Blázquez M, Fernández M, García-Pinar JM, Gómez-Pérez A

Building Ontologies at the Knowledge Level using the Ontology Design Environment

Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'98)
Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'98). Volume II.
Editor: Gaines, B.R.; Musen, M.
Editorial: University of Calgary, Alberta, Canadá
18-23 de abril de 1998
Banff, Alberta, Canadá.
Pages: SHARE 4.1 – 4.15

Building Ontologies at the Knowledge Level using the Ontology Design Environment

M. Blázquez, M. Fernández, J. M. García-Pinar, A. Gómez-Pérez
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn.
Boadilla del Monte, 28660. Madrid, Spain.
Tel: (34-1) 336-74-39, Fax: (34-1) 336-7412
Email: {mblazque, mfernand, jgarcia, asun}@delicias.dia.fi.upm.es

ABSTRACT

This paper discusses how ontologies can be specified at the knowledge level using the set of intermediate representations (Gómez-Pérez, Fernández & de Vicente 1996) proposed by METHONTOLOGY (Fernández, Gómez-Pérez & Juristo 1997; and Gómez-Pérez 1998). These intermediate representations bridge the gap between how people think about a domain and the languages in which ontologies are formalized. Thus, METHONTOLOGY enables experts and ontology makers unfamiliar with implementation environments to build ontologies from scratch. In this paper, we also present the ODE (Ontology Design Environment) as a software tool to specify ontologies at the knowledge level. ODE allows developers to specify their ontology by filling in tables and drawing graphs. Its multilingual generator module automatically translates the specification of the ontology into target languages.

1. INTRODUCTION

Like KBS development, ontology development faces the knowledge acquisition bottleneck problem. However, unlike KBS, the ontology developer comes up against the additional problem of not having any sufficiently tested and generalized methodologies recommending what activities to perform and at what stage of the ontology development process these activities should be performed. That is, each development team usually follows their own set of principles, design criteria and steps in the ontology development process. The absence of structured guidelines and methods hinders the development of shared and consensual ontologies within and between teams, the extension of a given ontology by others and its reuse in other ontologies and final applications. Until now, few domain-independent methodological approaches have been reported for building ontologies. Uschold's methodology (Uschold & Gruninger 1996), Grüninger and Fox's methodology (Grüninger & Fox 1995 and 1994) and METHONTOLOGY (Fernández, Gómez-Pérez & Juristo 1997 and Gómez-Pérez 1998) are the most representative. These methodologies have in common that they start from the identification of the purpose of the ontology and the need for domain knowledge acquisition. However, having acquired a significant amount of knowledge, Uschold proposes codification in a formal language and METHONTOLOGY proposes expressing the idea as a set of intermediate representations (IR) and then generating the ontology using translators. These IR's bridge the gap between how people see a domain and the languages in which ontologies are formalized. The absence of the conceptualization step causes the following problems:

- The conceptual models are implicit in the implementation codes. A reengineering process is usually required to make the conceptual models explicit.

- Ontological commitments and design criteria (Gruber 1992) are implicit in the ontology code.

- Ontology developer preferences in a given language condition the implementation of the acquired knowledge. So, when people code ontologies directly in a target language, they are omitting the minimal encoding bias criterion defined by Gruber (Gruber 1992).

- Ontology developers (who are unfamiliar with or simply inexperienced in the languages in which ontologies are coded) may find it difficult to understand implemented ontologies or even to build a new ontology. For example, it is hard for people who know how to build ontologies in one particular language and for those who know how to build ontologies (but are unfamiliar with the language in question) to work at the implementation level because the ontology maker focuses on implementation issues rather than on questions of design. For example, the expression d=m/v in a chemical domain could be written in Ontolingua (Gruber 1993) as follows:

```
(Define-Function Density( ?Element) :-> ?Density-At-20-C
"The density of an element is equal to its atomic weight dividad by its atomic volume"
:Iff-Def
(And (Elements ?Element)
        (Density-At-20-C ?Element ?Density-At-20-C)
        (Exists (?Atomic-Weight)
                ( Exists ( ?Atomic-Volume-At-20-C)
                        (And (Atomic-Weight ?Element ?Atomic-Weight)
                                (Atomic-Volume-At-20-C ?Element ?Atomic-Volume-At-20-C)
                                (> ?Atomic-Voume-At-20-C 0)
                                 (/ ?Atomic-Weight ?Atomic-Volume-At-20-C))))))
```

This example shows that unless you are very familiar with the language, it is almost impossible for you to understand existing definitions and to write new definitions. If you are successful in this process, it will have taken a big effort. So, the problem is not understanding that the density is equal to the mass divided by the volume at the knowledge level but writing this in a target language. Therefore, something which is apparently very simple at the conceptual level is extremely complicated when expressed at the implementation level, if developers are not familiar with the language. This means that ontologies are exclusively built by developers who are perfectly acquainted with the languages in which they are to be implemented. As ontology developers are not necessarily expert in the domain for which the ontology is built, ontologists spend a lot of time and resources on the knowledge acquisition process.

First, we present the METHONTOLOGY framework, including the ontology development process and a life cycle based on evolving protoypes. Then, we will show the process of building ontologies from scratch and how to re-structure an ontology when an ontological reengineering process is carried out. Finally, we will introduce the Ontological Design Environment (ODE), a software tool that supports METHONTOLOGY.

