Chapter 3
Methodologies and Methods for Building Ontologies

The 1990s and the first years of this new century have witnessed the growing interest of many practitioners in approaches for building ontologies from scratch, for reusing other ontologies and for using semi-automatic methods that reduce the knowledge acquisition bottleneck of the ontology development process. Until the mid-1990s, this process was an art rather than an engineering activity. Each development team usually followed their own set of principles, design criteria and phases for manually building the ontology. The absence of common and structured guidelines slowed the development of ontologies within and between teams, the extension of any ontology, the possibility of ontologies of being reused in others, and the use of ontologies in final applications.

In 1996, the first workshop on Ontological Engineering was held in conjunction with the 12th European Conference on Artificial Intelligence. Its goal was to explore a suite of principles, design decisions, and rules of good practice from which other ontology designers could profit. A second workshop was held in 1997 with the same topic though this time it took place with the AAAI Spring Symposium Series in Stanford. One of the main aspects dealt with in this workshop was the use of methodologies to design and evaluate ontologies. Since then, methodological aspects related to different activities of the ontology development process and its lifecycle are included in most of the international conferences on the Ontological Engineering field (EKAW, FOIS, K-CAP, KAW, etc.) and workshops held in conjunction with AAAI, ECAI and IJCAI.

The goal of this chapter is to present the main methodologies and methods used to build ontologies from scratch, by reusing and re-engineering other ontologies, by a process of merging or by using an ontology learning approach.
In the literature, the terms methodology, method, technique, process, activity, etc., are used indiscriminately (de Hoog, 1998). To make clear the use of these terms on the Ontological Engineering field, we have adopted the IEEE definitions for such terms in this chapter. Thus, all the works presented in this chapter are presented uniformly, independently of how each author uses its own terminology.

The IEEE defines a methodology as “a comprehensive, integrated series of techniques or methods creating a general systems theory of how a class of thought-intensive work ought be performed” (IEEE, 1990). Methodologies are broadly used in Software Engineering (Downs et al., 1998; Pressmann, 2000) and Knowledge Engineering (Waterman, 1986; Wielinga et al., 1992; and Gómez-Pérez et al., 1997).

As we can read in the previous definition of methodology, both methods and techniques are parts of methodologies. A method is a set of “orderly process or procedure used in the engineering of a product or performing a service” (IEEE, 1990). A technique is “a technical and managerial procedure used to achieve a given objective” (IEEE, 1990).

De Hoog (1998) explores the relationships between methodologies and methods for building Knowledge Based Systems (KBS). According to him, methodologies and methods are not the same because “methodologies refer to knowledge about methods”. Methodologies state “what”, “who” and “when” a given activity should be performed. Greenwood (Greenwood, 1973) also explores the differences between methods and techniques. A method is a general procedure while a technique is the specific application of a method and the way in which the method is executed. For example, the method to build a circumference consists in plotting a line whose points have the same distance with respect to the center. There are several techniques for applying this method: plotting the line with a compass, with a pencil tied to a string, with a circumference template, etc.

According to the IEEE methodology definition, methods and techniques are related. In fact, methods and techniques are used to carry out tasks inside the different processes of which a methodology consists. A process is a “function that must be performed in the software life cycle. A process is composed of activities” (IEEE, 1996). An activity is “a constituent task of a process” (IEEE, 1996). A task is the smallest unit of work subject to management accountability. “A task is a well-defined work assignment for one or more project members. Related tasks are usually grouped to form activities” (IEEE, 1996). For example, the system design in Software Engineering is a process that has to be performed in every project. To execute this process, the following activities have to be carried out: database design, architectural design, interface design, etc. The database design activity, in turn, requires performing different tasks like data conceptual modeling, physical structure design, etc. To carry out each task different methods can be used. For example, physical structure design can be performed through the functional dependency method, or through conceptual model transformation rules. In addition, each method has associated its own techniques.

The relationships between the aforementioned definitions are summarized in Figure 3.1, where we can see that a methodology is composed of methods and techniques. Methods are composed of processes and are detailed with techniques. Processes are composed of activities. Finally, the activities make groups of tasks.
3.1 Ontology Development Process

In 1997, the ontology development process (Fernández-López et al., 1997) was identified on the framework of the METHONTOLOGY methodology for ontology construction. Such a proposal was based on the IEEE standard for software development (IEEE, 1996). The ontology development process refers to *which* activities are performed when building ontologies. It is crucial to identify these activities if agreement is to be reached on ontologies built by geographically distant cooperative teams. It is advisable to carry out the three categories of activities presented in Figure 3.2 and steer clear of anarchic constructions:

**Ontology management activities** include scheduling, control and quality assurance. The *scheduling* activity identifies the tasks to be performed, their arrangement, and the time and resources needed for their completion. This activity is essential for ontologies that use ontologies stored in ontology libraries or for ontologies that require a high level of abstraction and generality. The *control* activity guarantees that scheduled tasks are completed in the manner intended to be performed. Finally, the *quality assurance* activity assures that the quality of each and every product output (ontology, software and documentation) is satisfactory.

**Ontology development oriented activities** are grouped, as presented in Figure 3.2, into pre-development, development and post-development activities.

During the pre-development, an *environment study* is carried out to know the platforms where the ontology will be used, the applications where the ontology will be integrated, etc. Also during the pre-development, the *feasibility study* answers...
questions like: is it possible to build the ontology? is it suitable to build the ontology?, etc.

Once in the development, the \textit{specification} activity\footnote{In (Fernández-López et al., 1997) we considered \textit{specification} as a pre-development activity. However, following more strictly the IEEE standard for software development, we have considered the specification activity as being part of the proper development process. In fact, the result of this activity is an ontology description (usually in natural language) that will be transformed into a conceptual model by the \textit{conceptualization} activity.} states why the ontology is being built, what its intended uses are and who the end-users are. The \textit{conceptualization} activity structures the domain knowledge as meaningful models at the knowledge level (Newell, 1982). The \textit{formalization} activity transforms the conceptual model into a formal or semi-computable model. The \textit{implementation} activity builds computable models in an ontology language (in Chapter 4 we describe several languages used to implement ontologies).

During the post-development, the \textit{maintenance} activity updates and corrects the ontology if needed. Also during the post-development, the ontology is \textit{(re)}used by other ontologies or applications.

Finally, \textbf{ontology support activities} include a series of activities performed at the same time as the development-oriented activities, without which the ontology could not be built. They include knowledge acquisition, evaluation, integration, merging, alignment, documentation and configuration management. The goal of the \textit{knowledge acquisition} activity is to acquire knowledge from experts of a given domain or through some kind of (semi)automatic process, which is called ontology learning (Kietz et al., 2000). The \textit{evaluation} activity (Gómez-Pérez et al., 1995)
makes a technical judgment of the ontologies, of their associated software environments, and of the documentation. This judgment is made with respect to a frame of reference during each stage and between stages of the ontology’s life cycle. The integration activity is required when building a new ontology by reusing other ontologies already available. Another support activity is merging (Gangemi et al., 1999; Noy and Musen, 2001; Stumme and Maedche, 2001), which consists in obtaining a new ontology starting from several ontologies on the same domain. The resulting ontology is able to unify concepts, terminology, definitions, constraints, etc., from all the source ontologies (Pinto et al., 1999). The merge of two or more ontologies can be carried out either in run-time or design time. The alignment activity establishes different kinds of mappings (or links) between the involved ontologies. Hence this option preserves the original ontologies and does not merge them. The documentation activity details, clearly and exhaustively, each and every one of the completed stages and products generated. The configuration management activity records all the versions of the documentation and of the ontology code to control the changes.

From this analysis we can conclude that the ontology development process identifies which activities are to be performed. However, it does not identify the order in which the activities should be performed (Fernández-López et al., 1997) (see also IEEE, 1996). The ontology life cycle identifies when the activities should be carried out, that is, it identifies the set of stages through which the ontology moves during its life time, describes what activities are to be performed in each stage and how the stages are related (relation of precedence, return, etc.).

