Building a Chemical Ontology Using Methontology and the Ontology Design Environment

Mariano Fernández López, Asunción Gómez-Pérez, and Juan Pazos Sierra, Polytechnic University of Madrid
Alejandro Pazos Sierra, University of Coruña

METHONTOLOGY PROVIDES GUIDELINES FOR SPECIFYING ONTOLOGIES AT THE KNOWLEDGE LEVEL, AS A SPECIFICATION OF A CONCEPTUALIZATION. ODE ENABLES ONTOLOGY CONSTRUCTION, COVERING THE ENTIRE LIFE CYCLE AND AUTOMATICALLY IMPLEMENTING ONTOLOGIES.

SOFTWARE AND KNOWLEDGE REuse have generated considerable interest because they reduce development time and the resources that projects require. For knowledge-based systems, in particular, the high cost of knowledge acquisition makes reuse essential. However, reuse involves these challenges: heterogeneity of representation formalisms, languages, and tools; lexical and semantic problems; assumptions implicit in each system; and commonsense-knowledge losses. Ontologies are a way around these obstacles. They are useful for unifying database, data-warehouse, and knowledge-base vocabularies and even for maintaining consistency when updating corporate memories used in knowledge management.

To meet the challenge of building ontologies, we've developed Methontology,[1,2] a framework for specifying ontologies at the knowledge level,[3] and the Ontology Development Environment. This article presents our experience in using Methontology and ODE to build the Chemicals ontology.[4]

The challenge of building ontologies

Ontology building is a craft rather than an engineering activity. Each development team usually follows its own set of principles, design criteria, and phases. The absence of structured guidelines and methods hinders the development of shared and consensual ontologies within and between teams, the extension of an ontology by others, and its reuse in other ontologies and final applications. We believe that the source of these problems is the absence of an explicit and fully documented conceptual model upon which to formalize the ontology.

Like knowledge-based-system development, ontology development faces a knowledge-acquisition bottleneck. Unlike KBS developers, ontology developers (ontologists) lack sufficiently tested and generalized methodologies recommending what activities should be performed and at what stage of ontology development. (For descriptions of related work, see the sidebar.)

Ontology developers often switch directly from knowledge acquisition to implementation, which poses these problems:

First, the conceptual models are implicit in the implementation codes. Making the conceptual models explicit usually requires reengineering.

Second, ontological commitments[5] and design criteria are implicit and explicit in the ontology code.

Third, domain experts and human end users have no understanding of formal ontologies codified in ontology languages.[6] Research has shown that, using the Ontology Server browser tools,[7] experts and users could gain a full understanding of and validate taxonomies and partially understand instances. However, they were unable to

1094-7167/99/$10.00 © 1999 IEEE

Methodologies for building ontologies

Until now, few domain-independent methodologies for building ontologies have been reported. Mike Uschold's methodology,[1] Michael Grüninger and Mark Fox's methodology,[2] and Methontology[3,4] are the most representative. These methodologies all start from the identification of the ontology's purpose and the need for domain knowledge acquisition. However, having acquired a significant amount of knowledge, Uschold's methodology and Grüninger and Fox's methodology propose coding in a formal language and Methontology proposes expressing the idea as a set of *intermediate representations* (see the main article). Methontology then uses translators to generate the ontology.

These three methodologies also identify the need for ontology evaluation. Uschold's methodology includes this activity but does not state how to carry it out. Grüninger and Fox propose identifying a set of competency questions.[2] Competency questions are the basis for a rigorous characterization of the knowledge that the ontology has to cover; they specify the problem and what constitutes a good solution. Once the ontology has been expressed formally, it is compared against this set of competency questions. Methontology proposes that evaluation occur throughout ontology development. Most of the evaluation happens during conceptualization.

Several representation systems use a frame-based modeling approach, a logic-based approach, or even both to formalize ontologies. They model the world using concepts, instances, relations, functions, and axioms. An ontology formalized using any of these approaches can then be implemented, more or less straightforwardly, in different languages such as Ontolingua,[5] CycL,[6] LOOM,[7] and Flogic.[8]

References

1. M. Uschold and M. Grüninger, "ONTOLOGIES: Principles, Methods and Applications," *Knowledge Eng. Rev.*, Vol. 11, No. 2, June 1996.
2. M. Grüninger and M.S. Fox, "Methodology for the Design and Evaluation of Ontologies," *Proc. Int'l Joint Conf. AI Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.
3. M. Fernández, A. Gómez-Pérez, and N. Juristo, "METHONTOLOGY: From Ontological Art towards Ontological Engineering," *Proc. AAAI Spring Symp. Series*, AAAI Press, Menlo Park, Calif., 1997, pp. 33–40.
4. A. Gómez-Pérez, "Knowledge Sharing and Reuse," *The Handbook of Applied Expert Systems*, J. Liebowitz, ed., CRC Press, Boca Raton, Fla., 1998.
5. A. Farqhuar et al., *Collaborative Ontology Construction for Information Integration*, Tech. Report KSL-95-69, Knowledge Systems Laboratory, Stanford Univ., Stanford, Calif., 1995; http://ksl-web.stanford.edu/publications.
6. B.D. Lenat, "Steps to Sharing Knowledge," *Towards Very Large Knowledge Bases*, N.J.I. Mars, ed., IOS Press, Amsterdam, 1995, pp. 3–6.
7. *Loom Users Guide Version 1.4*, ISX Corp., Westlake Village, Calif., 1991; http://www.isi.edu/isd/LOOM/documentation/LOOM-DOCS.html.
8. M. Kifer, G. Lausen, and J. Wu, "Logical Foundations of Object-Oriented and Frame-Based Languages," *J. ACM*, Vol. 42, No. 4, July 1995, pp. 741–843.

understand abstract definitions of concepts, relations, functions, and axioms. From the knowledge-acquisition viewpoint, they were quite unable to formalize their knowledge.

