WebODE: an integrated workbench for ontology representation, reasoning and exchange

Óscar Corcho, Mariano Fernández-López, Asunción Gómez-Pérez, Óscar Vicente

Facultad de Informática . Universidad Politécnica de Madrid
Campus de Montegancedo, s/n. 28660 Boadilla del Monte. Madrid. Spain
{ocorcho, mfernandez, asun}@fi.upm.es;
ovicente@delicias.dia.fi.upm.es

Abstract. We present WebODE as a scalable, integrated workbench for ontological engineering that eases the modelling of ontologies, the reasoning with ontologies and the exchange of ontologies with other ontology tools and ontology-based applications. We will first describe the WebODE's knowledge model. We will then describe its extensible architecture, focusing on the set of independent ontology development functionalities that are integrated in this framework, such as the Ontology Editor, the Axiom Builder, the OKBC-based inference engine, and the documentation and interoperability services.

1 Introduction

In the last decade, many definitions for the term "ontology" (in the context of AI) have appeared [15] and some of them have changed and evolved over the time. We think that the most representative definition is: "an ontology is a formal specification of a shared conceptualization" [19] (which extends Tom Gruber's and Pim Borst's ones).

Several tools for ontology development have also come up on the last decade: Ontolingua [9], OntoSaurus [21], WebOnto [7], Protégé2000 [22], OilEd [3], OntoEdit [20], etc. (a study on some of these tools appeared in [8]); and others for merging ontologies (Chimaera [18], PROMPT [11]), translating ontologies into ontology languages (Ontomorph [4]), annotating web pages with ontological information (OntoMat, SHOE Knowledge Annotator [16], COHSE [2]), etc.

These tools perform different activities of the ontology development process (design, implementation, merge and annotation, among others). However, most of these tools have been built as isolated independent tools and they are not normally capable of **interoperating** among them [13]. In fact, heterogeneous ontology platforms should be able to exchange ontologies owned by different organizations, built with different tools, and implemented on different languages; but no guidelines are available about how to make platforms mutually interoperable. The need for a deep study on tools' interoperability is being addressed in the Special Interest Group on Enterprise-Standard Ontology Environments of the European IST network OntoWeb and a survey on these tools can be found at [24].

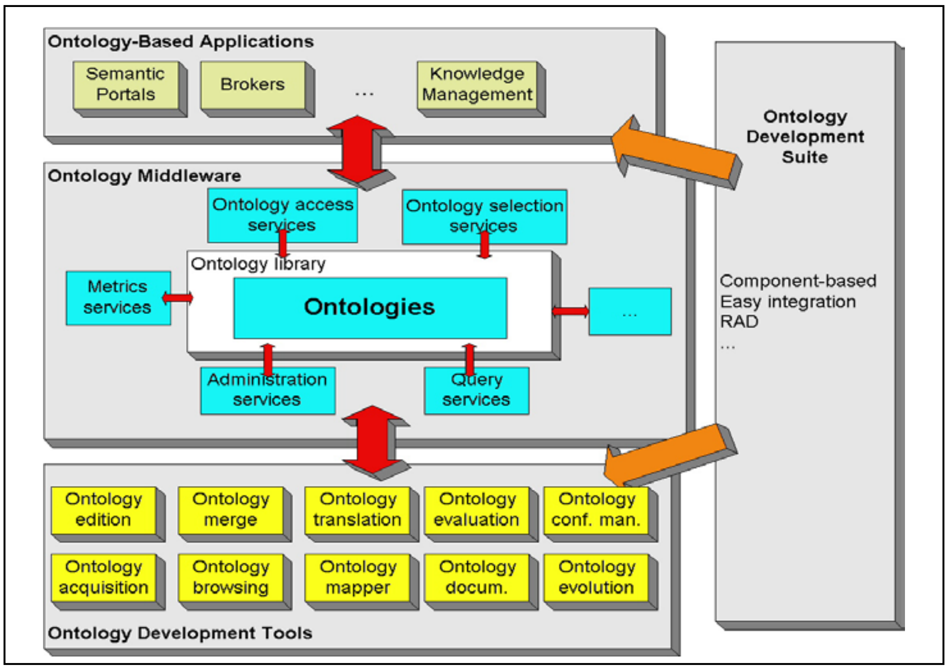


Fig 1. Main modules of an ontological engineering workbench (adapted from [13]).

Except for Protégé2000 [22] and OntoEdit [20], the other tools do not provide an integrated **support for most of the activities of the ontology lifecycle**, nor do they support any existing **methodology** for building ontologies.

Taking into account this situation, in [13] we presented the need for an integrated ontological engineering workbench supporting three groups of activities (see figure 1): (1) ontology development, management and population activities; (2) ontology middleware services to allow the easy used and integration of ontological technology in information systems; and (3) ontology-based applications' development suites to ease the creation of ontology-based applications.

In the context of this framework, we have developed WebODE[1] as a scalable ontological engineering workbench that gives support to most of the ontology development, management and population activities presented in figure 1. It also includes *middleware services* to aid in the integration of ontologies into real-world and Semantic Web applications as well as to provide rapid development tools for applications using ontologies. Finally, WebODE has been created to provide technological support for Methontology [10], an ontology construction methodology. Nevertheless, this fact does not prevent it from being used following other methodologies or no methodological approach at all.

Next sections describe the knowledge model, architecture and main features of WebODE. We will specially focus on the WebODE Axiom Builder (WAB) and the WebODE inference engine.

[1] http://webode.dia.fi.upm.es/

2 WebODE's Knowledge Model

The WebODE's knowledge model [1] is based on the intermediate representations proposed in Methontology [10]. Hence, it allows modelling concepts and their attributes (both class and instance attributes), taxonomies of concepts, disjoint and exhaustive class partitions, ad-hoc binary relations between concepts, properties of relations, constants, axioms and instances of concepts and relations.

Bibliographic references can be attached to any of the aforementioned ontology components. Besides, it is possible to import terms from other ontologies. Imported terms are referred to by means of URLs.