3.2 Ontology Methodology Evolution

Basically, a series of methods and methodologies for developing ontologies have been reported. In 1990, Lenat and Guha (1990) published some general steps and some interesting points about the Cyc development. Some years later, in 1995, on the basis of the experience gathered in developing the Enterprise Ontology (Uschold and King, 1995) and the TOVE (TOronto Virtual Enterprise) project ontology (Grüninger and Fox, 1995) both in the domain of enterprise modeling, the first guidelines were proposed and later refined in (Uschold and Grüninger, 1996; Uschold, 1996).

At the 12th European Conference for Artificial Intelligence (ECAI’96), Bernaras and colleagues (1996) presented a method to build an ontology in the domain of electrical networks as part of the Esprit KACTUS project (KACTUS, 1996). The METHONTOLOGY methodology (Gómez-Pérez et al., 1996) appeared simultaneously and was extended in further papers (Fernández-López et al., 1997; Gómez-Pérez, 1998; Fernández-López et al., 1999). In 1997, a new method was proposed for building ontologies based on the SENSUS ontology (Swartout et al., 1997).

Then some years later, the On-To-Knowledge methodology appeared within the project with the same name (Staab et al., 2001).
But methods and methodologies have not been created only for building ontologies from scratch. When (re)using an existing ontology it might happen that the ontology is implemented in a language with an underlying knowledge representation paradigm different to the knowledge representation conventions used by the ontology that reuses it, or that the ontology to be reused has different ontological commitments, etc. For solving some of these problems METHONTOLOGY includes a re-engineering method (Gómez-Pérez and Rojas, 1999).

Although one of the main purposes of ontologies is to reduce the knowledge acquisition bottleneck, to acquire knowledge for building ontologies still requires a lot of time and resources. As a consequence, ontology learning methods have been thought up to decrease the effort made during the knowledge acquisition process (Aussenac-Gilles et al., 2000a; Kietz et al., 2000). Such methods are used with several purposes: to create a new ontology from scratch, to enrich an existing one with new terms, and to acquire knowledge for some tasks. We present in this chapter ontology learning methods mainly based on natural language analysis from texts.

Ontologies aim to capture consensual knowledge of a given domain in a generic and formal way to be reused and shared across applications and by groups of people. From this definition we could wrongly infer that there is only one ontology for a domain. In fact, we can find in the literature several ontologies that model, though in different ways, the same kind of knowledge or domain. Noy and Musen (1999) distinguish two approaches for unifying the terminologies of the ontologies: ontology alignment and ontology merging. **Ontology alignment methods** establish different kinds of mappings (or links) between the ontologies, hence this option preserves the original ontologies. However, **ontology merging methods** propose to generate a unique ontology from the original ontologies. The merging process usually requires to establish mappings between the ontologies to merge. Given the current state of affairs and in the context of the Semantic Web, it is more suitable to establish ontological mappings between existing ontologies on the same topic than to pretend to build the unified knowledge model for such a topic from scratch. In this chapter, we present the ONIONS methodology (Gangemi et al., 1999), FCA-Merge method (Stumme and Maedche, 2001), Chimaera (McGuinness et al., 2000) and PROMPT (Noy and Musen, 2000) for merging ontologies. We also present the method used by AnchorPROMPT (Noy and Musen, 2001) for ontology alignment.

All the aforementioned approaches do not consider collaborative and distributed construction of ontologies. In fact, the first method that included a proposal for collaborative construction was Co4 (Euzenat, 1995; 1996). This method includes a protocol for agreeing new pieces of knowledge with the rest of the knowledge architecture, which has been previously agreed upon.

Ontologies cannot be reused by other ontologies or used by applications without evaluating first their content from a technical point of view. As for guidelines to evaluate ontologies, the first publications were (Gómez-Pérez et al., 1995; Gómez-Pérez, 1996); and she has continued working on the evaluation of taxonomic
knowledge (Gómez-Pérez, 1999; 2001). Guarino and colleagues have developed OntoClean (Welty and Guarino, 2001), a method to analyze and clean the taxonomy of an existing ontology by means of a set of principles based in philosophy. Other interesting works on knowledge base evaluation with ontologies are those of Kalfoglou and Robertson (1999a; 1999b).

To conclude, the purpose of this chapter is not to evaluate which methodology is the best. As de Hoog says (1998, with his own emphasis), “it is extremely difficult to judge the value of a methodology in an objective way. Experimentation is of course the proper way to do it, but it is hardly feasible because there are too many conditions that cannot be controlled”. On the one hand “introducing an experimental toy problem will violate the basic assumption behind the need for a methodology: a complex development process. On the other hand, if we extrapolate the argument that de Hoog provides for knowledge based systems to the ontology field, it is not very likely that someone will pay twice for building the same complex ontology with different approaches.

3.3 Ontology Development Methods and Methodologies

This section presents and compares the classical methodologies and methods used to build ontologies from scratch or by reusing other ontologies. In particular the approaches dealt with are the Cyc method, the Uschold and King’s method, the Grüninger and Fox’s methodology, the KACTUS approach, METHONTOLOGY, the SENSUS method, and the On-To-Knowledge methodology.

3.3.1 The Cyc method

In the middle of the 1980s, the Microelectronics and Computer Technology Corporation (MCC) started to create Cyc, a huge knowledge base (KB) with common sense knowledge. Cyc was built upon a core of over 1,000,000 hand-entered assertions designed to capture a large portion of what people normally consider consensus knowledge about the world.

To implement Cyc, the CycL language was used. This is a hybrid language that combines frames with predicate calculus. The CycL inference engine allows: multiple inheritance, automatic classification, maintenance of inverse links, firing of daemons, constraint checking, agenda-based best-first search, etc.; it also has a truth maintenance system, a contradiction detection and a resolution module.

The reason why Cyc can be considered as an ontology is because it can be used as a substrate for building different intelligent systems that can communicate and interact.

As Figure 3.3 illustrates, during the development of Cyc the following processes were carried out (Lenat and Guha, 1990):
Process I. Manual coding of articles and pieces of knowledge. This first process was carried out by hand because existing natural language systems and learning machines did not handle enough common sense knowledge to search for this kind of new knowledge. This knowledge was acquired in three ways:

- **Encoding the knowledge required to understand books and newspapers.** This does not mean encoding the contents of such works, but searching and representing the underlying common sense knowledge that the writers of those articles assumed their readers already possessed.
- **Examination of articles that are unbelievable,** for example, a paper that says that an airplane was flying for one year without filling up. The purpose of this examination is to study the rationale that makes some articles unbelievable.
- **Identification of questions that “anyone” should be able to answer by having just read the text.** The KB is augmented to be able to answer such questions.

Process II. Knowledge coding aided by tools using the knowledge already stored in the Cyc KB. This second process can be performed when tools for analyzing natural language and machine learning tools can use enough common sense knowledge to search for new common sense knowledge.

Process III. Knowledge codification mainly performed by tools using knowledge already stored in the Cyc KB. This third process delegates most of the work to the tools. To work with Cyc tools users only recommend the system the knowledge sources to be read and explain the most difficult parts of the text.

Two activities are performed in all of the three previous processes:

**Activity 1. Development of a knowledge representation and top level ontology containing the most abstract concepts.** As we saw in Chapter 2, terms like attribute or attribute value are examples of knowledge
representation terms, and thing, intangible or collection are examples of abstract concepts.

Activity 2. Representation of the knowledge of different domains using such primitives.

Up to now, this method has been used only for building the Cyc KB; however, Cyc has different micro-theories to gather the knowledge of different domains from different viewpoints. A micro-theory (Guha, 1991) is a theory of some topic, e.g., a theory of mechanics, chemical elements, etc. Different micro-theories might take different assumptions and simplifications about the world, be seen from different perspectives, and be used in different areas.

With regard to the applications in which Cyc ontologies are used, there are several modules integrated into the Cyc KB and the CycL inference engine. One of these is the Heterogeneous Database Integration System, which maps the Cyc vocabulary into database schemas, that is, the data stored on the databases are interpreted according to the Cyc vocabulary. Other module is the Knowledge-Enhanced Searching of Captioned Information, which permits making queries over images using their captions in natural language. Another module is the Guided Integration of Structured Terminology (GIST), which allows users to import and simultaneously manage and integrate multiple thesauri.