2. METHONTOLOGY FRAMEWORK

Ontological engineering requires the definition and standardization of an ontology life cycle as well as methodologies and techniques that drive their development. The METHONTOLOGY framework enables the construction of ontologies at the knowledge level and includes: the identification of the ontology development process, a life cycle based on evolving prototypes, a method for specifying ontologies at the knowledge level and multilingual translators that automatically transform the specification into several target codes. The environment for building ontologies using the METHONTOLOGY framework is called ODE (Ontology Design Environment).

2.1 Ontology development process

The ontology development process (Fernández, Gómez-Pérez & Juristo 1997) refers to *which* activities are carried out when building ontologies. It is crucial to identify these activities if consensual ontologies are to be built by geographically distant co-operative teams with some guarantee of correctness and completeness. If this is the case, it is advisable to perform the three categories of activities presented below and steer clear of anarchic constructions.

Project Management Activities include Planning, control and quality assurance. *Planning*, identifies which tasks are to be performed, how they will be arranged, how much time and what resources are needed for their completion. This activity is essential for ontologies that need to use ontologies which have already been built or ontologies that require levels of abstraction and generality. *Control*, guarantees that planned tasks are completed in the manner that they were intended to be performed. Finally, *Quality Assurance*, assures that the quality of each and every product output (ontology, software and documentation) is satisfactory.

Development-Oriented Activities include specification, conceptualization, formalization and implementation. *Specification*, states why the ontology is being built and what are its intended uses and who are the end-users. *Conceptualization*, structures the domain knowledge as meaningful models at the knowledge level. *Formalization*, transforms the conceptual model into a formal o semi-computable model. Finally, *Implementation* builds computable models in a computational language.

Support Activities include a series of activities, performed at the same time as development oriented activities, without which the ontology could not be built. They include knowledge acquisition, evaluation, integration, documentation and configuration management. *Knowledge Acquisition*, acquires knowledge of a given domain. *Evaluation*, makes a technical judgment of the ontologies, their associated software environments and documentation with respect to a frame of reference during each phase and between phases of their life cycle (Gómez-Pérez, 1996). *Integration* of ontologies is required when building a new ontology reusing other ontologies that are already available. *Documentation* is details, clearly and exhaustively, each and every one of the phases completed and products generated. *Configuration Management* records all the versions of the documentation, software and ontology code and to control the changes.

2.2. Ontology Life Cycle

It identifies the *set of stages* through which the ontology moves during its life time, describes what activities are to be performed in each state and how the states are related (relation of precedence, return, etc.). In (Fernández, Gómez-Pérez & Juristo 1997), the authors justified why the ontology life cycle is based on evolving prototypes. Figure 1 shows that:

- Planning is carried out prior to ontology development.

- Control and quality assurance are performed during the entire life cycle of the ontology.
- Most acquisition and evaluation are performed in the conceptualization phase. In the case of acquisition, this is because the conceptualization phase is like assembling a jigsaw puzzle from the pieces supplied by acquisition. As far as evaluation is concerned, although evaluation (verification and validation) (Gómez-Pérez 1996) should be performed throughout the ontology development life cycle, most evaluation is carried out in the conceptualization phase to prevent errors and their propagation in the implementation phase.
- Integration should not be understood as an integration in the implementation stage, it should be performed throughout the entire ontology life cycle.

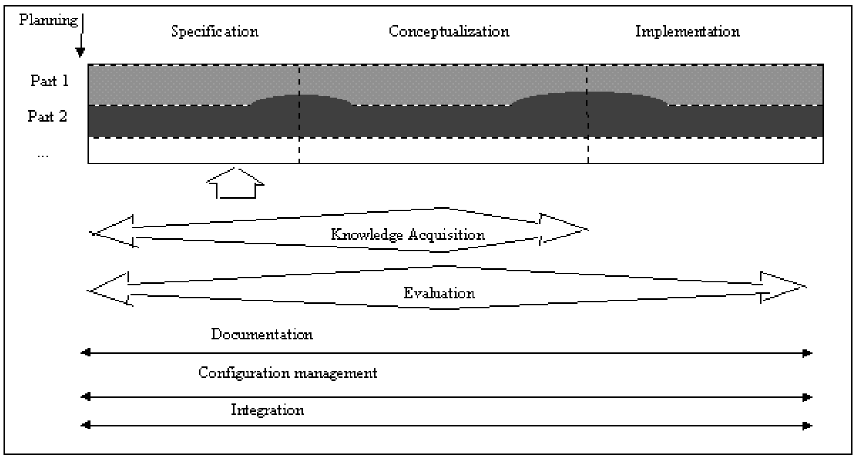


Figure 1. **Ontology Life Cicle**

3. ONTOLOGY DEVELOPMENT AT THE KNOWLEDGE LEVEL

The IRs used in the conceptualization phase are one of the essential pillars of METHONTOLOGY for building ontologies at the knowledge level. When most of the knowledge has been acquired, the ontology maker has a lot of unstructured knowledge that needs to be organized. The objective of conceptualization is precisely this: to organize and structure the acquired knowledge using external representations that are independent of the implementation languages and environments. METHONTOLOGY proposes ontology conceptualization using a set of IRs based on the IRs used in the conceptualization phase of the IDEAL methodology for knowledge-based systems development (Pazos 1995). In this paper, we discuss how ontologies were built at the knowledge level in the context of the $(KA)^2$ initiative.