Now we describe in depth all the components of the WebODE's knowledge model:

- **Concepts** are identified by their *name*, though they can also have *synonyms* and *abbreviations*. A natural language (NL) *description* can be also included.
  - **Class attributes** are attributes that specify characteristics of a class, and whose value is the same for all the instances of the concept. They are defined with the following information: *attribute name* (which must be different from the rest of attribute names of the same concept); *name of the concept* it belongs to (attributes are local to concepts, that is, two different concepts can have different attributes with the same name); *value type* or *range*, which can be a basic data type (String, Integer, Cardinal, Float, Boolean, Date, Numeric Range, Enumerated, URL) or a concept (specified by the concept name); *minimum* and *maximum cardinality*, which constrains the number of values that the class attribute may have; and *value(s)*.
    Class attributes can optionally have a *NL description*, *measurement unit* and *precision* (the last two ones just in case of numeric attributes).
  - **Instance attributes** are attributes whose value may be different for each instance of the concept. They have the same properties than class attributes and two additional properties, *minimum value* and *maximum value*, which are used in attributes with numeric value types. Values inserted for instance attributes are interpreted as default values for them.
- **Concept groups** are sets of disjoint concepts that are also known as partitions. They are used to create disjoint and exhaustive class partitions. They have a *name*, the *set of concepts* they group together and, optionally, a *NL description*. A concept can belong to several concept groups.
- **Built-in relations** are predefined relations in the WebODE's knowledge model. They are divided into three groups: taxonomy relations between concepts (*subclass-of*, *not-subclass-of*), taxonomy relations between groups and concepts (*disjoint-subclass-partition*, *exhaustive-subclass-partition*), and mereology relations between concepts (*transitive-part-of*, *intransitive-part-of*).
- **Binary ad-hoc relations** between concepts are characterized by their *name*, the *origin* (source) and *destination* (target) concepts, and their *cardinality*, which establishes the number of destination terms of each origin term through the relation. Cardinality can be restricted to 1 (only one destination term) or N (any number of destination terms). Optionally, we can provide their *NL description* and *properties* (they are used to describe algebraic properties of the relation).

- **Constants** are components that have always the same value and can be used in any expression. They are identified by their *name*, and have a *value type*, *value* and *measurement unit*. Its *NL description* can be optionally provided.
- **Axioms** and **rules** are defined by their *name*, an optional *NL description* and a *formal expression* in first order logic (using the syntax provided by WebODE). They will be deeply studied in section 4.2.
- **Properties** are used to describe algebraic properties of ad-hoc relations. They are divided in two groups: *built-in properties* (reflexive, irreflexive, symmetric, asymmetric, antisymmetric and transitive), and *ad-hoc user-defined properties*.
- **Imported terms** are components from other ontologies that are included in another ontology. They are described by their *name* and a URL that includes the *host* and the *ontology name* from which the term is retrieved and the *term name* in that ontology.

Besides, the WebODE's knowledge model supports **views** and **instance sets**. Views are used to highlight specific parts of the ontology. They are analogous to the classical views of database modelling theory.

Concerning instance sets, they make possible to populate a conceptual model for different applications or scenarios, maintaining different, independent instantiations of the same conceptual model in WebODE.

3 WebODE's Architecture

WebODE is built according to a four-tier architecture: client, presentation, business logic, and database tiers. In all these tiers, we have used standard technology. The client tier uses HTML, XML, CSS, JavaScript and Java applets. The presentation tier uses servlets and JSPs. The business logic tier uses Java and RMI-IIOP. Finally, the database tier uses JDBC and Oracle.

We will analyse further the database and business logic tiers, which comprise the "ontology middleware" modules and services from figure 1.

3.1 Database tier

Currently, WebODE ontologies can be stored in any relational database with JDBC support (WebODE has been tested both in Oracle and MySQL, which gives an idea of the flexibility of this workbench). Its main features are the optimisation of connections to the database (*connection pooling*) and transparent *fault tolerance* capabilities.

Besides, the underlying physical model that represents the WebODE knowledge model has been tuned for obtaining maximum performance.

The database access is abstracted out as an independent service in the platform. In fact, it has been designed as a pluggable component in the architecture. This allows its easy replacement by other modules in charge of data management, such as XML-based databases or RDF-based repositories.

3.2 Business logic tier

The business logic tier has been implemented through a proprietary Java and RMI-IIOP based application server called Minerva_LIA. This application server provides an API to create services, which can be added or removed easily with its management console, thus improving the system's flexibility, scalability and integration with third-party solutions, following the latest trends in enterprise middleware.

Minerva_LIA application server's core (built-in) services comprise the basic building blocks for more complex services in WebODE. They are the following: authentication, log, administration, thread management, scheduling and backup.

The other services are plugged on top of the Minerva_LIA application server. In figure 2 we show the structure of all the services used by the WebODE ontology editor. Any of these components can be plugged in or out, so that the whole WebODE workbench and the WebODE ontology editor can be easily personalised.

The core of the WebODE's ontology development services are: the database service (db), the cache, consistency and axiom services, and the ontology access service (ode), which defines an API for accessing WebODE ontologies. One of the main advantages of this architecture is that these services can be accessed remotely from any other application or any other instance of the WebODE workbench.

Finally, the WebODE interoperability services and the ontology documentation service are completely based on the WebODE ontology access API, and the inference engine uses extensively the Prolog exportation service. Other middleware services, such as ODEMerge, ODECatalogue and WebPicker also use the WebODE ontology access API.