Cyc agents have also been built. All these agents have a common core with knowledge of the Cyc KB plus domain knowledge from the specific domain of the agent.

Finally, the WWW Information Retrieval module uses the natural language tools to access the Cyc KB and allows extending the Cyc KB with information available on the Web.

3.3.2 Uschold and King’s method

Uschold and King (1995) proposed the first method for building ontologies, which was extended in (Uschold and Grüninger, 1996). They proposed some guidelines based on their experience of developing the Enterprise Ontology. As described in Section 2.4.4, this ontology was developed as a part of the Enterprise Project by the Artificial Intelligence Applications Institute at the University of Edinburgh with its partners IBM, Lloyd’s Register, Logica UK Limited, and Unilever.
To build an ontology according to Uschold and King’s approach, the following processes must be performed: (1) identify the purpose of the ontology, (2) build it, (3) evaluate it, and (4) document it. During the building process, the authors propose capturing knowledge, coding it and integrating other ontologies inside the current ontology. These processes, shown in Figure 3.4, are:

**Process 1. To identify the purpose and the scope.** The goal here is to clarify why the ontology is being built, what its intended uses are (to be reused, shared, used, used as a part of a KB, etc.) and what the relevant terms on the domain will be. Considering our traveling example, the purpose of building a travel ontology would be, for instance, to provide a consensual knowledge model of the traveling domain that will be used by travel agencies. Such ontology could be also used for other purposes, for instance, for developing a catalogue about lodgings or transport means. Concerning the scope, the list of relevant terms to be included are: places and types of place, lodging and types of lodging (hotel, motel, camping, etc.), trains, buses, undergrounds, etc.

**Process 2. To build the ontology.** It is broken down into three activities:

*Activity 2.1. Ontology capture.* The following tasks are proposed for capturing knowledge: to identify key concepts and relationships in the domain of interest; to produce precise and unambiguous textual definitions for such concepts and relationships; to identify the terms that refer to such concepts and relationships and thus to reach an agreement. The textual definitions are not created following the style of classical dictionaries but are built by referring to other terms and including notions such as class, relation, etc. Consequently, these natural language definitions determine the knowledge representation ontology to be used. Some concepts and relations identified for our travel agency ontology could be (with their associated terms and definitions) the following:

- **Transport means:** it is a class. Each transport means has a starting point.
- **Bus:** it is a class. It is a kind of transport means.
- **Local bus:** it is a class. It is a bus whose departure place, destination place and stops are at the same Location.

To identify the concepts in the ontology, Uschold and Grüninger (1996) pointed out three strategies: bottom-up, top-down, and middle-out.

The *bottom-up strategy* proposes identifying first the most specific concepts and then, generalizing them into more abstract concepts. The authors affirm that a bottom-up approach results in a very high level of detail. This approach: (1) increases the overall effort, (2) makes it difficult to spot commonality between related concepts, and (3) increases the risk of inconsistencies which leads to (4) re-work and even to more effort. In our example, if we want to represent transport means following a bottom-up strategy, we identify first the following types of means, as we can see in Figure 3.5: London underground, London local bus, London taxi, Madrid underground,
Madrid local bus and Madrid taxi. Such transport means could be grouped not only into underground, local bus and taxi, but also into Madrid transport means and London transport means. These last concepts may not be interesting for our ontology. We omit the concept bus and other types of buses like a shuttle bus, which might be important.

With the top-down strategy the most abstract concepts are identified first, and then specialized into more specific concepts. The main result of using a top-down approach is a better control of the level of detail; however, starting at the top can result in choosing and imposing arbitrary and possibly not needed high level categories. Because these do not arise naturally, there is a risk of less stability in the model, what leads to reworking and to greater effort. The emphasis on dividing up rather than putting together also results, for a different reason, in missing the commonality inherent in the complex web of interconnected concepts. In our example, as Figure 3.6 illustrates, we start building the taxonomy with the concept object and we distinguish between concrete object and abstract object. Then, we consider the different transport means as abstract objects that use concrete objects (taxi, bus, train). With this strategy we can generate too many concepts (object,
concrete object, etc.), and we may make an unnecessary distinction between transport means and the objects they need.

Finally, the middle-out strategy recommends identifying first the core of basic terms, and then specifying and generalizing them as required. Uschold and Grüninger claim that a middle-out approach, by contrast, strikes a balance in terms of the level of detail. Detail only arises as necessary by specializing the basic concepts, so some efforts are avoided. If we start with the most important concepts first, and define higher level concepts in terms of them, these higher level categories arise naturally and thus are more likely to be stable. This, in turn, leads to less re-work and less overall effort. In the example on the transport domain, as can be seen in Figure 3.7, we have identified first the concepts: underground, bus and taxi, which are the most important for us. Then, we have generated the top and the bottom concepts for bus, which are local bus, shuttle bus and coach.

Figure 3.7: Example of a taxonomy built following the middle-out approach.

Activity 2.2. Coding. This activity involves two tasks: (a) committing to basic terms that will be used to specify the representation ontology (e.g., class, entity, relation), and (b) writing the code.

Activity 2.3. Integrating existing ontologies. This activity refers to how and whether to use ontologies that already exist. Integration can be done in parallel with the previous activities of this process. Examples of ontologies that could be reused are: the Frame Ontology for modeling the domain ontology using a frame-based approach, KIF-Numbers to represent numbers, or the Standard-Units that contains descriptions of units of measure. This last ontology is useful for representing different kind of distances (meter, kilometer, etc.).

Process 3. To evaluate. The authors take the definition from Gómez-Pérez and colleagues (1995) and affirm that: “to make a technical judgment of the ontologies, their associated software environment, and documentation with respect to a frame of reference ... the frame of reference may be requirement specifications, competency questions, and/or the real world”.
Process 4. To document. On this process, the guidelines for documenting the ontology are established and are possibly different, according to the type and purpose of the ontology. A guideline example is to locate similar definitions together or to create naming conventions such as: using upper or lowercase letters to name the terms, or writing the terms of the representation ontology in uppercase.

According to Uschold and Grüninger (1996), the aforementioned processes are not sufficient to have a methodology. Every methodology should also include a set of techniques, methods, and principles for each of the above four stages, and it should indicate what relationships between the stages exist (e.g., recommended order, interleaving, inputs/outputs).

The main drawback of this method is the lack of a conceptualization process before implementing the ontology. The goal of a conceptualization process is to provide a domain model less formal than the implementation model but more than the definition of the model in natural language. Other problems provoked by this lack of a conceptualization process are (Fernández-López et al., 1999) that: (1) domain experts, human users, and ontologists have many difficulties in understanding ontologies implemented in ontology languages; and (2) domain experts are not able to build ontologies in their domain of expertise. So the bottleneck in knowledge acquisition still persists! To obtain the conceptual model from the ontology implementation, we can apply the process of re-engineering explained in Section 3.4.

3.3.3 Grüninger and Fox’s methodology

Based on the experience of the TOVE project on the enterprise domain, which was developed at the University of Toronto, Grüninger and Fox (1995) published a formal approach to build and evaluate ontologies. This methodology has been used to build the TOVE ontologies, which are the pillars of the Enterprise Design Workbench2, a design environment that permits the user to explore a variety of enterprise designs.

Grüninger and Fox’s methodology is inspired by the development of knowledge based systems using first order logic. They propose identifying intuitively the main scenarios, that is, possible applications in which the ontology will be used. Then, a set of natural language questions, called competency questions, are used to determine the scope of the ontology. These questions and their answers are both used to extract the main concepts and their properties, relations and formal axioms of the ontology. On the other hand, knowledge is formally expressed in first-order logic. This is a very formal methodology that takes advantage of the robustness of classic logic and can be used as a guide to transform informal scenarios in computable models.

The processes identified in this methodology are shown in Figure 3.8:

---

2 http://www.eil.toronto.ca/eil.html
Figure 3.8: Processes in Grüninger and Fox’s methodology.

