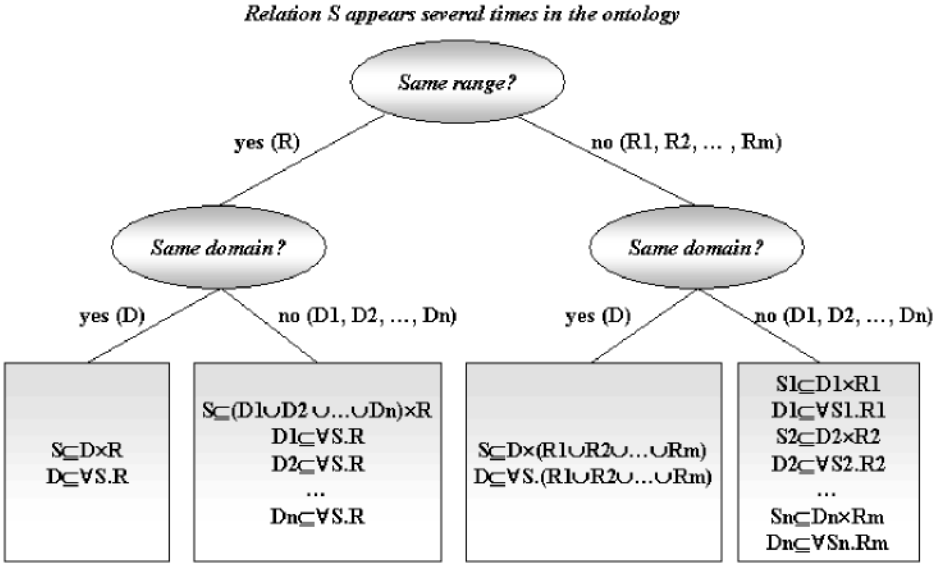
The goal of the first task is to select reusable pieces of code from the declarative specifications of other ontology translation systems. These pieces of code are selected on the basis of the results obtained from the first task of the design activity and can be related to any of the four translation layers.

The next five tasks are grouped together and should be performed almost in parallel, as shown in the figure. In these tasks, software and knowledge engineers

Table 3. Semantic transformation of WebODE partitions to OWL DL

WebODE	OWL DL
Partition (C, {C1,C2,…,Cn})	$C \equiv C1 \cup C2 \cup \ldots \cup Cn$ $Ci \subseteq C \quad \forall Ci \in \{C1,C2,\ldots,Cn\}$ $Ci \cap Cj \subseteq \perp \quad \forall Ci \neq Cj,, Ci,Cj \in \{C1,C2,\ldots,Cn\}$

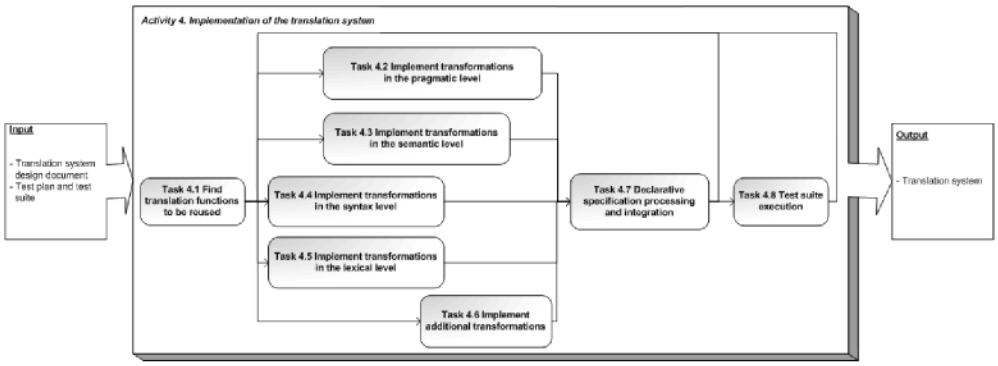
Figure 10. Pragmatic transformations with regard to the scope of WebODE ad hoc relations



actually must implement the transformations at the four layers—lexical, syntax, semantic, and pragmatic—and the additional transformations described in task 3.6. Unlike in the design activity, we propose to start with the low-level transformations (those at the lexical and syntax layers) and continue with the more abstract (and difficult) ones. The reason for the task ordering suggested is that the semantic and pragmatic transformation implementations usually need to take into account the specific implementations at the lexical and syntax layers. We are currently developing automatic tools that transform the declarative specifications in ODELex, ODESyntax, and ODESem into Java code.