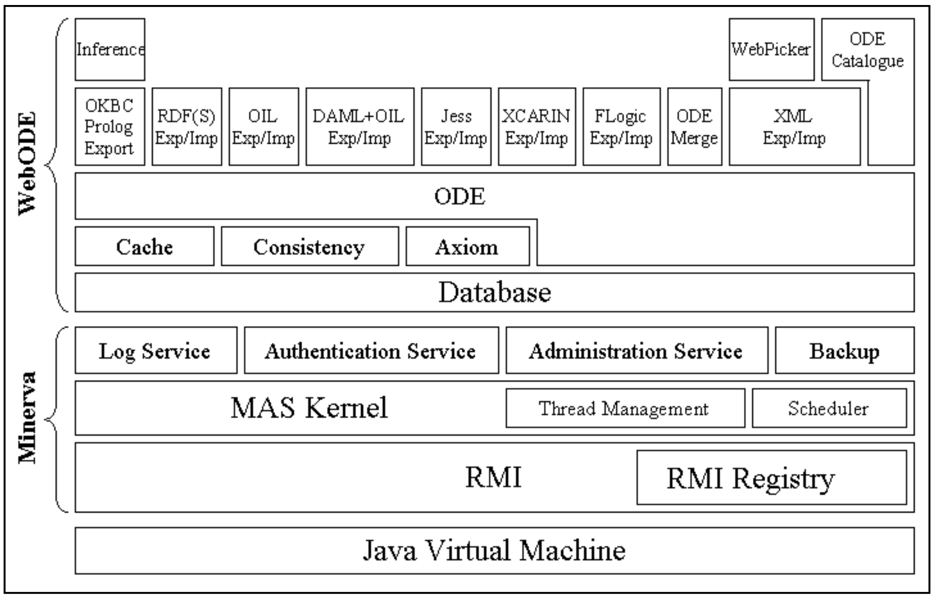
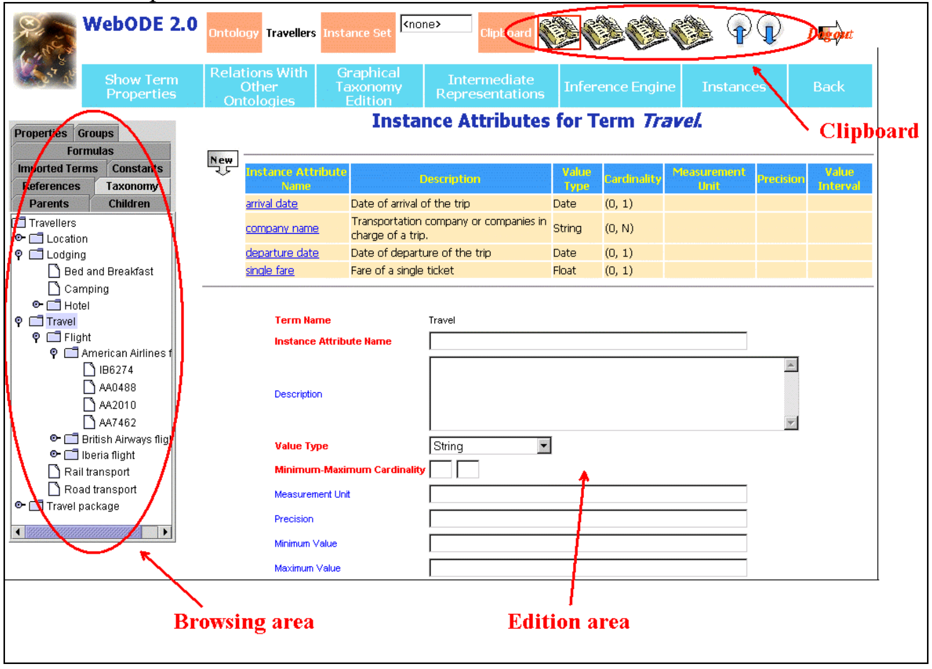


Fig 2. WebODE ontology engineering workbench's architecture.

4 WebODE Ontology Development Services

4.1 Ontology edition service

The WebODE Ontology Editor allows the collaborative construction of ontologies at the knowledge level. It provides a default form-based web user interface to create ontologies according to the knowledge model aforementioned. Figure 3 shows the look and feel of the ontology editor, as well as its three main areas. The main user interface components are:



Instance Attributes for Term *Travel*.

Instance Attribute Name	Description	Value Type	Cardinality	Measurement Unit	Precision	Value Interval
arrival date	Date of arrival of the trip	Date	(0, 1)			
company name	Transportation company or companies in charge of a trip.	String	(0, N)			
departure date	Date of departure of the trip	Date	(0, 1)			
single fare	Fare of a single ticket	Float	(0, 1)			

Term Name	Travel
Instance Attribute Name	
Description	
Value Type	String
Minimum-Maximum Cardinality	
Measurement Unit	
Precision	
Minimum Value	
Maximum Value	

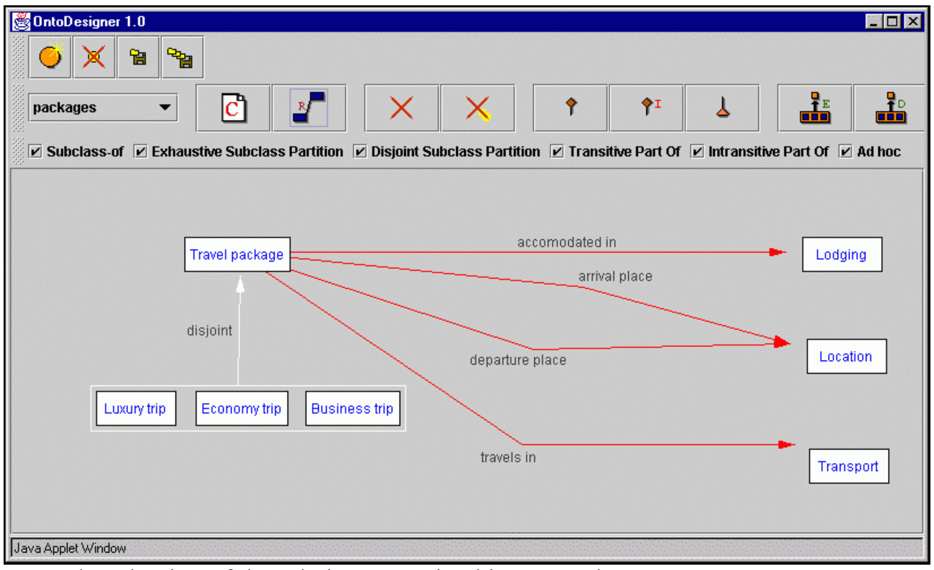
Fig 3. WebODE Ontology Editor

- Browsing area. It allows browsing the whole ontology and provides operations to create new elements and modify or delete the existing ones.
- Clipboard. It allows copying and pasting information easily between forms.
- Edition area. It presents the forms to be filled by the user, according to the component (concept, attribute, relation, etc.) that is being edited.