Fourth, as with traditional knowledge bases, direct coding of the knowledge-acquisition result is too abrupt a step, especially for complex ontologies.

Fifth, 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 Tom Gruber:[5]

> The conceptualization should be specified at the knowledge level without depending on a particular symbol level encoding. An encoding bias results when representation choices are made purely for the convenience of notation or implementation ….

Finally, ontology developers might have difficulty understanding implemented ontologies or even building new ontologies. This is because traditional ontology tools focus too much on implementation issues rather than on design questions. For example, someone who knows how to build ontologies but is unfamiliar with the language in question might have difficulties working at the implementation level.

For instance, the expression *density = mass*/*volume* in a chemical domain could be written in Ontolingua as shown in Figure 1. This example shows that unless you are very familiar with the language, understanding existing definitions and writing new definitions are almost impossible. If you are successful in this process, it will have taken a big effort. The problem is not understanding that density is equal to mass divided by volume at the knowledge level, but writing this in a target language. Therefore, something that is apparently very simple at the conceptual level is extremely complicated when expressed at the implementation level, if you're not familiar with the language.

This means that ontologies are built exclusively by developers who are perfectly acquainted with the languages in which the ontologies will be implemented. Because these ontologists are not necessarily experts in the domain for which the ontology is built, they spend a lot of time and resources on knowledge acquisition.

Methontology and ODE alleviate some of these problems. Methontology provides a user-friendly approach to knowledge acquisition by non-knowledge engineers,[6] and an effective, generally applicable method for domain-knowledge-model construction and validation. ODE supports this framework by letting ontologies be built at the knowledge level and implemented automatically using translators. So, ontologists don't need to know the ontology's implementation language.

Ontology development

Ontological engineering requires the definition and standardization of a life cycle ranging from requirements specification to maintenance, as well as methodologies and techniques that drive ontology development. So, the Methontology framework includes

- the identification of the ontology development process;
- a life cycle based on evolving prototypes; and
- the methodology itself, which specifies the steps for performing each activity, the techniques used, the products to be output, and how the ontologies are to be evaluated.

In developing Chemicals, our first step, as in any project, was to plan. Then, because we're not experts in the chemicals domain, we acquired knowledge to put together a preliminary version of the requirements specification. We then simultaneously acquired and conceptualized more knowledge; conceptualization helped guide acquisition.

```
(Define-Function Density(?Element):->?Density-At-20-C
"The density of an element is equal to its atomic weight divided 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-Volume-At-20-C 0)
                          (/?Atomic-Weight?Atomic-Volume-At-20-C))))))
```

Figure 1. The expression *density = mass/ volume* in a chemical domain written in Ontolingua.

Indeed, the conceptualization phase is like assembling a jigsaw puzzle from the pieces supplied by acquisition, which is why most knowledge acquisition is completed during conceptualization. We then gave the ontology to the expert for examination. The expert evaluated the conceptualization by interpreting the *intermediate representations* (which we'll present in the next section), which are fairly intuitive. During both specification and conceptualization, we integrated in-house and external ontologies. Once the conceptualization was complete, we used ODE to automatically generate the code in Ontolingua.

Our experience shows that the specification, conceptualization, integration, and implementation can be performed as often as required. Indeed, an ontology's specification frequently changes throughout the ontology life cycle as its definitions are created, modified, and deleted. Figure 2 presents the development life cycle of the Chemicals ontology.

The Chemicals ontology (available at *http://www-ksl.stanford.edu:5915* and *http://www-ksl-svc-lia.dia.fi.upm.es:5915*.) actually comprises two main ontologies: `Chemical-Elements` and `Chemical-Crystals`. `Chemical-Elements` has 16 concepts, 103 instances, three functions, 21 relations, and 27 axioms. `Chemical-Crystals` has 19 concepts, 66 instances, eight relations, and 26 axioms. Chemicals also includes public Ontolingua ontologies, such as `Standard-Units`, `Standard-Dimensions`, and `KIF-Lists`.

Specification. Ontology specification's goal is to put together a document that covers the ontology's primary objective, purpose, granularity level, and scope. The aim is to identify the set of terms to be represented, their characteristics, and their granularity. This specification should be as complete and concise as possible.

Figure 3 shows a short example of an ontology requirements specification document in the chemicals domain. We built this specification after acquiring domain knowledge.

Conceptualization. When most of the knowledge has been acquired, the ontologist has a lot of unstructured knowledge that must be organized. Conceptualization organizes and structures the acquired knowledge using external representations that are independent of the implementation languages and environments. Specifically, this phase organizes and converts an informally perceived view of a domain into a semiformal specification, using a set of intermediate representations that the domain expert and ontologist can understand.[6] These IRs bridge the gap between how people think about a domain and the languages in which ontologies are formalized.

This set of IRs is based on those used in the conceptualization phase of the Ideal methodology for knowledge-based systems development.[8] Figure 4 illustrates the order we follow for conceptualization.

First, we built a *glossary of terms* that includes all the terms (concepts, instances, attributes, verbs, and so on) of the chemicals domain and their descriptions (see Figure 5a).