$(KA)^2$ is an initiative to develop an ontology that models the Knowledge Acquisition Community (its researchers, topics, products...). A first release of this ontology was built in Flogic (Kifer, Lausen & Wu 1995). A decision was made to translate the ontology to Ontolingua to make it accessible to the entire community through the Ontology Server. Since concepts as far apart as authors, research centers, publications, etc., had been represented in a single ontology, the option of directly translating from Flogic to Ontolingua was ruled out, and it was decided to carry out an *ontological reengineering process*. First, we obtained the $(KA)^2$ conceptual model attached to the Flogic ontology manually by reverse engineering. Second, we restructured it (according to the modularity criterion) using ODE conceptualization modules and we got a new conceptual model. Finally, we converted the $(KA)^2$ conceptual model into Ontolingua language using METHONTOLOGY and ODE translators. Figure 2 shows the $(KA)^2$ ontology life cycle. Note that, in this case, our source of knowledge was an ontology that had already been implemented in the Flogic language. So, knowledge acquisition involved understanding an ontology that had already been built. The ontology implemented in Flogic was the specification we took as an input for the conceptualization phase. The knowledge restructuring process at the knowledge level was performed as shown in figure 3 and can be described as follows.

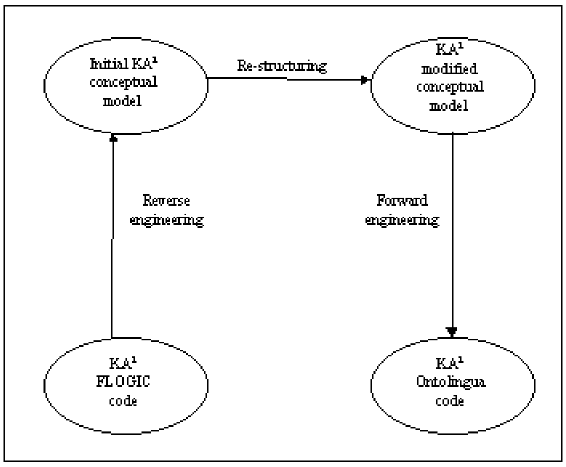


Figure 2. **Ontological Reengineering Process** *of the* KA^2 *Ontology*

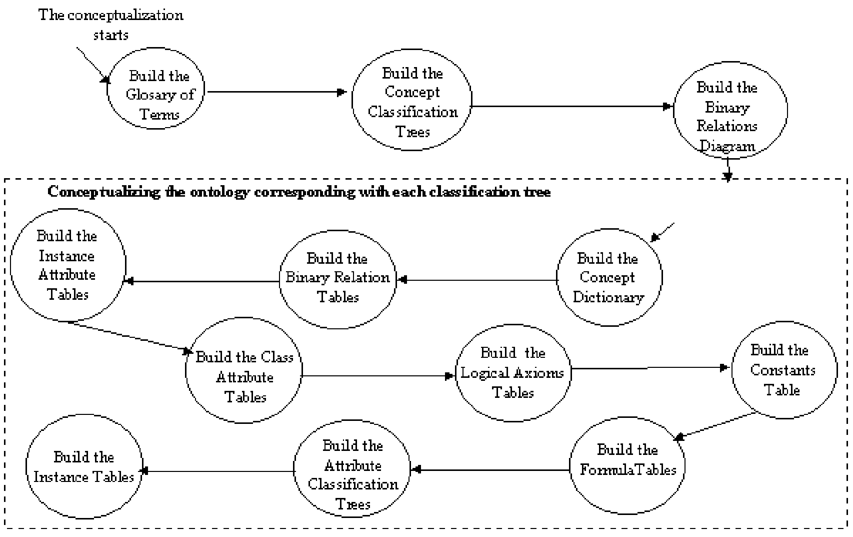

The conceptualization starts
Build the Glosary of Terms
Build the Concept Classification Trees
Build the Binary Relations Diagram
Conceptualizing the ontology corresponding with each classification tree
Build the Instance Attribute Tables
Build the Binary Relation Tables
Build the Concept Dictionary
Build the Class Attribute Tables
Build the Logical Axioms Tables
Build the Constants Table
Build the Instance Tables
Build the Attribute Classification Trees
Build the FormulaTables

Figure 3. **The knowledge restructuring process**

First, we built a *Glossary of Terms* that included all the terms (concepts, instances, attributes, verbs, etc.) of the domain and their description, as shown in table 1.

Name	Description
Academic-Staff	"He/She can be lecturer or researcher. One of his/her possible duties is to supervise PhD students"
Affiliation	"It relates an employee to the organization to which he/she belongs"
Asunción-Gómez-Pérez	A researcher in ontological engineering
Employs	"It relates an organization to its employees"
Foto	"It is the photo of the person which may appear in the web"
Researcher	"He/she is a member of the academic staff, and can be member of a research group and collaborate with other researchers"
Nicola-Guarino	"He is a researcher at CNR National Research Council. His research interest is ontologies and he works on the OntoSeek project"
Richard-Benjamins	"He is a researcher at IIIA CSIC Spanish Council from Scientific Research. His research interest is Problem Solving Methods"
Weight	It is a person's weight. It is measured in kilogrames"
…	…

Table 1. **Glosary of terms** in (KA)^2

When the glossary of terms contained a sizable number of terms, *Concept Classification Trees* were build using relations like: subclass-of[1], mutually-disjoint-subclass-of[2], exhaustive-subclass-of[3], etc. So, we identified the main taxonomies of this domain, and each taxonomy produced an ontology as prescribed by METHONTOLOGY. In the (KA)^2 ontology, we identified five taxonomies related to: people, publications, events, organizations and research topics. Figure 4 shows an outline of the above taxonomies.