The WebODE Ontology Editor also includes *OntoDesigner*, a visual tool that aids in the construction of concept taxonomies and ad-hoc relations between concepts.

Concept taxonomies are created with the following set of predefined relations: *subclass of*, *disjoint decomposition*, *exhaustive partitions*, *transitive part of* and *non-transitive part of*. In figure 5 we show a snapshot of OntoDesigner while editing an ontology on the travelling domain.

With OntoDesigner, users can create different views of an ontology, so that they can highlight parts of the ontology, as explained in section 2. Moreover, users can decide whether showing or hiding different kinds of relations among concepts (either predefined or ad-hoc ones), in the sense of a graphical prune. This feature helps in the



manual evaluation of the relations contained in an ontology.

Fig 4. OntoDesigner. Some concepts, groups and taxonomic and ad-hoc relationships.

4.2 WAB: WebODE Axiom Builder service

Axioms and rules are important modelling components in the WebODE's knowledge model. However, we noticed that ontology developers did not usually include them in their domain ontologies. The reason for this was twofold: either (a) they did not know the exact syntax they had to use to define axioms and rules; or (b) they found difficulties to write them in a simple HTML form, as WebODE did not provide adequate support for the axiom modelling task.

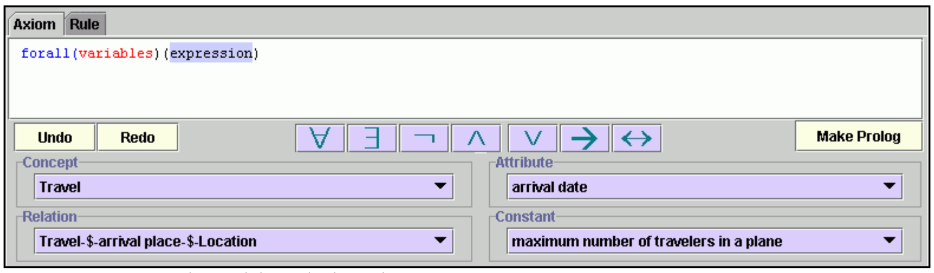
To solve this problem, we have created WAB (WebODE Axiom Builder). WAB is an axiom and rule editor that is integrated in the WebODE Ontology Editor. It allows creating first order logic axioms and rules using a graphical user interface. It also provides a library of built-in axioms, which can be reused for creating other axioms, rather than building them from scratch.

4.2.1 Axiom building

We will explain how WAB works using an example. Let us suppose that we want to create the following axiom: “every train that departs from a European location must arrive at another European location”. This axiom is written in WebODE as follows:

```
forall(?X,?Y,?Z)  (Train(?X) and
                    Origin(?X,?Y) and EuropeLocation(?Y) and
                    Destination(?X,?Z)
                    -> EuropeLocation(?Z))
```

Figure 5 shows the WAB interface once we have pressed the universal quantification symbol in order to write our axiom. In this sense, this interface helps



```
forall(variables)(expression)
```

Concept: Travel
Attribute: arrival date
Relation: Travel-$-arrival place-$-Location
Constant: maximum number of travelers in a plane

non-expert users in writing their axioms.

Fig 5. WAB: WebODE Axiom Builder. Well-formed axioms' construction

Figure 6 shows the axiom completed in WAB. In this axiom editor, we can create well-formed expressions in first order logic, using: universal quantification, existential quantification, negation, conjunction, disjunction, implication and biconditional. We can also use those terms that have been already defined in the ontology (concepts, attributes, relations and constants). If we select a concept in the *Concept* drop-down list, shown in figure 4, the attributes and relations that can be applied to this concept appear in the *Attribute* and *Relation* drop-down lists, respectively (including those attributes and relations that are not defined directly in the concept but are inherited in the concept taxonomy). In the example of figure 6, *companyName* is an attribute of concept *MeanOfTransport* and *Destination* is an ad-hoc relation between *MeanOfTransport* and *Location*. This prevents users from entering attributes or relations that cannot be applied to a concept.

WAB also allows users to write directly the axiom expression, without using its facilities for doing it. The following grammar is used in WebODE axioms:

axiom ::= atom | axiom **OR** axiom | axiom **AND** axiom | axiom **->** axiom | axiom **<->** axiom |
NOT axiom | **FORALL** (var_list) axiom | **EXISTS** (var_list) axiom | (axiom)

atom ::= ID (term_list) | **SUBCLASS** (term_list) | **NOT_SUBCLASS** (term_list) |
DISJOINT (term_list) | **EXHAUSTIVE** (term_list) | **TRANSITIVE** (term_list) |
INTRANSITIVE (term_list) |
term **>** term | term **<** term | term **>=** term | term **<=** term | term **=** term | (atom)

term ::= ID | num | ID (term_list) | term **+** term | term **-** term | term ***** term | term **/** term | (term)