In task 4.7—declarative specification processing and integration—the software engineer is in charge of transforming the previous declarative implementations at all levels, plus the additional transformations, into actual running code, which will perform the translations as specified in the previous code. In addition, the software engineer has to integrate the resulting ontology translation system into another information system (e.g., an ontology tool), if required. Given that most of the transformations have been implemented in formal languages, most of the

Figure 11. Task decomposition of activity 4 (implementation of the ontology translation system)



processes involved in this task can be automated. If problems are detected during this task, the method recommends going back to the implementation activities in order to solve them.

Finally, the method proposes to execute the test suite that was defined during the analysis activity, which is considered the system tests for our system. This does not prevent us from defining and executing other kinds of tests (from unitary tests to integration tests) at any point during the development. This task consists of inputting the ontologies in the test suite to the resulting ontology translation system and checking whether the output corresponds to the one expected. Note that in most cases, this check will consist of comparing whether the output file(s) and the expected file(s) are identical, but there are cases where this kind of comparison will not be possible, since the results can come in any order (e.g., in RDF and OWL ontologies). If any of the test fails, we must go back to the previous implementation activities to detect the problems. Furthermore, we must consider that the method allows moving to previous activities if problems are detected at any point of our development.

CONCLUSION

This chapter presents two important contributions to the current state of the art on ontology translation. First, it proposes to consider that ontology translation problems can appear at four different layers, which are interrelated, and can describe the most common problems that may appear at each of those layers. Some existing approaches have identified similar layers in ontology translation. However, these and other approaches have focused mainly on the problems related to the semantic layer and have not considered the other ones, which are also important for building systems that make good quality translations. The low quality of some translation systems has been shown recently in the interoperability experiment performed for the ISWC2003 workshop on Evalua-

tion of Ontology Tools[6]. The results obtained in this workshop showed that making good translation decisions at the lexical, syntax, and pragmatic levels is also as important as making good translation decisions at the semantic level.

The second main contribution of this chapter is related to the fact that it is the first approach that gives an integrated support for the complex task of building ontology translation systems. As we commented in the introduction, ontology translation systems are not easy to create and are difficult to maintain, as well. Most of the translation systems currently available have been developed ad hoc; the translation decisions that they implement are usually difficult to understand and hidden in the source code of the systems; and, in addition, it is neither clear nor documented how much knowledge is lost in the transformations that they perform. There are many complex decisions that have to be implemented, and these decisions are usually taken at the low implementation level instead of performing a detailed analysis and design of the different translation choices available and taking a decision based on the actual ontology translation requirements. The method proposed in this chapter helps in this task by identifying clearly the activities to be performed, the tasks in which each activity is decomposed, how these tasks have to be performed, the inputs and outputs of the activities, and the set of techniques that can be used to perform them. Moreover, a set of declarative languages is proposed, although not described in this chapter, to help in the implementation of translation decisions.

This method has been derived from our long experience in the generation of ontology translation systems from the ontology engineering platform WebODE to different ontology languages and tools, and vice versa (12 systems), and has been used for building other six ontology translation systems. These systems have been built successfully by different people with backgrounds in knowledge and software engineering, following the method proposed in this chapter and the techniques identified for each task.

RELATED WORK

Although there are no other integrated methods for building ontology translation systems available, we can find some technology that allows creating them. Specifically, we can cite two tools: Transmorpher and OntoMorph:

- *Transmorpher*[7] (Euzenat & Tardif, 2001) is a tool that facilitates the definition and processing of complex transformations of XML documents. Among other domains, this tool has been used in the context of ontologies, using a set of XSLT documents that is able to transform from one DL language to another, expressed in DLML[8]. This tool is aimed at supporting the "family of ontology languages" approach for ontology translation described in Euzenat and Stuckenschmidt (2003). The main limitation of this approach is that it only deals with problems in the semantic layer and does not focus on other problems related to the lexical, syntax, and pragmatic layers.