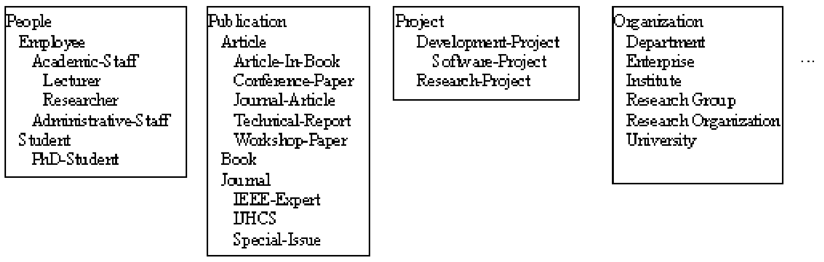


Figure 4. **Concept Classification Trees** *in (KA)^2*

The next step was to build "*Ad-hoc" Binary Relations Diagrams* between concept classification

---

[1] Class C is a subclass of parent class P if and only if every instance of C is also an instance of P.

[2] Is a set of subclasses of a class C whose objects have no elements which belong to different sets.

[3] A subrelation-partition of a class C is a set of mutually-disjoint classes (a subclass partition) which covers C. Every instance of C is an instance of exactly one of the subclasses in the partition.

trees. The goal of this diagram is to establish relationships between concepts of the same or different ontologies. Note that this diagram will set out the guidelines for integrating ontologies, because if a concept C1 is linked by a relation R to a concept C2, this means that the ontology containing C1 includes the ontology containing C2, provided that C1 and C2 are in different concept classification trees. Figure 5 presents a simplified diagram of the "ad hoc" binary relations in the $(KA)^2$ ontology. Note that the People-Ontology *includes* Organization-Ontology because the source concept of the relations named *Affiliation* is in the People-Ontology hierarchy. Similarly, if the relation *Employs* is defined as the inverse relation of *Affiliation*, the Organization-Ontology can be said to *include* the People-Ontology.

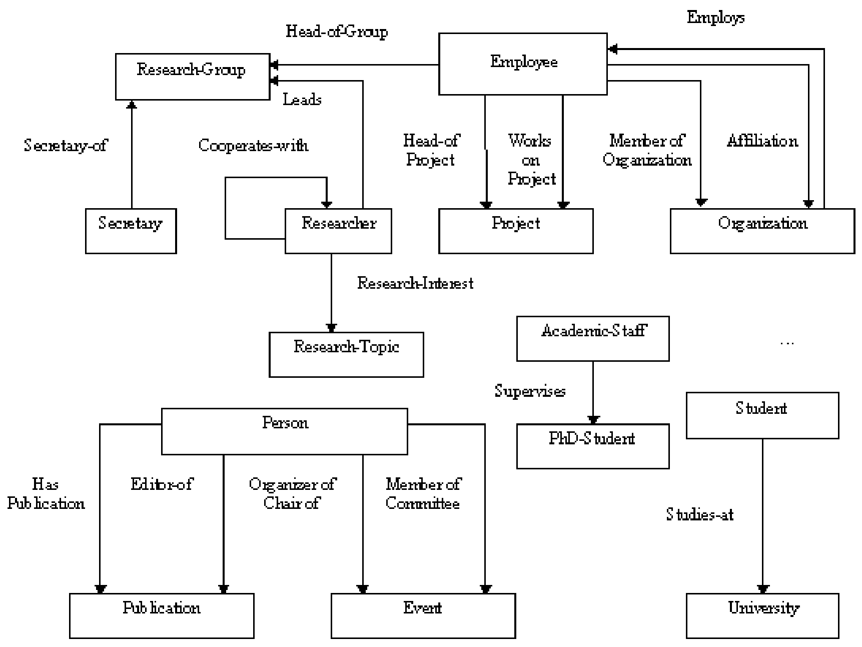
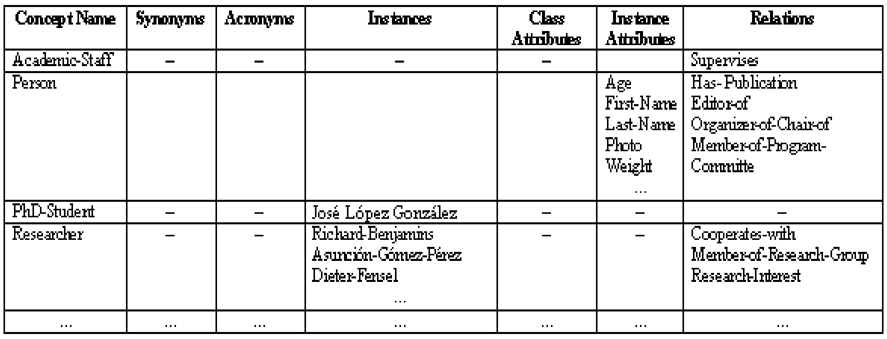


Figure 5. **Diagram of Binary Relations** *in* $(KA)^2$

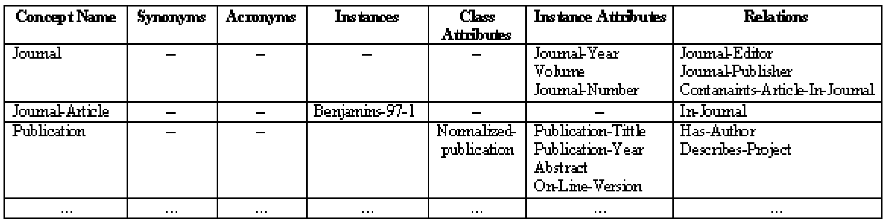
For *each* concept classification tree generated, we built the following IRs:

1. A *Concept Dictionary* containing all the domain concepts, instances of such concepts, class and instance attributes of the concepts, relations whose source is the concept and, optionally, concept synonyms and acronyms. Tables 2 and 3 show a small part of the concept dictionaries of the people and publication ontologies respectively.