term_list ::= term | term **,** term_list

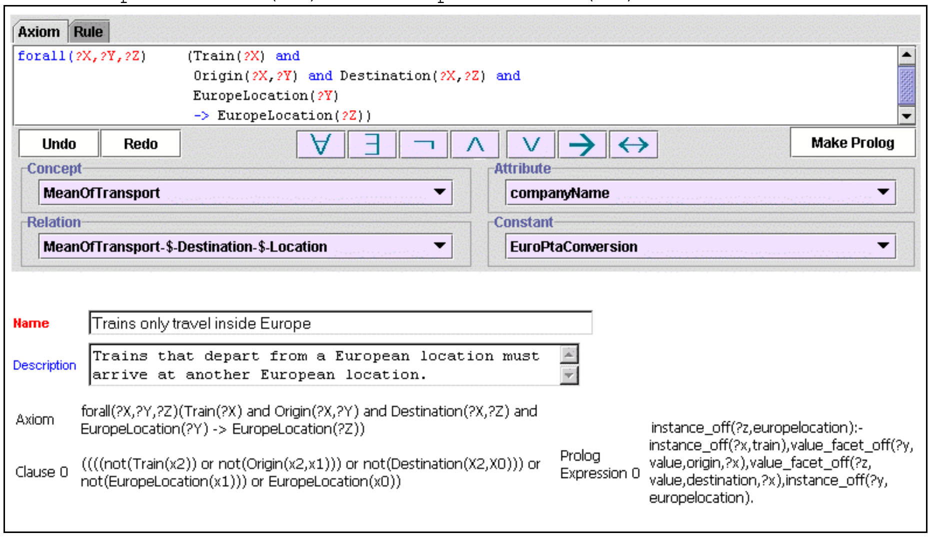
var_list ::= ID | ID **,** var_list

Once the axiom is defined, we must click on the “Make Prolog” button to parse it and ensure that it has been correctly defined. Not only does WAB perform a syntactic check of the axiom, in the sense of testing that it is compliant with the WebODE's axiom grammar. But also it checks that the vocabulary used in the axiom is defined in the ontology, that the ad-hoc relations can be applied between two variables, that the attributes are defined for the concepts to which they are applied, etc.

The final result of this parsing is that the axiom is transformed to Horn clauses, if possible. To perform this transformation, we follow a well-known process. First, WAB generates the prenex form of the axiom; then the Skolem Normal Form; next, it generates the Conjunctive Normal Form and, finally, WAB obtains the Horn clauses.

The result of this process for the axiom presented above is the following:

```
¬Train(x2) ∨ ¬Origin(x2,x1) ∨ ¬Destination(x2,x0) ∨
¬ EuropeLocation(x1) ∨  EuropeLocation(x0)
```



```
forall(?X,?Y,?Z)     (Train(?X) and
                      Origin(?X,?Y) and Destination(?X,?Z) and
                      EuropeLocation(?Y)
                      -> EuropeLocation(?Z))
```

Name: Trains only travel inside Europe

Description: Trains that depart from a European location must arrive at another European location.

Axiom: forall(?X,?Y,?Z)(Train(?X) and Origin(?X,?Y) and Destination(?X,?Z) and EuropeLocation(?Y) -> EuropeLocation(?Z))

Clause 0: ((((not(Train(x2)) or not(Origin(x2,x1))) or not(Destination(X2,X0))) or not(EuropeLocation(x1))) or EuropeLocation(x0))

Prolog Expression 0: instance_off(?z,europelocation):- instance_off(?x,train),value_facet_off(?y, value,origin,?x),value_facet_off(?z, value,destination,?x),instance_off(?y, europelocation).

Fig 6. WAB: WebODE Axiom Builder. Axiom transformation into Prolog

For each clause obtained in the previous transformation, WAB creates a Prolog rule, which can be stored in the WebODE database, so that they can be used in the WebODE's Prolog inference engine. In this process, the vocabulary used in the logical expression is also transformed according to the vocabulary provided by the OKBC protocol knowledge representation primitives [5]. As shown below, our example uses the OKBC primitives *instance_of* and *value_facet_of* are used.

```
instance_of(Z,europelocation):-
      instance_of(X,train),
      value_facet_of(Y,value,origin,X),
      value_facet_of(Z,value,destination,X),
      instance_of(Y,europelocation).
```

4.2.2 Rule building

The same approach is used for creating rules with WAB. Let's suppose that we want to create a rule that states that "all the trips by ship that depart from Europe are handled by the company Costa Cruises". In this case, we will create the following rule (shown in figure 7):

```
if EuropeLocation(Y) and Origin(X,Y) and Ship(X)
then companyName(X,costaCruises)
```

The syntax of WebODE rules is much simpler than that used for axioms. This eases the modelling of knowledge in the form of “if ... then” structures, which are used in many systems. The following grammar is used for WebODE rules:

```
lhs ::= atom | atom AND lhs
rhs ::= atom
rule ::= IF lhs THEN rhs
```

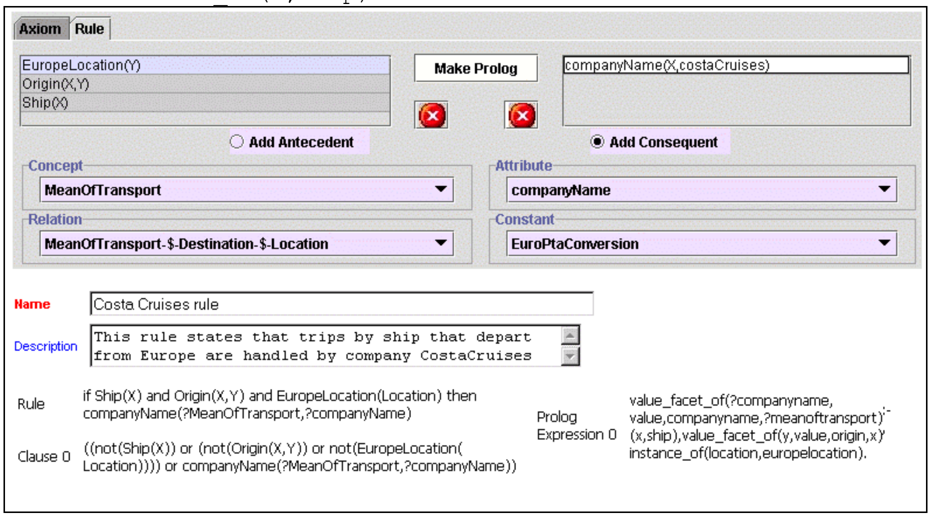
Hence, the left-hand side of the rule consists of conjunctions of atoms, while the right-hand side of the rule is a single atom.