**Process 1. To identify motivating scenarios.** The development of ontologies is motivated by scenarios related to the applications that will use the ontology. Such scenarios describe a set of the ontology’s requirements that the ontology should satisfy after being formally implemented. A motivating scenario also provides a set of intuitively possible solutions to the scenario problems. These solutions give a first idea of the informal intended semantics of the objects and relations that will later be included in the ontology. In our example, the purpose of building a travel ontology is to provide a consensual knowledge model of the traveling domain that could be used by travel agencies.

**Process 2: To elaborate informal competency questions.** Given the set of informal scenarios, a set of informal competency questions are identified. Informal competency questions are those written in natural language to be answered by the ontology once the ontology is expressed in a formal language. The competency questions play the role of a type of requirement specification against which the ontology can be evaluated. Examples of informal competency questions are:

a) Given the preferences of a traveler (cultural travel, mountain travel, beach travel, etc.) and some constraints (economical or about the travel itself), which destinations are the most appropriate?

b) Given a young traveler with a budget for lodging, what kinds of lodgings are available?

This methodology proposes to stratify the set of competency questions. An ontology is not well-designed if all the competency questions are simple queries, that is, if the questions cannot be decomposed or composed into more specific or general questions, respectively. Competency questions can be split off into more
specific (or atomic) competency questions, and the answer to a question can be used to answer more complex questions. For example, the competence question (a) could be decomposed to deal separately with cultural and economical constraints on the following:

a.1) Given the preferences of a traveler (cultural travel, mountain travel, beach travel, etc.) and some economical constraints, which destinations are the most appropriate?

a.2) Given the preferences of a traveler (cultural travel, mountain travel, beach travel, etc.) and some traveling constraints (i.e., the traveler hates traveling by plane), which destinations are the most appropriate?

Each competency question is useful as a base for obtaining assumptions, constraints, the necessary input data, etc., as Figure 3.9 illustrates.

![Diagram of competency question](image)

**Figure 3.9:** An example of a competency question.

**Process 3: To specify the terminology using first order logic.** The ontologist can use informal competency questions to extract the content of the ontology. Such content will be formally represented in the ontology. From the informal competency questions, the ontologist extracts the terminology that will be formally represented by means of concepts, attributes and relations in a first-order logic language. From the answers in natural language to the competency questions, the ontologist extracts the knowledge to be included in the formal definitions of concepts, relations, and formal axioms.
To build the ontology in first-order logic, the designers should carry out the tasks of a traditional formalization in first-order logic:

- **Identifying objects in the universe of discourse.** Examples of objects, which are instances, of the traveling domain are: Paris, Madrid, London, New York, Flight IB2140, Bus 125, Train C4, Hotel Palace in Madrid, Hotel Travel Lodge in Newcastle, etc.

- **Identifying predicates.** Unary predicates are used for representing concepts, binary predicates for attributes, and binary relations and n-ary predicates for n-ary relations. Examples of predicates that represent concepts are:
  - transport-means ($transport)
  - bus ($bus)
  - train ($train)
  - traveler ($traveler)
  - young-traveler ($traveler)
  - adult-traveler ($traveler)
  - old-traveler ($traveler)
  - destination ($destination)
  - lodging ($lodging)
  - camping ($camping)
  - hotel ($hotel)
  - travel-information ($travelinfo)
  - location ($location), etc.

Examples of predicates that represent attributes are:
  - traveler-name ($traveler,$string)
  - hotel-name ($hotel,$string)
  - bus-number ($bus,$integer), etc.

Note that we cannot use the attribute name to refer to the name of a traveler, a hotel, and a bus, since predicates are only distinguished by their name and their arity. We have created the predicates traveler-name, hotel-name, and bus-number.

Examples of predicates that represent binary relations are:
  - has-destination ($travelinfo,$location)
  - has-departure ($travelinfo,$location), etc.

Where has-destination represents that the travel $travelinfo arrives at the location $location. And has-departure means that the travel $travelinfo departs from the location $location.

As we will see later, the Subclass-Of relation can be represented with the implication if we make some assumptions.

---

3 Variables are preceded by the symbol “$".
**Process 4: To write competency questions in a formal way using formal terminology.** Informal competency questions are written as an entailment of consistency problems with respect to the axioms in the ontology. Such axioms will be defined in process 5. For instance, the previous informal competency question “Given a set of traveler’s preferences (cultural travel, mountain travel, beach travel, etc.) and some economical constraints, which destinations are the most appropriate?” would be formally represented in first-order logic as follows:

\[
\exists x \exists y \ (\text{destination}(x) \land \text{travel-information}(y) \land \\
\text{wants-to-travel}(c,y) \land \\
\text{age}(c,a) \land \text{preferences}(c,e) \land \text{max-expense}(c,b) \land \\
\text{has-destination}(y,x))
\]

where the meaning of the constants is: \(c\) is a traveler, \(b\) is the maximum quantity that the traveler can spend, \(a\) is the age of the traveler, and \(e\) are its preferences.

**Process 5: To specify axioms using first-order logic.** This methodology proposes using axioms to specify the definitions of terms in the ontology and constraints in their interpretation; axioms are defined as a first-order sentences using predicates of the ontology. The following axiom expresses that a traveler is a young traveler if and only if his age is equal or lower than 29.

\[
\forall x \ (\text{traveler}(x) \land \\
(\exists y \ (\text{integer}(y) \land \text{age}(x,y) \land (y<30)) \leftrightarrow \\
\text{young-traveler}(x))
\]

To represent the Subclass-Of relation the logical implication\(^4\) can be used:

\[
\forall x \ (\text{local-bus}(x) \rightarrow \text{bus}(x))
\]

If the proposed axioms are insufficient to represent the formal competency questions and to characterize the solutions to the questions, other axioms or objects are added.

**Process 6. To specify completeness theorems.** Once the competency questions have been formally stated, we must define the conditions under which the solutions to the questions are complete. This is the basis of completeness theorems for the ontology. In our example, a formal formulation for completeness is not needed since we only have to state that the system shows the traveler all the options.

As the main conclusion to this methodology, we can say that it is well-founded for building and evaluating ontologies, even though some management and support activities are missing.

---

\(^4\) Note that \(\forall x \ (P(x) \rightarrow Q(x))\) in classic first order logic does not have exactly the same meaning as Subclass-Of\((P,Q)\) in the paradigm of frames. For example, non-monotonic reasoning is usually considered when reasoning with frames, and disregarded in classic first order logic. To learn more about the relationships between frames and first order logic, consult Brewka (1987).
3.3.4 The KACTUS approach

This approach was proposed by Amaya Bernaras and her colleagues (1996), inside the Esprit KACTUS project (KACTUS, 1996). An objective of this project was to investigate the feasibility of knowledge reuse in complex technical systems and the role of ontologies to support it (Schreiber et al., 1995).

This approach for developing ontologies is conditioned by application development. Thus, every time an application is built, the ontology that represents the knowledge required for the application is refined. The ontology can be developed by reusing others and can be integrated into ontologies of later applications. Therefore, every time an application is developed, the following processes occur (Bernaras et al., 1996):

**Process 1. Specification of the application**, which provides context to an application and a view of the components that the application tries to model. In this process a list of terms and tasks have to be provided. For example, in our ontology for a travel agency, we will have terms such as: place, the name of the place, a cultural place, etc., and tasks such as: obtain the most suitable place for a client, etc.

**Process 2. Preliminary design based on relevant top-level ontological categories**, where the list of terms and tasks developed during the previous process are used as input for obtaining several views of the global model in accordance with top-level ontological categories, for example, concept, relation, attribute, etc. This design process involves searching ontologies already developed (perhaps for other applications). These ontologies are then refined and extended to be used in the new application, as Figure 3.10 shows. Let us imagine that before developing the travel agency application, another application in the geography domain had been developed for educational purposes. The designers of the travel agency application should try to adapt the knowledge on locations of the former application to the travel domain.