Concept Name	Synonyms	Acronyms	Instances	Class Attributes	Instance Attributes	Relations
Academic-Staff	–	–	–	–		Supervises
Person					Age First-Name Last-Name Photo Weight ...	Has-Publication Editor-of Organizer-of-Chair-of Member-of-Program-Committe
PhD-Student	–	–	José López González	–	–	–
Researcher	–	–	Richard-Benjamins Asunción-Gómez-Pérez Dieter-Fensel ...	–	–	Cooperates-with Member-of-Research-Group Research-Interest
...	...	...	...	...	...	...

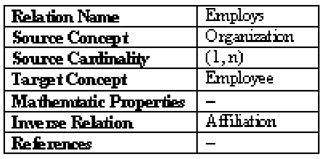
Table 2. **Concept Dictionary** in the domain of *People*



Concept Name	Synonyms	Acronyms	Instances	Class Attributes	Instance Attributes	Relations
Journal	–	–	–	–	Journal-Year Volume Journal-Number	Journal-Editor Journal-Publisher Contanaints-Article-In-Journal
Journal-Article	–	–	Benjamins-97-1	–	–	In-Journal
Publication	–	–		Normalized-publication	Publication-Tittle Publication-Year Abstract On-Line-Version	Has-Author Describes-Project
...	...	...	...	...	...	...

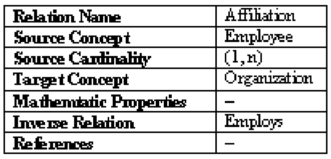
Table 3. **Concept Dictionary** in the domain of *Publication*

2. A *table of binary relations* for each "ad-hoc" relation whose source concept is in the concept classification tree. For each relation, we will specify: its name, the name of the source and target concept, its inverse relation, etc. Tables 4 and 5 show the *employs* relationship and its inverse *affiliation*.



Relation Name	Employs
Source Concept	Organization
Source Cardinality	(1, n)
Target Concept	Employee
Mathemtatic Properties	–
Inverse Relation	Affiliation
References	–

Table 4. **Binary Relation** *Employs*



Relation Name	Affiliation
Source Concept	Employee
Source Cardinality	(1, n)
Target Concept	Organization
Mathemtatic Properties	–
Inverse Relation	Employs
References	–

Table 5. **Binary Relation** *Affiliation*

3. An *Instance Attribute Table* for each instance attribute that appears in the concept dictionary. Instance attributes are attributes that are defined in the concept but that take values in its instances. For example, the weight of a person is proper to each instance. For each instance attribute, we will include: its name; the value type; the measurement unit for numerical values; accuracy for numerical values; range of values; default values; minimum and maximum cardinality; instance attributes, class attributes and constants that are used to infer the value of the attribute that is being defined; attributes that can be inferred using this attribute; a formula or rule for inferring the attribute that is being defined; and references used to fill in the attribute. First, table 6 presents an attribute

(photo) defined as a string. Second, table 7 shows an attribute (weight) whose value type is a mass-quantity (Gruber & Olsen 1994) and whose unit of measurement could be kilograms. This last term is in the standard units ontology available at the Ontology Server. Note that when a table field is filled in with a term belonging to another ontology, this ontology can be said to also be integrated.

Instance Attribute Name	Photo
Value Type	String
Unit of Measure	–
Precision	–
Range of Values	–
Default Value	–
Cardinality	(1, N)
Inferred from instance attribute	–
Inferred from class attribute	–
Inferred from constants	–
Formula	–
To infer	Beauty
References	–

Table 6. **Instance Attribute** *Photo*

Instance Attribute Name	Weight
Value Type	Mass-Quantity
Unit of Measure	Kilogram
Precision	0,001
Range of Values	[0, 200]
Default Value	–
Cardinality	(1, 1)
Inferred from instance attribute	–
Inferred from class attribute	–
Inferred from constants	–
Formula	–
To infer	–
References	–

Table 7. **Instance Attribute** *Weight*

4. A *Class Attribute Table* for each class attribute that appears in the concept dictionary. Class attributes describe concepts, not concept instances. Thus, for each class attribute, it will give: the name; possible value type; measurement unit for numerical values; value accuracy; attribute cardinality; instance attributes that could be inferred using the value of this attribute; and references, etc.

5. A *Logical Axioms Table.* This is used to define the concepts by means of logical expressions that are always true. Each axiom defined includes: its name, its natural language description, the concept to which the axiom refers, the attributes used in the axiom, the logical expression that formally describes the axiom using FOPC and references, as shown in tables 8 and 9.

Axiom Name	PhD-Student-Is-Not-In-Doctoral-Tesis-Jury
Description	"A PhD Student can not be in a Tesis Jury"
Concept	PhD-Student
Referred Attributes	–
Variables	S P
Expression	forall(S, P) PhD-Student(S) => not(Member-Of-Doctoral-Tesis-Jury(S, P))
Relations	–
References	–

Table 8. **Named Axiom**
PhD-Student-Is-Not-InDoctoral-Tesis-Jury

Axiom Name	The-Head-Of-Project- Works-In-The-Project
Description	"The employee that is the head of a project, also works in the project"
Concept	Employee
Referred Attributes	–
Variables	E P
Expression	forall(E, P) Employe(E) and Head-Of-Project(E, P) => Works-At-Project(E, P)
Relations	–
References	–

Table 9. **Named Axiom**
The-Head-of-Project-Works-on-at-Project

6. A *Constants Table.* For each constant, we will specify: its name, its natural language description, the value type (number, mass, etc.), the constant value, the measurement unit for numerical constants, the attributes that can be inferred using the constant and

references (see table 10).