Once a rule is created, WAB checks both the syntax of the rule and its consistency with the rest of the ontology, and it transforms the rule into Horn clauses. For the example above, we obtain the following Horn clause:

```
¬EuropeLocation(Y) ∨ ¬Origin(X,Y) ∨ ¬Ship(X) ∨
companyName(X,costaCruises)
```

And, consequently, the following Prolog rule, which can be stored in the WebODE database:

```
value_facet_of(costaCruises,value,companyname,X):-
    instance_of(Y,europelocation),
    value_facet_of(Y,value,origin,X),
     instance_of(X,ship).
```



Antecedents	EuropeLocation(Y)
	Origin(X,Y)
	Ship(X)

Consequent	companyName(X,costaCruises)
Concept	MeanOfTransport
Attribute	companyName
Relation	MeanOfTransport-$-Destination-$-Location
Constant	EuroPtaConversion
Name	Costa Cruises rule
Description	This rule states that trips by ship that depart from Europe are handled by company CostaCruises
Rule	if Ship(X) and Origin(X,Y) and EuropeLocation(Location) then companyName(?MeanOfTransport,?companyName)
Clause 0	((not(Ship(X)) or (not(Origin(X,Y)) or not(EuropeLocation(Location)))) or companyName(?MeanOfTransport,?companyName))
Prolog Expression 0	value_facet_of(?companyname, value,companyname,?meanoftransport):- (x,ship),value_facet_of(y,value,origin,x)' instance_of(location,europelocation).

Fig 7. WAB: WebODE Axiom Builder. Rule edition

The transformations of axioms and rules to the OKBC vocabulary are done because the WebODE's inference engine allows working with the primitives defined in OKBC, as we will see in the next section. Moreover, WebODE ontologies are completely translated to Prolog syntax, so that these generated Prolog rules can be used either for checking constraints on the ontology or for generating new information from it. We will see the results of the Prolog translation in section 4.4.

4.3 WebODE's inference engine service

As we have commented in section 4.2, WebODE includes an **OKBC-based inference engine**. This inference engine reasons with a subset of the primitives identified in that protocol [5], and allows using such primitives to query ontologies (currently, WebODE's inference engine makes use of Ciao Prolog [17]).

The following groups of OKBC primitives have been implemented:

- Primitives to query about concept taxonomies and instances: *get-class-instances*, *get-class-subclasses*, *get-class-superclasses* and *get-instance-types*.
- Primitives to query about slots: *get-frame-details*, *get-slots*, *get-slot-domain*, *get-slot-type*, *get-slot-value*, *get-slot-values*, *get-slot-values-in-detail*, *get-slot-facets*, *get-facet-value* and *get-facet-values*.
- Primitives to check whether a condition holds for a term (class, instance or slot): *individual-p*, *instance-of-p*, *member-facet-value-p*, and *member-slot-value-p*.

At present, we are using this inference engine for several purposes:

- Querying about the ontology components, either with these OKBC primitives or with a user-defined Prolog program, as long as it uses these OKBC primitives or the Prolog representation of ontology components, which are presented in section 4.4.
- Asserting new knowledge using the Prolog expressions that correspond to the rules and axioms created with WAB, as explained above.
- Detecting inconsistencies in the ontology. Each axiom in the ontology can be tested independently from the other ones, as long as it can be transformed into Horn clauses. It is important to mention that, in this sense, we are not using axioms for theorem proving, but just for constraint checking.

The WebODE's inference engine is also used by the WebODE's OntoClean service. This service uses the OntoClean method [23], which allows cleaning concept taxonomies according to philosophical notions such as: rigidity, identity and unity.

Besides, the WebODE's inference engine has been designed to be easily extensible, so that other inference engines can be attached to it and used with the same interface.

Finally, WebODE also provides other constraint checking capabilities, though it does not use the WebODE inference engine for it. It checks type constraints, numerical values constraints, cardinality constraints and taxonomic consistency [14] (i.e., common instances of disjoint classes, loops, etc.). These evaluation capabilities can be used when an ontology is built either using the form-based user interface or OntoDesigner. Such functionality is supplied through the use of a Minerva_LIA service, as part of the ontology development and management services.

4.4 WebODE interoperability services

Ontologies built using WebODE can be easily integrated in other ontology servers or used in ontology-based applications. Possible choices for interoperability include:

- WebODE's ontology access API, which can be accessed by other applications using RMI, and is completely compliant with the WebODE's knowledge model.
- XML. WebODE ontologies can be exported into and imported from XML, following a well-defined DTD[2] that uses the same knowledge representation vocabulary used for expressing the WebODE knowledge model. Ontologies can be translated completely or on a view or instance-set basis.
- Ontology languages, through the ontology language export/import modules. Currently, WebODE is able to export to and import ontologies from: RDF(S), OIL, DAML + OIL, the XMLization of CARIN and FLogic. If we take into account that the WebODE knowledge model is very expressive [1], we are able to provide high quality translations that preserve most of the original information contained in the ontology and take advantage of most of the modelling characteristics of the target and source languages. As with XML, ontologies can be translated completely or on a view or instance-set basis.
- Jess [12]. WebODE generates all the concepts as Java beans, which contain also information about their attributes and ad-hoc relations. These beans can be easily uploaded in the Jess system. This means that we can use the ontology developed in WebODE inside the Jess system, and develop our own programs in Jess using the ontology components.
- Prolog syntax of OKBC [5]. WebODE provides a subset of the primitives defined in the OKBC protocol. They are expressed in Prolog, as explained in the previous section. Table 1 summarizes the mapping between the WebODE's knowledge model and the Prolog OKBC translation.

WebODE ontology component	***OKBC Prolog representation***
Concept: Travel	**class**: class(travel)
Concept groups	-- (they do not exist in OKBC)
Class attribute: Hotel quality	**own slot**: own-slot-of(quality,hotel)
Instance attribute: Hotel price	**template slot**: template-slot-of(price,hotel)
Subclass-of: Flight is a subclass of Travel	**subclass of**: subclass-of(flight,travel)

Ad-hoc relation:
the departure place of a Travel is a Location
	slot:
	slot-of(departurePlace,travel)
	facet-of(type,departurePlace,travel,location)