![Figure 3.10: Design in KACTUS method.](image)
Process 3. Ontology refinement and structuring to achieve a definitive design following the modularization and hierarchical organization principles. For the application the designers should follow the traditional Software and Knowledge Engineering recommendations and methodologies. Given that the travel agency ontology has concepts such as client, transport means, etc., we should create a separate ontology for clients, another for transport means, etc.

The KACTUS project illustrates this approach. The authors present the development of three ontologies as the result of the development of the same number of applications. The purpose of the first application is to diagnose faults in an electrical network. The second concerns scheduling service resumption after a fault in that network. By the time the development of the ontology of this application started, the fault ontology had also been built. Then, the developers refined and augmented the domain concepts by looking at the top-level ontological categories, and modified the relevant domain concepts already identified in the diagnosis ontology in order to meet the needs of restoration. Afterwards, they refined the structure of the ontology and obtained its definitive design. The third application controls the electrical network. This application is based on the other two. The ontology of this application can be considered as the junction between the domain ontology for diagnosis and service recovery planning. The union of the ontologies yields a set of sub-ontologies that belong to the intersection and other sets used for one application or the other but not for both at the same time. The sub-ontologies of the intersection are more likely to be reused, since the relevant adaptations in these ontologies should be carried out in, at least, two different applications.

3.3.5 METHONTOLOGY

This methodology was developed within the Ontology group at Universidad Politécnica de Madrid. METHONTOLOGY (Fernández-López et al., 1997; Gómez-Pérez, 1998; Fernández-López et al., 1999) enables the construction of ontologies at the knowledge level. METHONTOLOGY has its roots in the main activities identified by the software development process (IEEE, 1996) and in knowledge engineering methodologies (Gómez-Pérez et al., 1997; Waterman, 1986). This methodology includes: the identification of the ontology development process (already presented in Section 3.1), a life cycle based on evolving prototypes, and techniques to carry out each activity in the management, development-oriented, and support activities.

ODE (Blázquez et al., 1998) and WebODE (Arpírez et al., 2003) were built to give technological support to METHONTOLOGY. Other ontology tools and tool suites can also be used to build ontologies following this methodology, for example, Protégé-2000 (Noy et al., 2000), OntoEdit (Sure et al., 2002a), etc.

METHONTOLOGY has been proposed for ontology construction by the

5 http://www.fipa.org/specs/fipa00086/ (last access, March 30, 2003)
Foundation for Intelligent Physical Agents (FIPA), which promotes inter-operability across agent-based applications.

3.3.5.1 Ontology crossed life cycles

The ontology development process of Section 3.1 was proposed on the framework of this methodology and refers to those activities performed during ontology building. This process does not identify the order in which such activities should be performed. This is the role of the ontology life cycle. METHONTOLOGY proposes an ontology building life cycle based on *evolving prototypes* because it allows adding, changing, and removing terms in each new version (prototype).

For each prototype, METHONTOLOGY proposes to begin with the schedule activity that identifies the tasks to be performed, their arrangement, and the time and resources needed for their completion. After that, the ontology specification activity starts and at the same time several activities begin inside the management (control and quality assurance) and support processes (knowledge acquisition, integration, evaluation, documentation, and configuration management). All these management and support activities are performed in parallel with the development activities (specification, conceptualization, formalization, implementation and maintenance) during the whole life cycle of the ontology.

Once the first prototype has been specified, the conceptual model is built within the ontology conceptualization activity. This is like assembling a jigsaw puzzle with the pieces supplied by the knowledge acquisition activity, which is completed during the conceptualization. Then the formalization and implementation activities are carried out. If some lack is detected after any of these activities, we can return to any of the previous activities to make modifications or refinements. When tools like the WebODE ontology editor are used, the conceptualization model can be automatically implemented into several ontology languages using translators. Consequently, formalization is not a mandatory activity in METHONTOLOGY.

Figure 3.11 shows the ontology life cycle proposed in METHONTOLOGY, and summarizes the previous description. Note that the activities inside the management and support processes are carried out simultaneously with the activities inside the development process.

Related to the support activities, the figure also shows that the knowledge acquisition, integration and evaluation is greater during the ontology conceptualization, and that it decreases during formalization and implementation. The reasons for this greater effort are:

- Most of the knowledge is acquired at the beginning of the ontology construction.
- The integration of other ontologies into the one we are building is not postponed to the implementation activity. Before the integration at the implementation level, the integration at the knowledge level should be carried out.
- The ontology conceptualization must be evaluated accurately to avoid propagating errors in further stages of the ontology life cycle.

The relationships between the activities carried out during ontology development are called *intra-dependencies*, or what is the same, they define the ontology life cycle.
METHONTOLOGY also considers that the activities performed during the development of an ontology may involve performing other activities in other ontologies already built or under construction (Fernández-López et al., 2000). Therefore, METHONTOLOGY considers not only intra-dependencies, but also inter-dependencies. Inter-dependencies are defined as the relationships between activities carried out when building different ontologies. Instead of talking about the life cycle of an ontology, we should talk about crossed life cycles of ontologies. The reason is that, most of the times and before integrating an ontology in a new one, the ontology to be reused is modified or merged with other ontologies of the same domain.

Let us illustrate now the activities commented before with an ontology we made in the past. Suppose that you want to build an ontology about monatomic ions to be used in environmental-pollutants-related studies. You should know that some knowledge about chemical elements and their properties, as well as units of measure of some properties, is required to represent knowledge about ionic concentration.

Before developing the monatomic ions ontology we should look for existing ontologies that could be reused. So we looked for ontologies in the domain of chemical elements of the periodic table and for ontologies describing units of measure. It is important to mention here that the inter-dependencies between the monatomic ions ontology and other ontologies started when the requirements of the monatomic ions ontology were identified. Let us see now in more detail that process. The initial activities performed were:

a) To find candidate ontologies to be reused. We located the Standard Units ontology (Gruber, 1993b) at the Ontolingua Server; this ontology defines basic units of measure. We also located the Chemical Elements ontology (Fernández-
López et al., 1999), which defines the chemical elements of the periodic table. The *Chemical Elements* ontology was built with ODE (Blázquez et al., 1998) and its Ontolingua code is available at the Ontolingua Server. Finally, we found some measure units and chemical entities (atom, ion, molecule, and radical) in the Cyc upper-level ontology at the Cyc server.

b) *To inspect the content and granularity of the candidate ontologies.* The *Standard Units* ontology includes for each unit: natural language definition, physical quantity and some factors of conversion to other units of the same quantity. The Cyc ontology includes only a natural language definition.

c) *To select the ontologies to be reused.* We selected the *Chemical Elements* and the *Standard Units* ontologies because they were more suitable for our purposes, and we used Cyc ontologies for reference purposes.

d) *To evaluate the selected ontologies from a knowledge representation point of view.* The ontologists did a preliminary evaluation of the *Standard Units* and *Chemical Elements* ontologies from a KR point of view. As described in (Gómez-Pérez and Rojas, 1999), several problems were met at the *Standard Units* and *Chemical Elements* ontologies. The most important was that *Standard Units* lacked taxonomic organization since all the instances were of the root class. The result of the review process in *Chemical Elements* showed that different versions of the ontology needed to be merged to output a new unified and corrected ontology.

Simultaneously with the previous evaluation, the domain experts also did a preliminary evaluation of the *Chemical Elements* ontology as they understood its conceptual model. However, we postponed the *Standard Units* domain expert evaluation because the domain experts were unable to understand the Ontolingua code.

Figure 3.12: Inter-dependencies and intra-dependencies in ontology development.
As Figure 3.12 illustrates, the specification of monatomic ions provoked the evaluation of different versions of Chemical Elements and the evaluation of Standard Units. During the conceptualization activity of the Monatomic Ions ontology we built its conceptual model and, simultaneously, we started the integration activity with the goal of reusing definitions from the Standard Units and Chemical Elements. It was at this integration activity of the Monatomic ions where most of the inter-dependencies with the other ontologies appeared.

With respect to Chemical Elements, we evaluated each of the versions before merging them manually. Once they were merged, we evaluated the resulting ontology. During the whole process, we also performed the configuration management of Chemical Elements.