Constant Name	Description	Value	Unit of Measurementon	To infer
Congress-Paper-Constant	It is a coefficient that is applied to obtain the standard publications of congress papers	0.75	–	Standard-Publications -rate
Workshop-Paper-Constant	It is a coefficient that is applied to obtain the standard publications of workshop papers	0.5	–	Standard-Publications -rate

Table 10. **Constants** *in the domain of Publication*

7. A *Formula Table* for each formula that appears in the instance attribute tables. These tables will be used to infer numerical instance attribute values from the values of other instance attributes, class attributes or even constants. Each table should specify information about the name of the formula, the attribute inferred with the formula, the mathematical expression of the formula; its natural language description; the instance and class attributes and constants used in the calculation; the accuracy with which the value will be calculated; the constraints under which it makes sense to use the formula; and the references employed in filling in the formula table. Table 11 shows the formula of *standard-publications-rate*.

Formula Name	Formula-of-Standard-Publications-rate
Concept	Person
Inferred attribute	Standard-Publications-rate
Formula	Standard-Publications-rate = Journal-Articles + Congress-Paper-Constant * Congress-Papers + Workshop-Paper-Constant * Workshop-Papers
Description	For standard publications, a journal-article has as value of 1, a congress paper of 0.75 and a workshop paper of 0.50
Basic Attributes	Journal-Articles Congress-Paper Workshop-Papers
Constants	Congress-Paper-Constant Workshop-Paper-Constant
Accuracy	0.01
Constraints	–
References	–

Table 11. **Formula** *of Normalized Publications*

8. *Attribute Classification Trees*, which graphically show related attributes and constants in a root attribute inference sequence, and the sequence of the formulas employed. Figure 6 summarizes the inference sequence of the attribute *standard-publications-rate*. We used this IR to validate that all attributes used in the formula make sense and no attributes have been omitted.

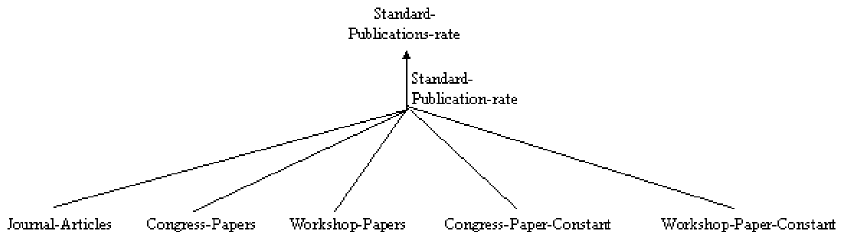
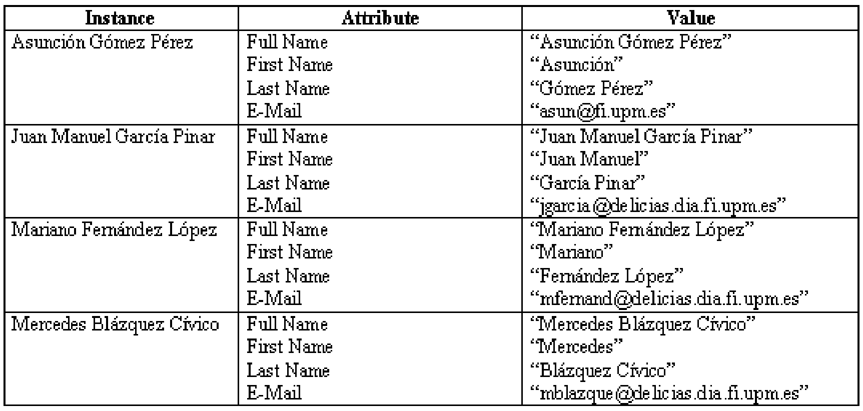


Figure 6. **Attributes Classification Tree** *in (KA)*^{2}

9. An *Instance Table* for each instance that appears in the concept dictionary. Each table will include: the name of the instance, the attributes with known values in the instance and the values of the above attributes. Table 12 presents instances in the People domain.



Instance	Attribute	Value
Asunción Gómez Pérez	Full Name First Name Last Name E-Mail	"Asunción Gómez Pérez" "Asunción" "Gómez Pérez" "asun@fi.upm.es"
Juan Manuel García Pinar	Full Name First Name Last Name E-Mail	"Juan Manuel García Pinar" "Juan Manuel" "García Pinar" "jgarcia@delicias.dia.fi.upm.es"
Mariano Fernández López	Full Name First Name Last Name E-Mail	"Mariano Fernández López" "Mariano" "Fernández López" "mfernand@delicias.dia.fi.upm.es"
Mercedes Blázquez Cívico	Full Name First Name Last Name E-Mail	"Mercedes Blázquez Cívico" "Mercedes" "Blázquez Cívico" "mblazque@delicias.dia.fi.upm.es"

Table 12. **Instances** *in the domain of People*

Note that the process of building these IRs is not sequential in the sense of a waterfall life-cycle model, but some order must be followed to assure the consistency and completeness of the knowledge already represented. However, our experience in the use of the model shows that the first table built is the glossary of terms. After this, we build concept classification trees and "ad-hoc" binary relations diagrams. Then, we proceed to build the concept dictionary, the "ad-hoc" binary relations tables, the class attributes and instance attributes and axiom tables.