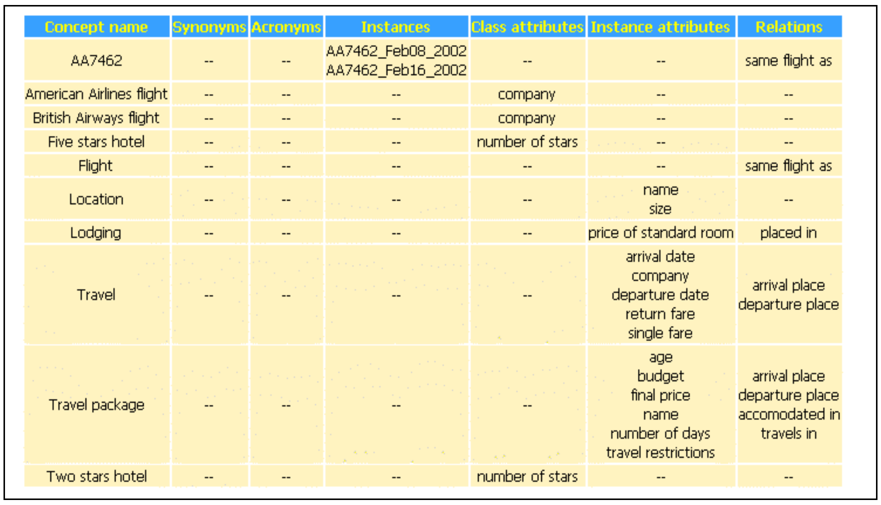
Constant: average price	**term**: averagePrice
Axiom & rule	**Prolog rule**: cf. section 4.2
Instance: John is a Traveller	**instance**: instance-of(john,traveller)
Property	-- (they do not exist in OKBC)
Imported term: Date	**term**: date

Table 1. Summary of WebODE transformations into OKBC Prolog syntax.

[2] http://webode.dia.fi.upm.es/webode/dtd/webode_2_0.dtd

4.5 WebODE's ontology documentation service

WebODE ontologies are automatically documented in different formats: HTML tables representing the Methontology's intermediate representations, HTML concept taxonomies and XML.



Concept name	Synonyms	Acronyms	Instances	Class attributes	Instance attributes	Relations
AA7462	--	--	AA7462_Feb08_2002 AA7462_Feb16_2002	--	--	same flight as
American Airlines flight	--	--	--	company	--	--
British Airways flight	--	--	--	company	--	--
Five stars hotel	--	--	--	number of stars	--	--
Flight	--	--	--	--	--	same flight as
Location	--	--	--	--	name size	--
Lodging	--	--	--	--	price of standard room	placed in
Travel	--	--	--	--	arrival date company departure date return fare single fare	arrival place departure place
Travel package	--	--	--	--	age budget final price name number of days travel restrictions	arrival place departure place accomodated in travels in
Two stars hotel	--	--	--	number of stars	--	--

Fig 8. WebODE's documentation service: Concepts Dictionary Intermediate Representation.

Users can decide whether obtaining the documentation of the whole ontology or of parts of it (specific views or instance sets). As an example, figure 8 presents part of the concept dictionary of the *Travel* ontology, which contains its concepts, their class and instance attributes, instances and ad-hoc binary relations.

The HTML documentation service shows the concept taxonomy, the concept attributes and the ad-hoc binary relations between concepts.

5 WebODE middleware services

In this section, we present some *middleware* applications that we have built inside the WebODE workbench. They are fully integrated in the middle tier and, as such, run within the Minerva_LIA application server. They use some of the services described in this paper, such as the interoperability services and the inference engine.

- **WebPicker** [6] is a set of wrappers that allow importing standards of classification of products and services in the context of electronic commerce into WebODE (UNSPSC, e-cl@ss and RosettaNet). We are currently extending it to wrap other sources of information, such as Cyc.
- **ODEMerge** performs merging of concepts, attributes and relationships from two different ontologies built for the same domain, according to semantic criteria and resources used for natural language processing.
- **ODECatalogue** is able to generate electronic catalogs from ontologies according to some parameters. The catalogue generation from an ontology assures a correct and rich classification of the different products.

6 Conclusions

In this paper, we have presented the WebODE ontological engineering workbench, whose main contributions are detailed below:

1) **Integrated technological support for many activities of the ontology lifecycle**.

- WebODE supports in an integrated platform many activities of the ontology lifecycle that, until recently, have just been supported by isolated, independent tools. In fact, it does not only support development activities, but also management and support ones, such as documentation, evaluation, merge or integration.
- Additionally, WebODE allows developing ontologies at the knowledge level. Ontology developers do not need worry about building an ontology directly in an implementation language. Later, the contents of the ontology can be automatically translated into several implementation languages.
- First order logic axioms and rules can be more easily created according to the WebODE syntax, using the WebODE Axiom Builder. They are also translated into Prolog syntax, if possible, using some primitives extracted from the OKBC knowledge model. Primitives from the OKBC protocol can be used to send queries about WebODE ontologies with the Prolog inference service.
- Finally, WebODE is not only useful for building ontologies, but also provides a wide range of services for ontology-based applications.

2) **Technological support for ontology development methodologies.**

- WebODE has been built to provide support to a methodology for ontology development: Methontology.
- However, this does not prevent WebODE from being used with another ontology development methodology or without any specific methodological approach.

3) **Ontology interoperability.**

- Interoperability amongst modules and services inside the WebODE workbench. The use of a common API to access WebODE ontologies from any service and the use of XML exportation/importation functionalities allow modules and services interoperate easily.
- Interoperability with other tools and applications. The translation functionalities available in the workbench allow exchanging ontologies with other tools, environments or applications.

This workbench has been successfully used, with different domains and purposes and by different groups of people, in the following projects:

- The European IST project MKBEEM (IST 1999-10589). In this project, B2B and B2C ontologies have been built and reengineered using WebODE.
- The Ontoweb thematic network (IST-2000-29243). We have built the OntoRoadMap application[3] on top of WebODE. It is an ontology-based web application that allows the community to register, browse and search ontologies,

[3] http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html

methodologies, tools and languages for building ontologies, as well as ontology-based applications in areas like the: semantic web, e-commerce, KM, NLP, III, etc., This application uses an ontology in the domain of ontologies.