With respect to Standard Units, we re-engineered the old version and carried out the configuration management of the new one. Some of the most important motivations that we had for re-engineering the Standard Units ontology were:

- No conceptual model of the ontology existed, just its implementation code in the Ontolingua language.
- There was no taxonomic organization; there was a single class to which all the instances were subordinated.
- Some definitions had a poor, informal language description, which did not give any information.

Let us analyze the Standard Units life cycle presented in Figure 3.12. The Standard Units ontology was built at the beginning of the 1990s and, probably, several applications now use its definitions. Since the Standard Units ontology was built to the present day, only a few changes have been carried out in its version at the Ontolingua Server and several ontologies and applications reuse the ontology. So, we can say that the Standard Units ontology life cycle was “latent” or “hibernating”. When we developed Chemical Elements in 1996, we identified some units of measure that did not appear at Standard Units, and which we added to the ontology at the Ontolingua Server. We updated the ontology with the new units but we did not carry out big changes in its structure nor in its content. Consequently, these updates could be seen as maintenance activities. And now we can say that the Standard Units life cycle “wakes up” when Standard Units is reused by the Monatomic Ions ontology and the re-engineering process over the Standard Units ontology starts. At this point, the Standard Units life cycle is alive since we have modified its structure and content.

It is interesting to observe in Figure 3.12 how the Standard Units ontology life cycle branches out in two. Thus, two Standard Units ontologies – the Standard Units at the Ontolingua Server and the re-engineered Standard Units – were available after running a re-engineering process on the original one. The opposite occurs with Chemical Elements where several ontologies coexist, each one with its own life cycle, and converge with the new life cycle of the merged Chemical Elements ontology after the merging process.

These confluences and forking of life cycles call for a global management of ontologies. The configuration management of each ontology must not be carried out separately from the ones in which they are integrated, though it must be global and should affect simultaneously all the ontologies handled.
3.3.5.2 Conceptual modeling in METHONTOLOGY

In this section we present the METHONTOLOGY's proposal for ontology conceptualization. This activity deserves a special attention because it determines the rest of the ontology construction. Its objective is to organize and structure the knowledge acquired during the knowledge acquisition activity, using external representations that are independent of the knowledge representation paradigms and implementation languages in which the ontology will be formalized and implemented. Consequently, the conceptualization activity has a strong relationship with the knowledge acquisition activity.

Once the conceptual model is built, the methodology proposes to transform the conceptual model into a formalized model, which will be implemented in an ontology implementation language. That is, along this process the ontologist is moving gradually from the knowledge level to the implementation level, increasing slowly the degree of formality of the knowledge model so that it can be understood by a machine.

In that sense, in Figure 3.13 we adapt Blum’s essential process model (Blum, 1996) of software engineering to the ontological engineering field. The transformation $T_1$, which refers to the conceptual modeling process, can be seen as a transformation of an idea of a domain into a conceptual model that describes such an idea. The transformation $T_2$ converts the conceptual model into a formalized model. The transformation $T_3$ transforms the formalized model into a model which can be executed in a computer. The figure presents $T_1$ and $T_3$ in a continuous line, and $T_2$ in a discontinuous one. They are represented differently because some domain knowledge may be lost when transforming the conceptual model into the formalized model. This happens when the components used to create conceptual models are more expressive than those used to create formal models.

![Figure 3.13: Essential process model in ontology development, adapted from Blum (1996) and Gómez-Pérez and colleagues (1997).](image)

The conceptualization activity in METHONTOLOGY organizes and converts an informally perceived view of a domain into a semi-formal specification using a set of intermediate representations (IRs) based on tabular and graph notations that can be understood by domain experts and ontology developers. METHONTOLOGY proposes to conceptualize the ontology using a set of tabular and graphical IRs that extend those used in the conceptualization phase of the IDEAL methodology for
knowledge-based systems development (Gómez-Pérez et al., 1997). These IRs bridge the gap between the people’s perception of a domain and the languages used to implement ontologies. The expressiveness of the METHONTOLOGY IRs eases the transformation process. In fact, we have proven that our IRs allow conceptualizing the main components (concepts, attributes, relations, formal axioms, rules, etc.) of the traditional ontology languages described in Chapter 4: Ontolingua (Farquhar et al., 1997), LOOM (MacGregor, 1991), OKBC (Chaudhri et al., 1998) OCML (Motta, 1999), and FLogic (Kifer et al., 1995). ODE and WebODE translators transform the conceptual model of the ontology into several ontology languages, as presented in Chapter 5.

When dealing with ontologies, ontologists should not be anarchic in the use of modeling components in the ontology conceptualization. They should not define, for instance, a formal axiom if the terms used to define the axiom are not precisely defined on the ontology. METHONTOLOGY includes in the conceptualization activity the set of tasks for structuring knowledge, as shown in Figure 3.14.

The figure emphasizes the ontology components (concepts, attributes, relations, constants, formal axioms, rules, and instances) attached to each task, and illustrates the order proposed to create such components during the conceptualization activity. This modeling process is not sequential as in a waterfall life cycle model, though some order must be followed to ensure the consistency and completeness of the knowledge represented. If new vocabulary is introduced, the ontologist can return to any previous task.

Figure 3.14: Tasks of the conceptualization activity according to METHONTOLOGY.
Our experience of building ontologies has revealed that ontologists should carry out the following tasks:

Task 1: To build the glossary of terms that identifies the set of terms to be included on the ontology, their natural language definition, and their synonyms and acronyms.

Task 2: To build concept taxonomies to classify concepts. The output of this task could be one or more taxonomies where concepts are classified.

Task 3: To build ad hoc binary relation diagrams to identify ad hoc relationships between concepts of the ontology and with concepts of other ontologies.

Task 4: To build the concept dictionary, which mainly includes the concept instances for each concept, their instance and class attributes, and their ad hoc relations.

Once the concept dictionary is built, the ontologist should define in detail each of the ad hoc binary relations, instance attributes and class attributes identified on the concept dictionary, as well as the main constants of that domain.

Task 5: To describe in detail each ad hoc binary relation that appears on the ad hoc binary relation diagram and on the concept dictionary. The result of this task is the ad hoc binary relation table.

Task 6: To describe in detail each instance attribute that appears on the concept dictionary. The result of this task is the table where instance attributes are described.

Task 7: To describe in detail each class attribute that appears on the concept dictionary. The result of this task is the table where class attributes are described.

Task 8: To describe in detail each constant and to produce a constant table. Constants specify information related to the domain of knowledge, they always take the same value, and are normally used in formulas.

Once that concepts, taxonomies, attributes and relations have been defined, the ontologist should describe formal axioms (task 9) and rules (task 10) that are used for constraint checking and for inferring values for attributes. And only optionally should the ontologists introduce information about instances (task 11).

We will show now how to apply this knowledge structuring activity with an example in the traveling domain.

**Task 1: To build the glossary of terms.** First, the ontologist builds a glossary of terms that includes all the relevant terms of the domain (concepts, instances, attributes that represent concept properties, relations between concepts, etc.), their natural language descriptions, and their synonyms and acronyms. Table 3.1 illustrates a section of the glossary of terms of the travel ontology. It is important to mention that on the initial stages of the ontology conceptualization the glossary of terms might contain several terms that refer to the same concept. Then the ontologist should detect that they appear as synonyms.

---

Although instances can be created when the ontology is used (after its construction) the ontologist can decide whether to model relevant instances or not. This field is optional.
Table 3.1: An excerpt of the Glossary of Terms of our travel ontology.