- *OntoMorph* (Chalupsky, 2000) is a tool that allows creating translators declaratively. Transformations between the source and target formats are specified by means of pattern-based transformation rules and are performed in two phases: syntactic rewriting and semantic rewriting. The last one needs the ontology or part of it translated into PowerLoom, so that this KR system can be used for certain kinds of reasoning, such as discovering whether a class is a subclass of another, whether a relation can be applied to a concept or not, and so forth. Since this tool is based on PowerLoom (and consequently on Lisp), it cannot handle easily all the problems that may appear in the lexical and syntax layers.

Although these tools do not give an integrated support for the task of building ontology translation systems, this does not mean that they cannot be used as a technological support for the method proposed in this chapter, especially for the implementation activity.

ACKNOWLEDGMENTS

This work has been supported by the IST project Esperonto (IST-2001-34373). Part of this chapter is based on Section 3.4 of the book *A Layered Declarative Approach to Ontology Translation with Knowledge Preservation*, (Corcho, 2005).

REFERENCES

Arpírez, J.C., Corcho, O., Fernández-López, M., & Gómez-Pérez, A. (2003). WebODE in a nutshell. *AI Magazine, 24*(3),37-48.

Baader, F. (1996). A formal definition for the expressive power of terminological knowledge representation languages. *Journal of Logic and Computation, 6*(1), 33-54.

Baader, F., McGuinness, D., Nardi, D., & Patel-Schneider, P. (2003). *The description logic handbook: Theory, implementation and applications*. Cambridge, UK: Cambridge University Press.

Bechhofer, S., Horrocks, I., Goble, C., & Stevens, R. (2001). *OilEd:* A reasonable ontology editor for the Semantic Web. *Proceedings of the Joint German/Austrian conference on Artificial Intelligence (KI'01)*, Vienna, Austria.

Borgida, A. (1996). On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence, 82*(1-2), 353-367.

Brickley, D., & Guha, R.V. (2004). RDF vocabulary description language 1.0: RDF Schema. *W3C*. Retrieved from *http://www.w3.org/TR/PR-rdf-schema*

Chalupsky, H. (2000). OntoMorph: A translation system for symbolic knowledge. *Proceedings of the 7th International Conference on Knowledge Representation and Reasoning (KR'00)*. Breckenridge, Colorado.

Corcho, O. (2005). *A layered declarative approach to ontology translation*

with knowledge preservation. Frontiers in Artificial Intelligence and its Applications. Dissertations in Artificial Intelligence. IOS Press

Corcho, O., & Gómez-Pérez, A. (2000). *A roadmap to ontology specification languages. Proceedings of the 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00)*, Berlin, Germany.

Corcho, O., & Gómez-Pérez, A. (2004). *ODEDialect: A set of declarative languages for implementing ontology translation systems. Proceedings of the ECAI2004 Workshop on Semantic Intelligent Middleware for the Web and the Grid*, Valencia, Spain.

Dean, M., & Schreiber, G. (2004). OWL Web ontology language reference. *W3C*. Retrieved from *http://www.w3.org/TR/owl-ref/*

Domingue, J. (1998). Tadzebao and WebOnto: Discussing, browsing, and editing ontologies on the Web. *Proceedings of the 11th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*. Banff, Canada.

Euzenat, J. (2001). Towards a principled approach to semantic interoperability. *Proceedings of the IJCAI 2001 Workshop on Ontologies and Information Sharing*, Seattle, Washington.

Euzenat, J., & Stuckenschmidt, H. (2003). *The "family of languages" approach to semantic interoperability*. In B. Omelayenko, & M. Klein (Eds.), *Knowledge transformation for the semantic Web* (pp. 49-63). Amsterdam, The Netherlands: IOS Press.

Euzenat, J., & Tardif, L. (2001). XML transformation flow processing. *Markup Languages: Theory and Practice, 3*(3), 285-311.

Farquhar, A., Fikes, R., & Rice, J. (1997). The ontolingua server: A tool for collaborative ontology construction. *International Journal of Human Computer Studies, 46*(6), 707-727.