- The Spanish CICYT project ContentWeb (TIC2001-2745). In this project, we have created WebPicker for (semi)automatic ontology acquisition from e-commerce standards for the classification of products and services (*UNSPSC*, *RosettaNet* and *e-cl@ss*) and in the domain of leisure activities.
- The Spanish CICYT project on Methodology for Knowledge Management (TIC-980741). We have built using WebODE ontologies that model a few institutions.
- (Onto)^{2}Agent. We have built the Reference Ontology, that asseses ontology-based applications' developers on the most suitable ontology to use in an application.
- Environment Ontology (UPM-AM-9819). In this project, we have built the *Elements* and *Environmental Ions* ontologies, in the domain of chemistry, and have integrated them with existing ontologies, such as the Ontolingua's ontology Standard Units.
- MRO (Ontologies for cataloguing business services). In this project, we have merged heterogeneous electronic catalogues in the domain of office furniture.

In the future we will provide more functions both to the WebODE Ontology Editor and the *middleware* area, such as an ontology translation manager for the WebODE interoperability services, ontology configuration management capabilities, ontology upgrading and semantic annotation services. We will therefore lay the foundations for a more complete implementation of the workbench presented in section 1. We will also focus on the implementation of an ontology development suite that allows a high reusability of ontologies and rapid creation of ontology-based applications. Finally, WebODE services will be soon made available as Semantic Web services.

Acknowledgements

This work is supported by a FPI grant funded by UPM and by the project "ContentWeb: Plataforma tecnológica para la web semántica: ontologías, lenguaje natural y comercio electrónico[4]" (TIC-2001-2745). This work would not have been possible without the help of JC Arpírez, JF Cebrián, R de Diego, M Lama, A López, V López, E Mohedano, JP Pérez and JA Ramos in the implementation and tests of WebODE services, and M Blázquez and JM García-Pinar in the development of ODE.

References

1. Arpírez, J.C.; Corcho, O.; Fernández-López, M.; Gómez-Pérez, A. *WebODE: a scalable ontological engineering workbench*. First International Conference on Knowledge Capture (K-CAP 2001). Victoria, Canada. October, 2001.
2. Bechhofer, S., Goble, C. *Towards Annotation Using DAML+OIL*. KCAP'01 Workshop on Semantic Markup and Annotation. Victoria, Canada. October, 2001.

[4] ContentWeb: Platform for the Semantic Web: ontologies, natural language and e-commerce

3. Bechhofer, S.; Horrocks, I; Goble, C.; Stevens, R. *OilEd: a Reason-able Ontology Editor for the Semantic Web*. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.
4. Chalupsky, H. *OntoMorph: A Translation System for Symbolic Knowledge*. KR-2000. 471-482. 2000.
5. Chaudhri V. K.; Farquhar A.; Fikes R.; Karp P. D.; Rice J. P. *The Generic Frame Protocol 2.0*. Technical Report, Stanford University.1997.
6. Corcho, O.; Gómez-Pérez, A. *WebPicker: Knowledge Extraction from Web Resources*. 6th Intl. Workshop on Applications of Natural Language for Information Systems (NLDB'01). Madrid. June, 2001.
7. Domingue, J. *Tadzebao and Webonto: Discussing, Browsing and Editing Ontologies on the Web*. KAW98. Banff, Canada. 1998.
8. Duineveld, A.; Studer, R.; Weiden, M; Kenepa, B.; Benjamis, R. *WonderTools? A comparative study of ontological engineering tools*. KAW99. Banff. Canada. 1999.
9. Farquhar A., Fikes R., Rice J., *The Ontolingua Server: A Tool for Collaborative Ontology Construction*. 10th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada. 1996.
10. Fernández-López, M.; Gómez-Pérez, A.; Pazos, J.; Pazos, A. *Building a Chemical Ontology using methontology and the Ontology Design Environment*. IEEE Intelligent Systems and their applications. #4 (1):37-45. 1999.
11. Fridman, N., Musen, M. *PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment*. AAAI-2000. Austin, Texas. August, 2000.
12. Friedman-Hill, E.J. *Jess, The Expert System Shell for the Java Platform*. Version 6.1a1 (3 April 2002). http://herzberg.ca.sandia.gov/jess/docs/61/
13. Gómez-Pérez, A. *A proposal of infrastructural needs on the framework of the semantic web for ontology construction and use*. FP6 Programme Consultation Meeting 9. April 27th, 2001.
14. Gómez-Pérez, A. *Evaluation of Ontologies*. International Journal of Intelligent Systems. 16(3). March, 2001.
15. Guarino, N.; Giaretta, P. *Ontologies and Knowledge Bases: Towards a Terminological Clarification*. In N. Mars (ed.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. IOS Press, Amsterdam: 25-32. 1995
16. Heflin, J.; Hendler, J. *A Portrait of the Semantic Web in Action*. IEEE Intelligent Systems, 16(2), 2001.
17. Hermenegildo, M., Bueno, F., Cabeza, D., Carro, M., García, M., López, P., Puebla, G. *The Ciao Logic Programming Environment*. International Conference on Computational Logic (CL2000). July, 2000.
18. McGuinness, D., Fikes, R., Rice, J., Wilder, S. *The Chimaera Ontology Environment*. AAAI-2000. Austin, Texas. August, 2000.
19. Studer, R.; Benjamins, V.R.; Fensel, D. *Knowledge Engineering: Principles and Methods*. IEEE Transactions on Data and Knowledge Engineering, 25(1-2), 1998, pp.161–197.
20. Sure, Y.; Erdmann, M.; Angele, J.; Staab, S.; Studer, R.; Wenke, D. *OntoEdit: Collaborative Ontology Development for the Semantic Web***.** International Semantic Web Conference (ISWC02). Sardinia. Italy. June, 2002. LNCS 2342. pp. 221-235.
21. Swartout, B.; Ramesh P.; Knight, K.; Russ, T. *Toward Distributed Use of Large-Scale Ontologies*. AAAI Symposium on Ontological Engineering. Stanford. USA. March, 1997.
22. *Using Protégé-2000 to Edit RDF*. Technical Report. Stanford University. http://www.smi.Stanford.edu/ projects/protege/protege-rdf/protege-rdf.html
23. Welty, C.; Guarino, N. *Supporting Ontological Analysis of Taxonomic Relationships*. Data and Knowledge Engineering. September 2001.

24. A survey on ontology tools. Deliverable D13.IST OntoWeb Thematic Network. May 2002.