<table>
<thead>
<tr>
<th>Name</th>
<th>Synonyms</th>
<th>Acronyms</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bed and Breakfast</td>
<td>--</td>
<td>--</td>
<td>An establishment (as an inn) offering lodging and breakfast</td>
<td>Concept</td>
</tr>
<tr>
<td>British Airways Flight</td>
<td>--</td>
<td>BA Flight</td>
<td>Flight operated by British Airways.</td>
<td>Concept</td>
</tr>
<tr>
<td>Business Trip</td>
<td>--</td>
<td>--</td>
<td>A special package for businessmen, consisting of a flight and a good quality hotel.</td>
<td>Concept</td>
</tr>
<tr>
<td>Camping</td>
<td>--</td>
<td>--</td>
<td>Temporal lodging in a camp.</td>
<td>Concept</td>
</tr>
<tr>
<td>Economy Trip</td>
<td>--</td>
<td>--</td>
<td>An economic package, usually costing less than 1000$.</td>
<td>Concept</td>
</tr>
<tr>
<td>European Location</td>
<td>--</td>
<td>--</td>
<td>A location in Europe.</td>
<td>Concept</td>
</tr>
<tr>
<td>Five-star Hotel</td>
<td>--</td>
<td>--</td>
<td>High quality hotel.</td>
<td>Concept</td>
</tr>
<tr>
<td>Flight</td>
<td>--</td>
<td>--</td>
<td>A journey by plane identified by a flight number.</td>
<td>Concept</td>
</tr>
<tr>
<td>Hotel</td>
<td>--</td>
<td>--</td>
<td>An establishment that provides lodging and usually meals, entertainment, and various personal services for the public</td>
<td>Concept</td>
</tr>
<tr>
<td>Iberia Flight</td>
<td>--</td>
<td>IB Flight</td>
<td>Flight operated by Iberia.</td>
<td>Concept</td>
</tr>
<tr>
<td>Japan Location</td>
<td>--</td>
<td>--</td>
<td>A location in Japan.</td>
<td>Concept</td>
</tr>
<tr>
<td>Location</td>
<td>Place</td>
<td>--</td>
<td>A position or site occupied or available for occupancy or marked by some distinguishing feature.</td>
<td>Concept</td>
</tr>
<tr>
<td>Lodging</td>
<td>Accommodation</td>
<td>--</td>
<td>A temporary place to stay during a trip, sleeping accommodations.</td>
<td>Concept</td>
</tr>
<tr>
<td>Luxury Trip</td>
<td>--</td>
<td>--</td>
<td>A luxury and expensive trip.</td>
<td>Concept</td>
</tr>
<tr>
<td>Spain Location</td>
<td>--</td>
<td>--</td>
<td>A location in Spain.</td>
<td>Concept</td>
</tr>
<tr>
<td>Train Travel</td>
<td>Rail Travel</td>
<td>--</td>
<td>A journey by train.</td>
<td>Concept</td>
</tr>
<tr>
<td>Travel</td>
<td>--</td>
<td>--</td>
<td>A journey from place to place.</td>
<td>Concept</td>
</tr>
<tr>
<td>Travel Package</td>
<td>--</td>
<td>--</td>
<td>A travel package that a person can ask for. It consists of one or several means of transport and one or several accommodations.</td>
<td>Concept</td>
</tr>
<tr>
<td>USA Location</td>
<td>--</td>
<td>--</td>
<td>A location in the United States.</td>
<td>Concept</td>
</tr>
<tr>
<td>maximum Number of Travelers in a Plane</td>
<td>--</td>
<td>--</td>
<td>The maximum number of travelers in a plane at the same time.</td>
<td>Constant</td>
</tr>
<tr>
<td>arrival Date</td>
<td>--</td>
<td>--</td>
<td>Date of arrival of the trip.</td>
<td>Instance Attribute</td>
</tr>
<tr>
<td>departure Date</td>
<td>--</td>
<td>--</td>
<td>Date of departure of the trip.</td>
<td>Instance Attribute</td>
</tr>
<tr>
<td>final Price</td>
<td>--</td>
<td>--</td>
<td>The final price of the package for a traveler.</td>
<td>Instance Attribute</td>
</tr>
<tr>
<td>name</td>
<td>--</td>
<td>--</td>
<td>The location name.</td>
<td>Instance Attribute</td>
</tr>
<tr>
<td>single Fare</td>
<td>--</td>
<td>--</td>
<td>Fare of a single ticket.</td>
<td>Instance Attribute</td>
</tr>
<tr>
<td>departure Place (Travel, Location)</td>
<td>--</td>
<td>--</td>
<td>The location where the travel departs from.</td>
<td>Relation</td>
</tr>
<tr>
<td>placed in (Lodging, Location)</td>
<td>--</td>
<td>--</td>
<td>The place where a lodging is located.</td>
<td>Relation</td>
</tr>
</tbody>
</table>
Task 2: To build concept taxonomies. When the glossary of terms contains a sizable number of terms, the ontologist builds concept taxonomies to define the concept hierarchy. As explained in Section 3.3.2, any of the three approaches proposed by Uschold and Grüninger (1996) can be used: top-down, bottom-up, and middle-out.

To build concept taxonomies, the ontologist selects terms that are concepts from the glossary of terms. For this, it is really important to identify in the concept taxonomy sets of disjoint concepts, that is, concepts that cannot have common instances. METHONTOLOGY proposes to use the four taxonomic relations defined in the Frame Ontology and the OKBC Ontology: Subclass-Of, Disjoint-Decomposition, Exhaustive-Decomposition, and Partition.

A concept $C_1$ is a Subclass-Of another concept $C_2$ if and only if every instance of $C_1$ is also an instance of $C_2$. For example, as Figure 3.15 illustrates, Iberia Flight is a subclass of Flight, since every flight operated by Iberia is a flight. A concept can be a subclass of more than one concept in the taxonomy. For instance, the concept AA0488 is a subclass of the concepts American Airlines Flight and Iberia Flight, since it is a code-shared flight.

A Disjoint-Decomposition of a concept $C$ is a set of subclasses of $C$ that do not have common instances and do not cover $C$, that is, there can be instances of the concept $C$ that are not instances of any of the concepts in the decomposition. For example, the concepts BA0068, BA0066 and BA0069 make up a disjoint decomposition of the concept British Airways Flight because no flight can be simultaneously a BA0068 flight, a BA0066 flight, and a BA0069 flight. Besides, there may be instances of the concept British Airways Flight that are not instances of any of the three classes. This disjoint decomposition is also shown in Figure 3.15.

![Figure 3.15: An excerpt of the Concept Taxonomy of flights in our travel ontology.](image)

An Exhaustive-Decomposition of a concept $C$ is a set of subclasses of $C$ that cover $C$ and may have common instances and subclasses, that is, there cannot be instances of the concept $C$ that are not instances of at least one of the concepts in the decomposition. For example, the concepts Economy Trip, Business Trip and Luxury Trip make up an exhaustive decomposition of the concept Travel Package because there are no travel packages that are not instances of at least one
of those concepts, and those concepts can have common instances. For example, a business trip can be economic or very expensive. This exhaustive decomposition is shown in Figure 3.16.

A **Partition** of a concept C is a set of subclasses of C that does not share common instances but that covers C, that is, there are not instances of C that are not instances of one of the concepts in the partition. For example, Figure 3.17 shows that the concepts **International Flight** and **Domestic Flight** make up a partition of the concept **Flight** because every flight is either international or domestic.

Once the ontologist has structured the concepts in the concept taxonomy, and before going ahead with the specification of new knowledge, s(he) should examine that the taxonomies contain no errors. In Section 3.8.2 we describe several types of taxonomic errors that can be evaluated: loops in the hierarchy, common instances in a partition, etc.

**Task 3: To build ad hoc binary relation diagrams.** Once the taxonomy has been built and evaluated, the conceptualization activity proposes to build ad hoc binary relation diagrams. The goal of this diagram is to establish ad hoc relationships between concepts of the same (or different) concept taxonomy. Figure 3.18 presents a fragment of the ad hoc binary relation diagram of our travel ontology, with the relations **arrival Place** and **departure Place**, and their inverses **is Arrival Place of** and **is Departure Place of**. Such relations connect the root concepts (**Travel** and **Location**) of the concept taxonomies of travels and locations. From an ontology integration perspective, such ad hoc relations express that the flight ontology will include the location ontology and vice versa.

Before going ahead with the specification of new knowledge, the ontologist should check that the ad hoc binary diagrams have no errors. The ontologist should
figure out whether the domains and ranges of each argument of each relation delimit exactly and precisely the classes that are appropriate for the relation. Errors appear when the domains and ranges are imprecise or over-specified.