Embedded in the conceptualization method itself, there are a series of controls for verifying that each IR is used correctly and that the knowledge represented is valid, that is, that the semantics of the conceptualized knowledge is what it should be. For a detailed description of the intermediate representations and individual and cross representation verification see (Gómez-Pérez, Fernández & deVicente 1996).

4. ODE

4.1. An overview

ODE (Ontology Design Environment) is the environment that enables *the development of ontologies at the knowledge level* using the approach proposed by METHONTOLOGY. The goal of ODE is to support the ontology maker during the entire life cycle of the ontology development process, from requirements specification, through the phases of knowledge acquisition and conceptualization, to implementation with as much integration and evaluation as possible. To reach this goal, it seeks to *automate each ontology development activity* and *automatically integrate the results of each phase with the input of the following phase.*

The functions included in the current version of ODE are: manage ontologies (including: open, close, save, save as, integrate, print, etc., an ontology); manage ontologies in tabular notation (create and remove table, add and delete table rows, print table, etc.); manage intermediate representations in graphic notation, automatically generate code in formal languages; customize the user interface; and manage conceptualization and user interface errors.

As the conceptualization of a complete and consistent ontology involves the management of a huge amount of information, the decision was *made to store the entire ontology in a relational database*. This has the big advantage that future applications will access the ontology easily using SQL queries because this database management and access language is widely used in practice which facilitates the creation of applications, networked or otherwise, that use our program.

In order *to maintain the consistency within each intermediate representation and between intermediate representations*, it was considered necessary to study the constraints in the fields of a conceptual table or between the fields belonging to different conceptual tables. This study was based on the rules of verification of the intermediate representations proposed in (Gómez-Pérez, Fernández & de Vicente 1996) and seeks to identify any effects possibly caused by the operations of editing an ontology at the conceptual level and how the above operations are implemented in the database.

As user experience in ontology development varies, ODE also provides an option *of guided conceptualization* that enables inexperienced users to learn about ontology development. This guided conceptualization, has proved very useful for domain experts who want to build their own ontologies.

It shuld be noted to say that ODE is being designed *as an environment that is completely independent of any system capable of managing ontologies*. That is, although ODE generates Ontolingua code in ASCII, ODE does not interact with Ontolingua.

Finally, the computational requirements to run ODE are: a Pentium with a Windows 95 operating system. ODE has been programmed in Visual Basic version 5.0. Its user interface complies with Microsoft design standards for the development of event-oriented applications and the ontology is stored in an Access (version 7.0) database.

4.2. ODE translator module

ODE' s big advantage is that it enables*specification of ontologies at the knowledge level, delegating implementation to fully automated code generators*. So, non-experts in the languages in which ontologies are implemented could specify ontologies using this environment.

In order to assure translation of the conceptual model to as many languages as possible, the translator module was designed using the principles of modularity and reusability. The translator module works on the following elements:

1. The conceptual model discussed in section 3 has been expressed declaratively using a grammar. So, on the basis of the following rules notation:

   - $A \rightarrow B$ means: A has the structure indicated in B;

   - [A]   means: A is optional;

- [A | B] means the same as: A or B;
- $\{A\}_x^y$ is the same as: A is repeated a number of times which is between x and y ; y
- terminal symbols are in **bold** and non-terminal symbols in *italics*;

the data dictionary is expressed as shown below:

```
data_dictionary →        Concept Name word
                         Synonyms     [{word}_0^n | --]
                         Acronyms     [{word}_0^n | --]
                         Instances    [{word}_0^n | --]
                         Class Attributes [{word}_0^n | --]
                         Instance Attributes [{word}_0^n | --]
                         Relations [{word}_0^n | --]
```

2. A series of patterns have been identified containing structures to be generated in each language. The patterns have been defined using the same notation as explained for the intermediate representations. For example, the pattern for defining classes in Ontolingua is defined, as follows:

```
def_clase →        {;;; class
                   (Define-Class class (?class)
                   "documentation"
                   [:def
                   (and
                         {(superclass ?class)}
                         [(Individual ?class)]
                         [(Superclass-Of {subclass})]
                         [(Has-Instance  ?class {instance})]
                         [{(Has-At-Most ?class relation max_cardinality)
                         (>=Value-Cardinality ?class relation min_cadinality) |
                         (Has-One ?class relation) |
                         (Has-Some ?class relation)]}_0^n)]
                   [:axiom-def (Exhaustive-Subclass-Partition class
                                         (Setof  {subclass}_1^n))])}_1^n
```

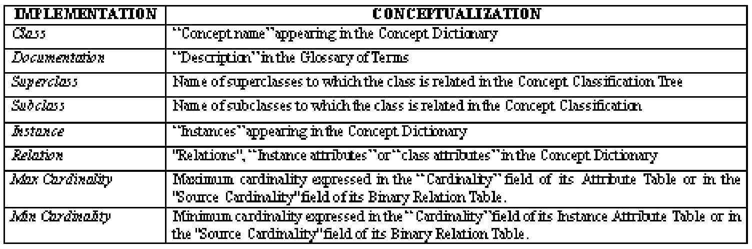
3. For each type of valid definition in the language, a table has been built that relates the terms used in the pattern to the terms employed in the conceptual model. So, the non-terminal symbols of the pattern will be placed in the left-hand column and the fields of the intermediate representations needed to get the information represented by the pattern will be placed in the right-hand column. Table 13 shows that the name of the class in the implementation matches a concept name in a concept dictionary (see table 2) of the conceptualization, that the documentation in the implementation is obtained from the description given in the glossary of terms (see table 1), that each superclass and subclass name is supplied by the concept classification tree (see figure 4), that the names of the instances are taken from the instance field defined in the concept dictionary (see table 2), that the relation names are obtained from the instance and class attributes defined for the concept in question in the concept dictionary (see table 2) and that the cardinalities appear in a binary relations table or in the instance or class attributes table. So, for the ontology

specified in section 3, the translator generated the following Ontolingua code:

```
;;; Academic-Staff

(Define-Class Academic-Staff (?Academic-Staff)
"He/She can be a lecturer or researcher. One of his/her possible duties is
  to supervise PhD Students"
:def
(and
        (Employee ?Academic-Staff)
        (>= Value-Cardinality ?Academic-Staff Supervises 0)
        (Superclass-Of Lecturer Researcher)))
```