When the glossary contained a sizable number of terms, we built *concept-classification trees* using relations such as subclass-of, subclass-partition-of, and exhaustive-subclass-of. (Class C is a subclass of parent class P if and only if every instance of C is also an instance of P. A subclass partition of C is a set of subclasses of C that are mutually disjoint. A exhaustive subclass of C is a set of mutually-

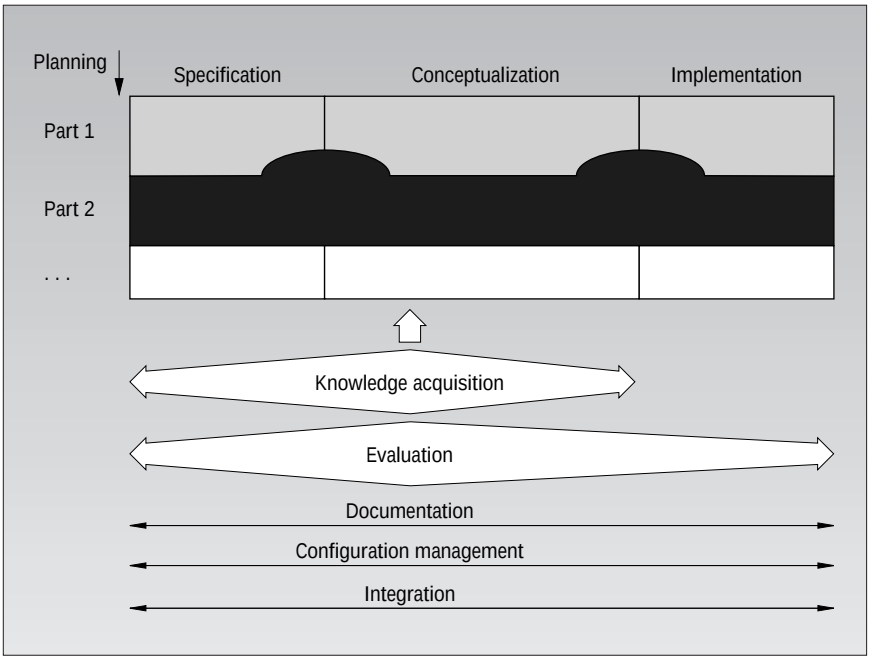


Figure 2. Ontology development life cycle.

disjoint classes—a subclass partition—that covers C. Every instance of C is an instance of exactly one of the subclasses in the partition.) So, we identified this domain's taxonomies, and each taxonomy produced an ontology as prescribed by Methontology. The concept-classification tree in Figure 5b outlines the taxonomy of the `Chemical-Elements` ontology.

The next step was to build ad hoc *binary-relations diagrams* between concept classification trees. These diagrams establish relationships between concepts of the same or different ontologies. They will set out the guidelines for integrating ontologies, because if a concept C1 is linked by a relation R to a concept C2, the ontology containing C1 includes the ontology containing C2, provided that C1 and C2 are in different con-

Domain: Chemical
Date: May 15, 1996
Developed by Asunción Gómez-Pérez and Mariano Fernández López

Purpose: Ontology about chemical substances to be used when information about chemical elements is required in teaching, manufacturing, analysis, and so on.

Level of formality: Semiformal.

Scope: List of 103 elements: *lithium*, *sodium*, *chlorine*, *mercury*, ….
List of concepts: *element*, *halogen*, *noble gas*, *semimetal*, *metal*, *third-transition metal*, ….
Information about at least the following properties: *atomic number*, *atomic weight*, *electronegativity*, *melting point*, ….

Sources of knowledge:
(a) Three interviews with the expert.
(b) The following books:
[Handbook, 84–85] *Handbook of Chemistry and Physics*, 65th ed., CRC Press Inc., 1984–1985.

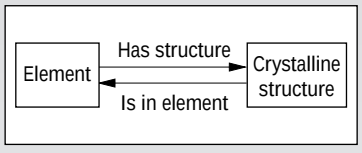
Figure 3. Ontology requirements specification document for the Chemicals ontology.

Name	Description
Atomic weight	The relative mass of an atom of an element against the mass of the twelfth part of the carbon-12 isotope.
Crystalline structure	A 3D skeleton containing the set of points or ionic positions of a crystal that have an identical environment.
Element	A substance made up of atoms with the same number of protons.
Mercury	(Planet Mercury), Hg (hydrargyrum, liquid silver); at. wt. 200.59 ± 3; at. no. 80; m.p. —38.842°C; b.p. 356.58°C; valence 1 or 2. The metal is obtained by heating cinnabar in a current of air and by condensing the vapor. It is a heavy, silvery-white metal; a rather poor conductor of heat, as compared with other metals; and a fair conductor of electricty. [Handbook, 84–85]
Third transition series	The set of elements that belong to Ib, IIb, IIIb, IVb, Vb, VIb, VIIb, and VIII groups of the sixth period.

(a)

Element
  Reactiveness
    Nonmetal
      Halogen
    Semimetal
    Metal
      Transition metal
        First transition series
        Second transition series
        Third transition series
          Lanthanide
        Actinide
      Nontransition metal
        Alkalai
        Alkalai terreum

(b)



(c)

Concept name	Synonyms	Acronyms	Instances	Class attributes	Instance attributes	Relations
Element	—	Elm.	—	—	Atomic number Atomic volume at 20°C Atomic weight Chemical group Chemical period Density at 20°C Electronegativity Melting point …	Has structure
Third transition series	Sixth-period transition series	3TS.	Gold Hafnium Mercury Osmium Iridium Platinum Rhenium Tantalum Wolfram	—	—	—

(d)

Figure 5. Intermediate representations for the Chemicals ontology: (a) part of the glossary of terms; (b) a concept-classification tree; (c) a binary-relations diagram; (d) part of the concept dictionary; (e) a binary-relations table; (f) an instance-attribute table; (g) a logical-axioms table; (h) a formula table; (i) an attribute-classification tree; (j) part of an instance table.