Genesereth, M.R., & Fikes, R.E. (1992). Knowledge Interchange Format. Version 3.0. Reference Manual. Technical Report Logic-92-1. Retrieved from *http://meta2.stanford.edu/kif/Hypertext/kif-manual.html*

Gómez-Pérez, A., Fernández-López, M., & Corcho, O. (2003). *Ontological engineering: With examples from the areas of knowledge management, e-commerce and the Semantic Web*, New York: Springer-Verlag.

Gómez-Pérez, A., Juristo, N., Montes, C., & Pazos, J. (1997). *Ingeniería del conocimiento*. Centro de Estudios Ramón Areces

Gruber, T.R. (1992). Ontolingua: A mechanism to support portable ontologies. Technical report KSL-91-66. Retrieved from *ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-91-66.ps*

Gruber, T.R. (1993). A translation approach to portable ontology specification. *Knowledge Acquisition, 5*(2), 199-220.

Horrocks, I., Fensel, D., Harmelen, F., Decker, S., Erdmann, M., & Klein, M. (2000). OIL in a Nutshell. *Proceedings of the 12th International Con-*

ference in Knowledge Engineering and Knowledge Management (EKAW'00). Juan-Les-Pins, France.

Horrocks, I., & van Harmelen, F. (Eds.) (2001). *Reference description of the DAML+OIL (March 2001) ontology markup language* [technical report]. Retrieved from *http://www.daml.org/2001/03/reference.html*

Johnson, S.C. (1975). *Yacc: Yet another compiler compiler*. *Computing science technical report no. 32*. Murray Hill, NJ: Bell Laboratories.

Karp, P.D., Chaudhri, V., & Thomere, J. (1999). XOL: An XML-based ontology exchange language. Retrieved from *http://www.ai.sri.com/~pkarp/xol/xol.html*

Kifer, M., Lausen, G., & Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *Journal of the ACM, 42*(4), 741-843.

Klein, M. (2001). Combining and relating ontologies: An analysis of problems and solutions. *Proceedings of the Workshop on Ontologies and Information Sharing*, Seattle, Washington.

Knublauch, H. (2003). Editing semantic Web content with Protégé: The OWL pPlugin. *Proceedings of the 6th Protégé Workshop*. Manchester, UK.

Lassila, O., & Swick, R. (1999). Resource description framework (RDF) model and syntax specification. *W3C*. Retrieved from *http://www.w3.org/TR/REC-rdf-syntax/*

Lenat, D.B., & Guha, R.V. (1990). *Building large knowledge-based systems: Representation and inference in the cyc project*. Boston: Addison-Wesley.

Lesk, M.E. (1975). *Lex—A lexical analyzer generator*. *Computing Science Technical Report No. 39*. Murray Hill, NJ: Bell Laboratories.

Luke, S., & Heflin, J. (2000). *SHOE 1.01*. Proposed specification. Technical Report. Parallel Understanding Systems Group. Retrieved from *http://www.cs.umd.edu/projects/plus/SHOE/spec1.01.htm*

MacGregor, R. (2001). Inside the LOOM classifier. *SIGART Bulletin, 2*(3), 88-92.

Maedche, A., Motik, B., Stojanovic, L., Studer, R., & Volz, R. (2003). Ontologies for enterprise knowledge management. *IEEE Intelligent Systems, 18*(2), 26-33.

Morris, C.W. (1938). Foundations of the theory of signs. In O. Neurath, R. Carnap, C.W. Morris (Eds.), *International Encyclopedia of Unified Science*. Chicago, IL: Chicago University Press.

Motta, E. (1999). *Reusable components for knowledge modelling: Principles and case studies in parametric design*. Amsterdam, The Netherlands: IOS Press.

Noy, N.F., Fergerson, R.W., & Musen, M.A. (2000). The knowledge model of Protege-2000: Combining interoperability and flexibility. *Proceedings of the 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00)*, Juan-Les-Pins, France.

Schreiber, G., et al. (1999). *Knowledge engineering and management. The commonKADS methodology*. Cambridge, MA: MIT Press.

Studer, R., Benjamins, V.R., & Fensel, D. (1998). Knowledge engineering: Principles and methods. *IEEE Transactions on Data and Knowledge Engineering, 25*(1-2), 161-197.