Using the translator that converts the conceptualization into an implementation, error-free code can be generated, which dramatically cuts the time and effort involved in implementation. Furthermore, the architecture of this translator allows other translators to be developed in series. By merely changing the rules that identify the patterns of the terms to be generated and the second column of the table which relates the conceptualization to the implementation, a new translator can be built. This was how we built a translator for SFK in a short period of time.



IMPLEMENTATION	CONCEPTUALIZATION
Class	"Concept name" appearing in the Concept Dictionary
Documentation	"Description" in the Glossary of Terms
Superclass	Name of superclasses to which the class is related in the Concept Classification Tree
Subclass	Name of subclasses to which the class is related in the Concept Classification
Instance	"Instances" appearing in the Concept Dictionary
Relation	"Relations", "Instance attributes" or "class attributes" in the Concept Dictionary
Max Cardinality	Maximum cardinality expressed in the "Cardinality" field of its Attribute Table or in the "Source Cardinality" field of its Binary Relation Table.
Min Cardinality	Minimum cardinality expressed in the "Cardinality" field of its Instance Attribute Table or in the "Source Cardinality" field of its Binary Relation Table.

Table 13. **Relationship between conceptualization and implementation** *for classes*

5. CONCLUSIONS

This paper provides a series of guidelines for specifying ontologies at the knowledge level, as defined by Gruber (Gruber 1993), as an automatically generated explicit specification of a conceptualization. It also presents ODE as an environment for building ontologies during the entire ontology life cycle and implementing them automatically using translators.

METHONTOLOGY is a structured method for building ontologies at the knowledge level from scratch and when an ontological reengineering process is carried out. It seeks to build the ontology incrementally using a life cycle based on evolving prototypes. The methodology centers on the process of building conceptual models by defining concepts, instances and properties of the above concepts, organizing concepts in taxonomies, defining relations between taxonomies by means of "ad-hoc" relations, defining concept axioms and defining formulas. It also includes verification and validation of the ontology at the knowledge level.

ODE is a tool that automates ontology development activities and, therefore, seeks to automatically integrate the results of each phase. ODE uses a relational database to store the ontology and to assure consistency within and between IRs. Finally, the ontology can be constructed off-line and target code can be generated using ODE translators. It has a very intuitive user interface and it works on a Pentium.

Acknowledgements

We are grateful to the Knowledge Sharing and Reuse Group at the Laboratorio de Inteligencia Artificial of the UPM for their contribution to ODE design and implementation, and particularly to Julio Arpírez, Oscar Corcho and Javier Castillo.

REFERENCES

- Fernández, M.; Gómez-Pérez, A.; Juristo, N. (1997) *METHONTOLOGY: From Ontological Art Towards Ontological Engineering*. **Spring Symposium Series**. Stanford. PP: 33-40.
- Gómez-Pérez. A. (1998). *Knowledge Sharing and Reuse*. **The Handbook of Applied Expert Systems**. Edited by Liebowitz. CRC.
- Gómez-Pérez, A. (1996). *A framework to Verify Knowledge Sharing Tec hnology.* **Expert Systems with Application.** Vol. 11, N. 4. PP: 519-529.
- Gómez-Pérez, A.; Fernández, M.; de Vicente, A. (1996). *Towards a Method to Conceptualize Domain Ontologies.* **Workshop on Ontological Engineering.** ECAI' 96. Budapest. Hungary. PP: 41-52.
- Gruber, T. (1992). *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. **Technical Report KSL-93-04**. Knowledge Systems Laboratory. Stanford University, CA.
- Gruber, T. (1993). *A translation Approach to Portable Ontology Specifications.* **Knowledge Acquisition.** Vol5. 1993.
- Gruber, T.; Olsen, G. (1994). *An Ontology for Engineering Mathematics.* **Fourth International conference on Principles of Knowledge Representation and Reasoning**. Doyle, Torasso and Sandewall (Eds). Morgan Kaufmann. Also as KSL-94-18.
- Grüninger, M.; Fox, M.S. (1995). *Methodology for the Design and Evaluation of Ontologies.* **IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing**. Montreal, Quebec, Canada.
- Grüninger, M.; Fox, M.S. (1994). *The Role of Competency Questions in Enterprise Engineering*. IFIP WG 5.7. **Workshop on Benchmarking.** Theory and Practice. Trondheim, Norway.
- Kifer, M.; Lausen, G.; Wu, J. (1995). *Logical Foundations of Object-Oriented and Frame-Based Languages*. **Journal of the ACM**.
- Pazos J. (1995). **Conceptualización**. Máster en Ingeniería del Conocimiento. Facultad de Informática de Madrid. Universidad Politécnica de Madrid. SPAIN.
- Uschold, M.; Grüninger, M. (1996) *ONTOLOGIES: Principles, Methods and Applications.* **Knowledge Engineering Review.** Vol. 11; N. 2.