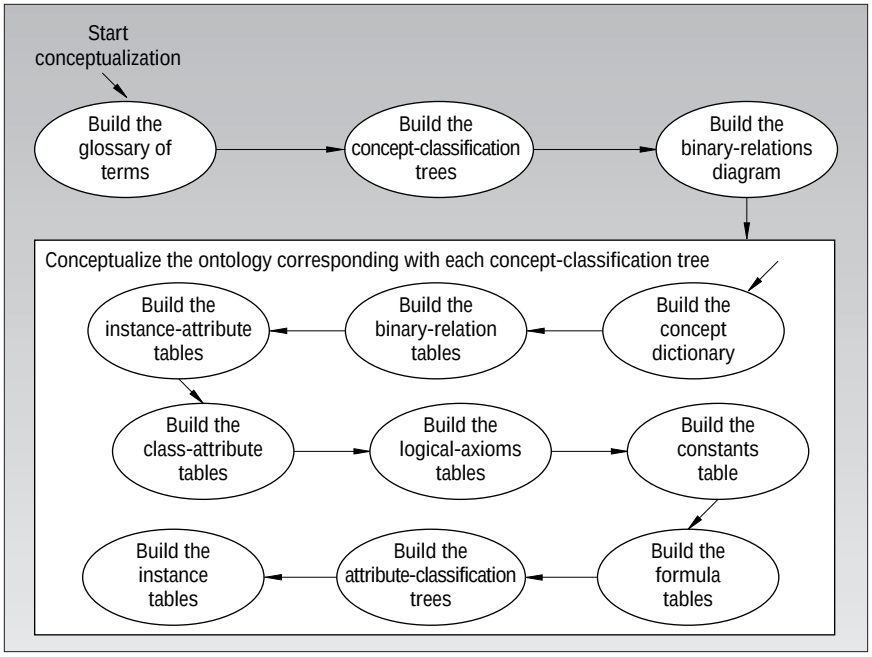


Figure 4. Conceptualization according to Methontology.

cept-classification trees. Figure 5c presents a simplified diagram of the ad hoc binary relations in the Chemicals ontology. The `Chemical-Elements` ontology includes the `Chemical-Crystal` ontology because the source concept of the relations named `Has-Structure` is in the `Chemical-Elements` hierarchy. Similarly, if the relation `Is-in-Element` is defined as the inverse relation of `Has-Structure`, the `Chemical-Crystal` ontology can be said to include the `Chemical-Elements` ontology.

For each concept-classification tree generated, we built these IRs:

The *concept dictionary* contains all the domain concepts, instances of such concepts, class and instance attributes of the concepts, and, optionally, concept synonyms

Relation name	Has structure
Source concept	Element
Source cardinality	(0, *n*)
Target concept	Crystalline structure
Target cardinality	(0, *n*)
Mathematic properties	—
Inverse relation	Is in element
References	[Cullity, 78]

(e)

Instance attribute name	Density-at-20°C
Value type	Density-Quantity
Unit of measure	Kilogram/meter^3
Precision	0.001
Range of values	[0, 25]
Default value	—
Cardinality	(1, *n*)
Inferred from instance attribute	Atomic-Weight Atomic-Volume
Inferred from class attribute	—
Inferred from constants	—
Formula	Density
To infer	—
References	—

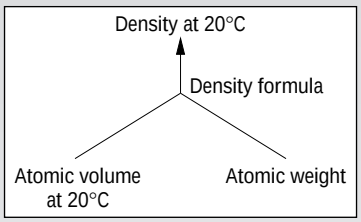
(f)

Axiom name	High electronegativity of nonmetals
Description	Electronegativity of nonmetals is higher than 2.1
Concept	Nonmetal
Referred attributes	Electronegativity
Variables	*N*, *E*
Expression	For all (*X*, *Y*) (Non-Metal(*X*) and Electronegativity(*X*, *Y*) => *Y* > 2.1)
Relations	—
References	[Janssen, 90]

(g)

Formula name	Density
Inferred attribute	Density-at-20-Degrees-Celsius
Formula	Density-at-20-Degrees-Celsius = Atomic-Weight/Atomic-Volume-at-20-Degrees-Celsius
Description	An element's density is equal to its atomic weight divided by its atomic volume
Basic instance attributes	Atomic-Weight Atomic-Volume-at-20-Degrees-Celsius
Basic class attributes	—
Constants	—
Precision	—
Constraints	Atomic-Volume-at-20-Degrees-Celsius > 0

(h)



(i)

Instance	Attribute	Value
Mercury	Atomic number	80
	Atomic weight	200.59
	Density at 20 degrees Celsius	13.546
	Electronegativity	1.9
	Melting point	–38.842

(j)

and acronyms. Figure 5d gives a small part of the concept dictionary of `Chemical-Elements`.

A *binary-relations table* specifies the name, the names of the source and target concept, the inverse relation, and so on for each ad hoc relation whose source concept is in the concept-classification tree (see Figure 5e).