Sure, Y., Staab, S., & Angele, J. (2002). OntoEdit: Guiding ontology development by methodology and inferencing. *Proceedings of the Confederated International Conferences CoopIS, DOA and ODBASE 2002*, Berlin, Germany.

Swartout, B., Ramesh, P., Knight, K., & Russ, T. (1997). Toward distributed use of large-scale ontologies. *Proceedings of the Spring Symposium on Ontological Engineering*, Stanford, California.

ENDNOTES

* The current affiliation of the author is Intelligent Software Components, Spain. The work presented was performed at Universidad Politécnica de Madrid.

1 The problems that may appear in the context of semantic interoperability are due not only to the fact that ontologies are available in different formats, but they are also related to the content of ontologies, their ontological commitments, and so forth. We only focus on the problems related exclusively to the differences between ontology languages and/or tools.

2 These types of problems also may be related to the pragmatic layer, as we will describe later in this section. We also will see that the limits of each translation layer are not strict; hence, we can find transformation problems that are in the middle of several layers.

3 As with naming conventions, this decision also will be related to the pragmatic translation layer.

4 Protégé Axiom Language

5 We must note that this second option may be obtained because expressions in OWL ontologies may appear in any order in an OWL file and, hence, may be processed independently.

6 http://km.aifb.uni-karlsruhe.de/ws/eon2003/

7 http://transmorpher.inrialpes.fr/

8 Description Logic Markup Language. http://co4.inrialpes.fr/xml/dlml/

Dr. Oscar Corcho has worked at iSOCO as a research manager since March 2004. Previously, he belonged to the Ontological Engineering Group of the AI Department of the Computer Science School, Universidad Politécnica de Madrid (UPM). He graduated in computer science from UPM in 2000 and received the third Spanish award in computer science from the Spanish Government. He obtained an MSc in software engineering from UPM (2001) and a PhD in artificial intelligence (2004). His research activities include ontology languages and tools, the ontology translation problem and the Semantic Web and grid. He has published the books Ontological Engineering *and* A Layered Declarative Approach to Ontology Translation with Knowledge Preservation. *He has published more than 30 journal and conference/workshop papers on ontology languages and tools in important forums for the*

ontology community (ISWC, EKAW, KAW, KCAP, AI Magazine, *IEEE Intelligent Systems), and reviews papers in many conferences, workshops and journals. He chaired the demo/industrial sessions at EKAW2002 and co-organised the ISWC2003 and ISWC2004 Workshops on Evaluation of Ontology Tools (EON2003, EON2004).*

Dr. Asunción Gómez-Pérez is the director of the Ontological Engineering Group at UPM. She earned a BA in computer science (1990), an MSc in knowledge engineering (1991), and a PhD in computer science (1993) from the Universidad Politécnica de Madrid (UPM). She also has an MBA (1994) from the Universidad Pontificia de Comillas. From 1994 to 1995, she visited the Knowledge Systems Laboratory at Stanford University. She is associate professor at the Computer Science School at UPM. From 1995 to 1998 she was executive director of the Artificial Intelligence Laboratory at the school. She is currently a research advisor of the same lab. Her current research activities include, among others: interoperability between different kinds of ontology development tools; methodologies and tools for building and merging ontologies; ontological reengineering; ontology evaluation; and ontology evolution, as well as uses of ontologies in applications related with Semantic Web, e-commerce and knowledge management. She has published more than 100 papers on the above issues. She has led several national and international projects related to ontologies funded by various institutions and/or companies. She is the author of one book on ontological engineering and is co-author of a book on knowledge engineering. She was the co-director and local organiser of the first and second European summer schools on ontologies and the Semantic Web, and chair of the 13th International Conference on Knowledge Acquisition and Management (EKAW-02). She has been co-organizer of the workshops and conferences on ontologies at ECAI-04, IJCAI-03, ECAI-02, IJCAI-01, ECAI-00, IJCAI-99, ECAI-98, SSS-97 and ECAI-96. She has taught tutorials on ontological engineering at ECAI-04, ECAI-98, SEKE-97 and CAEPIA-97. She acts as reviewer in many conferences and journals.