An *instance-attribute table* describes each instance attribute in the concept dictionary (see Figure 5f). Instance attributes are attributes that are defined in the concept but that take values in its instances. For example, a chemical element's atomic weight is proper to each instance. For each instance attribute, we included

- its name;
- the value type;
- the measurement unit for numerical values;
- the accuracy for numerical values;
- the range of values;
- the default values;
- minimum and maximum cardinality;
- the instance attributes, class attributes, and constants that are used to infer the value of the attribute that is being defined;
- the attributes that can be inferred using this attribute;
- the formula or rule for inferring the attribute that is being defined; and
- the references used to fill in the attribute.

A *class-attribute table* describes concepts, not concept instances, for each class attribute in the concept dictionary. So, for each class attribute, the table will give the name, possible value type, measurement unit for numerical values, value accuracy, attribute cardinality, instance attributes for which a value can be inferred using the value of this attribute, references, and so on.

A *logical-axioms table* defines the concepts by means of logical expressions that are always true (see Figure 5g). Each defined axiom 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 (first-order predicate calculus), and references.

A *constants table* specifies each constant's name, its natural-language description, its value type (number, mass, and so on), its constant value, its measurement unit (for numerical constants), the attributes that can be inferred using the constant, and references.

A *formula table* describes each formula in the instance-attribute tables (see Figure 5h). We used these tables to infer numerical instance-attribute values from the values taken by other instance attributes, class attributes, or even constants. Each table should specify

- the formula's name,
- the attributes inferred with the formula,
- the formula's mathematical expression,
- 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 using the formula makes sense, and
- the references employed in filling in the formula table.

THE PROCESS OF BUILDING THESE IRS IS NOT SEQUENTIAL IN THE SENSE OF A WATERFALL LIFE-CYCLE MODEL, BUT IT MUST FOLLOW SOME ORDER SO AS TO ASSURE THE CONSISTENCY AND COMPLETENESS OF THE REPRESENTED KNOWLEDGE.

An attribute-classification tree graphically shows related attributes and constants in a root attribute inference sequence, and the sequence of the formulas employed (see Figure 5i). We used it to validate that all attributes used in the formula make sense and no attributes have been omitted.

An *instance table* lists the name, the attributes with known values in a instance, and the values of the above attributes, for each instance in the concept dictionary (see Figure 5j).

The process of building these IRs is not sequential in the sense of a waterfall life-cycle model, but it must follow some order so as to assure the consistency and completeness of the represented knowledge. Embedded in the conceptualization method 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. Asunción Gómez-Pérez, Mariano Fernández López, and Antonio de Vicente have provided a detailed description of the IR evaluation.[9]

Our experience shows that domain experts and human end users understand and validate most of the Methontology IRs. In one set of trials, two environmental and chemical experts understood and validated 80% of the knowledge represented in the IRs. Also, we found that, from the knowledge-acquisition point of view, experts can fill in many of the IRs.

Knowledge acquisition. This is an independent activity within ontology development. However, it coincides with other activities. For Chemicals, we acquired most knowledge at the start of ontology development. The level of knowledge acquisition fell as development progressed and we became more familiar with the application domain.

Knowledge acquisition occurred in three stages:

(1) We held preliminary meetings with the expert to look for general, not detailed, knowledge. The depth of these meetings was minimal; we were looking for the coarse grain—the overview.
(2) We studied the documentation. We needed to learn as much as possible about the domain of expertise (chemistry), to save the time the expert would otherwise have spent on instructing us in the domain. That's why having a method with which experts can build their own ontologies is very useful.
(3) Having obtained some basic knowledge, we initiated the expert-knowledge-acquisition cycle. We started by looking for more general knowledge and gradually moved down into the particular details for configuring the full ontology.

During knowledge acquisition, we used the following set of KBS knowledge-acquisition techniques in an integrated manner:

- *Nonstructured interviews with experts* to build a preliminary draft of the requirements-specification document. They output terms, definitions, a concept classification, and so on.
- *Informal text analysis* to study the main concepts in books and handbooks. This

study enabled us to fill in IRs such as the concept dictionary.

- *Formal text analysis*. We performed this manually without using specialized environments for this purpose. First, we identified the patterns to be located in the text. For example, for Chemicals, some of the structures to be detected were "A is B," "the highest value of A is attained in B," and "A increases as B increases." Afterward, we selected the instantiated structures in the text that matched the patterns. Finally, we analyzed the marked sentences to extract attributes, natural-language definitions, rules, assignation of values to attributes, and so on.
- *Structured interviews with the expert* to get specific and detailed knowledge about concepts, their properties, and their relationships with other concepts, and to evaluate the conceptual model once the conceptualization activity is complete. For Chemicals, because the expert was familiar with the IRs, we used structured interviews for validation purposes. We delivered the IRs to the expert, who proposed changes.
- *Domain table analysis*. This was very useful for ascertaining the values of the concept attributes and for identifying certain data regularities. For example, we could get electronegativity attribute values for each element from the electronegativities table—for instance, checking that the electronegativity of `nonmetals` is over 2.1. However, this technique can be risky when data are taken from domain tables whose sources differ, because they are probably based on different criteria. Electronegativity, for example, can be obtained according to the criterion of Linus Pauling or Robert Mulliken.
- *Domain-graph analysis* to look for regularities. For example, using a graph that related atomic numbers to volumes, we concluded that the atomic volume of all the lanthanides is similar. If we had had domain graphs with exact data, we could have obtained information similar to that output by domain-table analysis.
- *Units-of-measurement analysis* to determine the attributes involved in formulas and to determine the quantities of these attributes.
- *Detailed reviews by the expert*. When conceptualization was at an advanced stage, we submitted the IRs to the expert for detailed inspection. After several days working on his own, he returned the tables and trees with a series of suggestions and corrections.
- *Formula analysis* to check whether the formulas are correct and to determine their attributes. For example, for `Chemical-Elements`, the expert originally expressed the density formula as *density* = *mass*/*volume*. The expert recommended that *mass* was the `Atomic-Weight`; *volume*, the `Atomic-Volume` at a given temperature (specifically, 20°C); and *density*, the element's `Density` at that temperature. Afterward, we checked the formula for correctness, applying the formula to several cases in which the three data were known. We then analyzed the units of measurement.

THE USER OR APPLICATION THAT ACCESSES THE ONTOLOGY ONLY HAS TO KNOW THE IRs THAT ARE USED FOR CONCEPTUALIZATION AND NOT HOW THE CONCEPTUALIZATION IS STORED IN THE DATABASE.

The expert was very important in all these techniques (although he was not always directly involved), because he gave clues as to what we were to look for. For example, thanks to the expert, we knew that the `Lanthanides` have properties very similar to those of `Lanthanum`. Therefore, we always checked in the tables and the graphs whether it was possible to affirm in the ontology that the maximum and minimum value within the lanthanides was very close.

Integration. Throughout ontology development, we identified terms that could be included from other ontologies. For the Chemicals ontologies, we found these possibilities for reuse of ontologies stored in the Ontology Server: `KIF-Number` supplied all the mathematical operators (multiplication, sum, and so on). `KIF-List` contained the definitions for building lists. `Standard-Dimensions`[10] included terms such as `mass-quantity` and `length-quantity`, which define an attribute that stores a value that is a mass or length. Finally, `Standard Units` defined the SI (Système International) units of measurement. It provided some, but not all, of the units of measurement identified in the conceptualization.

Having identified candidate ontologies for reuse, we checked that these ontologies had been validated and verified. Because no software environment for validating ontologies exists, we did this manually, following the guidelines given by Asunción Gómez-Pérez.[11]

Implementation. Finally, ODE automatically generated Ontolingua language code using a translator that transforms the conceptual model into an implemented model. We'll describe this in the next section.

ODE

The Ontology Design Environment's goal is to support the ontologist throughout ontology development, from requirements specification, through knowledge acquisition and conceptualization, to implementation, with as much integration and evaluation as possible. To achieve this goal, ODE seeks to automate each ontology-development activity and automatically integrate the results of each phase with the input of the following phase.

ODE capabilities. Because the conceptualization of a complete and consistent ontology involves the management of a huge amount of information, we stored the entire ontology in a relational database. This has the big advantage that the applications access the ontology using SQL (Structured Query Language), which encourages the use of ontologies in real applications. ODE uses SQL directly on the IRs and not on the database internal structure. Therefore, the user or application that accesses the ontology only has to know the IRs that are used for conceptualization and not how the conceptualization is stored in the database.

To maintain the consistency within each IR and between IRs, we studied the constraints in the fields of a conceptual table or between the fields belonging to different conceptual tables. This study, based on proposed rules of verification of the IRs,[9] 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.

Because user experience in ontology de-

data_dictionary → **Concept name** *word*
Synonyms [{*word*}$_1^n$ | —]
Acronyms [{*word*}$_1^n$ | —]
Instances [{*word*}$_1^n$ | —]
Class attributes [{*word*}$_1^n$ | —]
Instance attributes [{*word*}$_1^n$ | —]
Relations [{*word*}$_1^n$ | —]

Figure 6. The expression of the data dictionary in the ODE rules notation.

def_class → {;;; *class*
(**Define-Class** *class* (?*class*)
"*documentation*"
[**:def**
(**and**
{(*superclass* ?*class*)}
[(**Superclass-Of** {*subclass*})]
[(**Has-Instance** ?*class* {*instance*})]
[{(**Has-at-Most** *relation* ?*class max_cardinality*)
(**Has-at-Least** *relation* ?*class min_cardinality*) |
(**Has-One** ?*class relation*) |
(**Has-Some** ?*class relation*)]}$_0^n$)(
[**:axiom-def (Exhaustive-Subclass-Partition** *class*
(**Setof** {*subclass*}$_1^n$))])}$_1^n$

Figure 7. The transformation rule for defining classes in Ontolingua.

```
;;; Element

(Define-Class Element (?Element)
"A substance that is made up of atoms with the same number of
protons."
:def
(and
      (Has-One ?Element Atomic-Number)
      (Has-Some ?Element Atomic-Volume-at-20-Degrees-Celsius)
      (Has-One ?Element Atomic-Weight)
      (Has-One ?Element Chemical-Group)
      (Has-One ?Element Chemical-Period)
      (Has-Some ?Element Density-at-20-Degrees-Celsius)
      (Has-One ?Element Electronegativity)
      (Has-Some ?Element Melting-Point)
.....)
:axiom-def
      (Exhaustive-Subclass-Partition Element
      (Setof Reactiveness Reactiveless)))
```

Figure 8. Ontolingua code generated by the ODE translator for the concept Element.

velopment varies, ODE provides optional guided conceptualization that helps inexperienced users learn about ontology development. This conceptualization is very useful for domain experts who want to build their own ontologies.

ODE currently includes these functions:

- Managing ontologies (opening, closing, saving, saving as, integrating, printing, and so on);
- Managing ontologies in tabular notation (creating and removing tables, adding and deleting table rows, printing tables, and so on);
- Managing IRs in graphic notation;
- Automatically generating code in formal languages;
- Customizing the user interface;
- Managing the development-environment help; and
- Managing conceptualization and user interface errors.

We've designed ODE to be completely independent of other systems that manage ontologies. For example, although ODE generates Ontolingua code in ASCII, ODE does not interact with Ontolingua.

ODE requires a Pentium running Windows 95 and Access 7. 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.

The translator module. Because ODE delegates implementation to fully automated code generators, nonexperts in the languages in which ontologies are implemented can specify ontologies using the IRs already presented. To assure translation of the conceptual model to as many languages as possible, we designed ODE's translator to be modular and reusable. The translator module incorporates these elements:

First, we use a grammar to declaratively express the conceptual model. We use this rules notation:

- A → B means *A has the structure indicated in B*;
- [A] means *A is optional*;
- [A | B] means *A or B*;
- {A}$_x^y$ means *A is repeated a number of times that is between x and y;*
- Terminal symbols are in **bold** and nonterminal symbols are in *italics*;
- — means *the field is filled in with no value.*

Figure 6 shows how we express the data dictionary.

Second, we identified a series of transformation rules containing the structures to be generated in each language. We use the same notation as for the IRs. For example, Figure 7 shows the transformation rule for defining classes in Ontolingua.

Third, for each type of valid definition in the language, we built a table that relates the terms used in the transformation rules to the terms employed in the conceptual model (see Table 1). So, the nonterminal symbols of the transformation rules go in the left-hand column; the fields of the IRs needed to get the information represented by the transformation rule go in the right-hand column. Table 1 shows that

- the name of the class in the implementation matches a concept name in the concept dictionary;
- the documentation in the implementation is obtained from the description in the glossary of terms;

Table 1. The relationship between conceptualization and implementation for classes.

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 tree
Instance	Instances appearing in the concept dictionary
Relation	Instance attributes, class attributes, or relations in the concept dictionary
Max cardinality	Maximum cardinality expressed for the attribute in the Cardinality field of its instance-attribute table or binary-relation table
Min cardinality	Minimum cardinality expressed for the attribute in the Cardinality field of its instance-attribute table or binary-relations table

- each superclass and subclass name is supplied by the concept-classification tree;
- the names of the instances are taken from the instance field defined in the concept dictionary;
- the relation names are obtained from the instance and class attributes and the relations defined for the concept in question in the concept dictionary; and
- the cardinalities appear in the instance- or class-attribute tables and ad hoc binary-relation tables.

Figure 8 shows an example of the Ontolingua code generated for the Chemicals ontology. The generated code is error-free, which dramatically cuts the time and effort involved in implementation.

Furthermore, the translator's architecture allows other translators to be developed in series. By merely changing the rules that identify the transformation rules of the terms to be generated and changing the second column of the table that relates the conceptualization to the implementation, we can build a new translator.

METHONTOLOGY'S TABULAR and graphic-based notation is a user-friendly approach to knowledge acquisition by domain experts who are not knowledge engineers. So, all the ontologies built using this approach are not hand-crafted; they rely on the same conceptualization process; they have been built independently of their end use; and the final ontology code is generated automatically using ODE translators.

The Chemicals ontology is being used in several applications. Ontogeneration[6] is an information-retrieval system that lets Spanish users consult and access, in their own language, the knowledge contained in the Chemicals ontology. The system uses a domain ontology (Chemicals) and a linguistic ontology (the Generalized Upper Model[12]) to generate Spanish text descriptions in response to the queries in the chemistry domain.

The other application is Chemical OntoAgent,[13] an ontology-based WWW broker in the chemistry domain. It is a teaching broker that lets students learn chemistry in a very straightforward manner, providing the necessary domain knowledge and helping students test their skills. To make the answers more understandable to students, this system can interact with Ontogeneration.

We are building other ontologies according to the Methontology framework and using ODE: the `(KA)2-Ontology`, developed by the Knowledge Annotation Initiative,[14] which seeks to model the knowledge-acquisition community (its researchers, topics, products, and so on); the `Reference-Ontology`,[13] a domain ontology about ontologies; and a `Monatomic-Ions-Ontology`, to be included in an `Environmental-Pollutants-Ontology`. These two ontologies reuse the Chemicals ontology.

Acknowledgments

This work has been partially supported by the program Ayudas de I&D para grupos potencialmente competitivos of the Polytechnic University of Madrid (Reference A9706).

References

1. M. Fernández, A. Gómez-Pérez, and N. Juristo, "METHONTOLOGY: From Ontological Art towards Ontological Engineering," *Proc. AAAI Spring Symp. Series*, AAAI Press, Menlo Park, Calif., 1997, pp. 33–40.
2. A. Gómez-Pérez, "Knowledge Sharing and Reuse," *The Handbook of Applied Expert Systems*, J. Liebowitz, ed., CRC Press, Boca Raton, Fla., 1998.
3. A. Newell, "The Knowledge Level," *Artificial Intelligence*, Vol. 18, No. 1, Jan. 1982, pp. 87–127.
4. M. Fernández, *Chemicals: Ontología de Elementos Químicos* (Ontology of Chemical Elements), final-year project, Facultad de Informática de la Universidad Politécnica de Madrid, 1996.
5. T.R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," *Proc. Int'l Workshop on Formal Ontology*, 1992. Also available as Tech. Report KSL-93-04, Knowledge Systems Laboratory, Stanford Univ., Stanford, Calif., 1993; http://ksl-web.stanford.edu/publications.
6. G. Aguado et al., "Ontogeneration: Reusing Domain and Linguistic Ontologies for Spanish Text Generation," *Workshop on Applications of Ontologies and Problem Solving Methods* (part of ECAI '98: 1996 European Conf. AI), European Coordinating Committee for Artificial Intelligence, 1998, pp. 1–10.
7. A. Farqhuar et al., *Collaborative Ontology Construction for Information Integration*, Tech. Report KSL-95-69, Knowledge Systems Laboratory, Stanford Univ., 1995; http://ksl-web.stanford.edu/publications.
8. A. Gómez-Pérez et al., *Ingeniería del Conocimiento: Construcción de Sistemas Experto* (Knowledge Engineering: The Construction of Expert Systems), Editorial Centro de Estudios Ramón Areces, Madrid, 1997.
9. A. Gómez-Pérez, M. Fernández, and A. De Vicente, "Towards a Method to Conceptualize Domain Ontologies," *Workshop on Ontological Eng.* (part of ECAI '96: 1996 European Conf. AI), European Coordinating Committee for Artificial Intelligence, 1996, pp. 41–52.
10. T. Gruber and G. Olsen, "An Ontology for Engineering Mathematics," *Proc. Fourth Int'l Conf. Principles of Knowledge Representation and Reasoning*, Morgan-Kaufmann, San Francisco, 1994. Also available as Tech. Report KSL-94-18, Knowledge Systems Laboratory, Stanford Univ., 1994; http://ksl-web.stanford.edu/publications.
11. A. Gómez-Pérez, "Towards a Framework to Verify Knowledge Sharing Technology," *Expert Systems with Applications*, Vol. 11, No. 4, 1996, pp. 519–529.

12. J.A. Bateman, B. Magnini, and G. Fabris, "The Generalized Upper Model Knowledge Base: Organization and Use," *Towards Very Large Knowledge Bases*, IOS Press, Amsterdam, 1995, pp. 60–72.

13. J. Arpírez et al., "(ONTO)2Agent: An Ontology-Based WWW Broker to Select Ontologies," *Workshop on Applications of Ontologies and Problem Solving Methods* (part of ECAI '98: 1996 European Conf. AI), European Coordinating Committee for Artificial Intelligence, 1998, pp. 16–24.

14. V.R. Benjamins, D. Fensel, and A. Gómez-Pérez, "Ontology-Based Knowledge Management in Networks," *Proc. Second Int'l Conf. Practical Aspects of Knowledge Management*, 1998.

Mariano Fernández López is a teaching assistant at the Centro de Estudios Universitarios and an invited professor in the Universidad Pontificia de Salamanca. His research activities include methodologies for building ontologies, ontological reengineering, and uses of ontologies in applications. He received his BA in computer science from the Polytechnic University of Madrid. He received his MSC in knowledge and software engineering from the Facultad de Informática de Madrid, and is a PhD student in computer science there. He belongs to the Knowledge Reusing Group of the Artificial Intelligence Laboratory at the Facultad de Informática de Madrid. He has received the Gold Chip. Contact him at Laboratorio de Inteligencia Artificial, Facultad de Informática, Univ. Politécnica de Madrid, Campus de Montegancedo sn., Boadilla del Monte, 28660 Madrid, Spain; mfernand@delicias.dia.fi.upm.es.

Asunción Gómez-Pérez is an associate professor at the Computer Science School at the Polytechnic University of Madrid. She also is the executive director of the School's Artificial Intelligence Laboratory. Her research activities include theoretical ontological foundations, methodologies for building ontologies, ontological reengineering, evaluation of ontologies, uses of ontologies in applications, and knowledge management on the Web. She received her BA in computer science, her MSC in knowledge engineering, and her PhD in computer science, all from the Polytechnic University of Madrid. She also has an MSC in business administration from the Universidad Pontificia de Comillas, Spain. Contact her at Laboratorio de Inteligencia Artificial, Facultad de Informática, Univ. Politécnica de Madrid, Campus de Montegancedo sn., Boadilla del Monte, 28660 Madrid, Spain; asun@delicias.dia.fi.upm.es or asun@fi.upm.es.

Alejandro Pazos Sierra leads the RNASA Lab (Neural Networks and Adaptative Systems Laboratory), which he founded in 1995. He received his MDr in Medicine from Santiago de Compostela University, his MS in knowledge engineering and his PhD in computer science from the Polytechnic University of Madrid, and his PhD in medicine from the Universidad Complutense de Madrid. He is a member of the IEEE, ACM, International Neural Network Society, IAKE (International Association of Knowledge), American Association for the Advancement of Science, Genetic and Evolutionary Computation Conference (GECCO) Program Committee, and New York Academy of Science. Contact him at Laboratorio de Redes de Neuronas, Artificiais e Sistemas Adaptativos, Departamento de Computación, Facultad de Informática, Univ. da Coruña, Campus de Elviña, 15071 A Coruña, Spain; ciapazos@udc.es.

Juan Pazos Sierra is a professor of computer science at the Department of Artificial Intelligence of the School of Computer Science in Madrid, where he set up the first Spanish AI laboratory. He and his colleagues developed the domino program, Luciano, which was awarded a gold medal at the 1989 Computer Olympics in London. He received the first Spanish doctorate in computer science from the Polytechnic University of Madrid. Contact him at Laboratorio de Inteligencia Artificial, Facultad de Informática, Univ. Politécnica de Madrid, Campus de Montegancedo sn., Boadilla del Monte, 28660 Madrid, Spain; jpazos@delicias.dia.fi.upm.es.