![Diagram of Ad Hoc Binary Relations of our travel ontology](image)

**Figure 3.18:** An excerpt of the Diagram of ad hoc Binary Relations of our travel ontology.

**Task 4: To build the concept dictionary.** Once the concept taxonomies and ad hoc binary relation diagrams have been generated, the ontologist must specify which are the properties and relations that describe each concept of the taxonomy in a concept dictionary, and, optionally, their instances.

A concept dictionary contains all the domain concepts, their relations, their instances, and their class and instance attributes. The relations specified for each concept are those whose domain is the concept. For example, the concept Travel has two relations: departure Place and arrival Place. Relations, instance attributes and class attributes are local to concepts, which means that their names can be repeated in different concepts. For example, the attribute name is repeated in the concepts Location and Travel Package. The relation arrival Place is also repeated in the concepts Travel and Travel Package. Table 3.2 shows a small section of the concept dictionary of our travel ontology.

**Table 3.2:** An excerpt of the Concept Dictionary of our travel ontology.

<table>
<thead>
<tr>
<th>Concept name</th>
<th>Class attributes</th>
<th>Instance attributes</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA7462</td>
<td>--</td>
<td>--</td>
<td>same Flight as</td>
</tr>
<tr>
<td>American Airlines Flight</td>
<td>company Name</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>British Airways Flight</td>
<td>company Name</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Five-star Hotel</td>
<td>number of Stars</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Flight</td>
<td>--</td>
<td>name size</td>
<td>is Arrival Place of</td>
</tr>
<tr>
<td>Location</td>
<td>--</td>
<td>--</td>
<td>is Departure Place of</td>
</tr>
<tr>
<td>Lodging</td>
<td>--</td>
<td>price of Standard Room placed in</td>
<td></td>
</tr>
<tr>
<td>Travel</td>
<td>--</td>
<td>arrival Date</td>
<td>arrival Place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>company Name</td>
<td>departure Place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>departure Date</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>return Fare</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>single Fare</td>
<td></td>
</tr>
<tr>
<td>Travel Package</td>
<td>--</td>
<td>budget final Price</td>
<td>arrival Place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name number of Days</td>
<td>departure Place</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>accommodated in travels in</td>
</tr>
<tr>
<td>USA Location</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>
As we said before, once the concept dictionary has been built, the ontologist must describe in detail each of the ad hoc binary relations, class attributes, and instance attributes appearing in it. In addition, the ontologist must describe accurately each of the constants that appear in the glossary of terms. Though METHONTOLOGY does all these tasks, it does not propose a specific order to perform them.

**Task 5: To define ad hoc binary relations in detail.** The goal of this task is to describe in detail all the ad hoc binary relations included in the concept dictionary, and to produce the ad hoc binary relation table. For each ad hoc binary relation, the ontologist must specify its name, the names of the source and target concepts, its cardinality, its inverse relation and its mathematical properties. Table 3.3 shows a section of the ad hoc binary relation table of our travel ontology, which contains the definition of the relations *same Flight as*, *placed in*, and *accommodated in*, and two definitions of the relations *arrival Place* and *departure Place*.

<table>
<thead>
<tr>
<th>Relation name</th>
<th>Source concept</th>
<th>Source card. (Max)</th>
<th>Target concept</th>
<th>Mathematical properties</th>
<th>Inverse relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>same Flight as</td>
<td>Flight</td>
<td>N</td>
<td>Flight</td>
<td>Symmetrical Transitive</td>
<td>--</td>
</tr>
<tr>
<td>placed in</td>
<td>Lodging</td>
<td>1</td>
<td>Location</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>accommodated in</td>
<td>Travel Package</td>
<td>N</td>
<td>Lodging</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>arrival Place</td>
<td>Travel</td>
<td>1</td>
<td>Location</td>
<td>is Arrival Place of</td>
<td>--</td>
</tr>
<tr>
<td>departure Place</td>
<td>Travel</td>
<td>1</td>
<td>Location</td>
<td>is Departure Place of</td>
<td>--</td>
</tr>
<tr>
<td>arrival Place</td>
<td>Travel Package</td>
<td>1</td>
<td>Location</td>
<td>is Departure Place of</td>
<td>--</td>
</tr>
<tr>
<td>departure Place</td>
<td>Travel Package</td>
<td>1</td>
<td>Location</td>
<td>is Departure Place of</td>
<td>--</td>
</tr>
</tbody>
</table>

**Task 6: To define instance attributes in detail.** The aim of this task is to describe in detail all the instance attributes already included in the concept dictionary by means of an instance attribute table. Each row of the instance attribute table contains the detailed description of an instance attribute. Instance attributes are those attributes whose value(s) may be different for each instance of the concept. For each instance attribute, the ontologist must specify the following fields: its name; the concept it belongs to (attributes are local to concepts); its value type; its measurement unit, precision and range of values (in the case of numerical values); default values if they exist; minimum and maximum cardinality; instance attributes, class attributes and constants used to infer values of the attribute; attributes that can be inferred using values of this attribute; formulae or rules that allow inferring values of the attribute; and references used to define the attribute.

Table 3.4 shows a fragment of the instance attribute table of our travel ontology. Some of the previous fields are not shown for the sake of space. This table contains the attributes *price of Standard Room* of a lodging, *departure Date* and *arrival Date* of a travel, etc. The use of measurement units in numerical attributes causes the integration of the *Standard Units* ontology. This is an example of how METHONTOLOGY proposes to integrate ontologies during the
conceptualization activity, and not to postpone the integration to the ontology implementation activity.

Table 3.4: An excerpt of the Instance Attribute Table of our travel ontology.

<table>
<thead>
<tr>
<th>Instance attribute name</th>
<th>Concept name</th>
<th>Value type</th>
<th>Measurement unit</th>
<th>Precision</th>
<th>Range of values</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>budget</td>
<td>Business Trip</td>
<td>Float</td>
<td>Currency Quantity</td>
<td>0.01</td>
<td>1000....3000</td>
<td>(0,1)</td>
</tr>
<tr>
<td>budget</td>
<td>Economy Trip</td>
<td>Float</td>
<td>Currency Quantity</td>
<td>0.01</td>
<td>0....1000</td>
<td>(0,1)</td>
</tr>
<tr>
<td>name</td>
<td>Location</td>
<td>String</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(1,N)</td>
</tr>
<tr>
<td>size</td>
<td>Location</td>
<td>Integer</td>
<td>Square Meters</td>
<td>1</td>
<td>--</td>
<td>(1,1)</td>
</tr>
<tr>
<td>price of Standard Room</td>
<td>Lodging</td>
<td>Float</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(0,1)</td>
</tr>
<tr>
<td>budget</td>
<td>Luxury Trip</td>
<td>Float</td>
<td>Currency Quantity</td>
<td>0.01</td>
<td>--</td>
<td>(0,1)</td>
</tr>
<tr>
<td>arrival Date</td>
<td>Travel</td>
<td>Date</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(0,1)</td>
</tr>
<tr>
<td>company Name</td>
<td>Travel</td>
<td>String</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(0,N)</td>
</tr>
<tr>
<td>departure Date</td>
<td>Travel</td>
<td>Date</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(0,1)</td>
</tr>
<tr>
<td>return Fare</td>
<td>Travel</td>
<td>Float</td>
<td>Currency Quantity</td>
<td>0.01</td>
<td>--</td>
<td>(0,1)</td>
</tr>
<tr>
<td>single Fare</td>
<td>Travel</td>
<td>Float</td>
<td>Currency Quantity</td>
<td>0.01</td>
<td>--</td>
<td>(0,1)</td>
</tr>
<tr>
<td>budget</td>
<td>Travel Package</td>
<td>Float</td>
<td>Currency Quantity</td>
<td>0.01</td>
<td>--</td>
<td>(0,1)</td>
</tr>
<tr>
<td>final Price</td>
<td>Travel Package</td>
<td>Float</td>
<td>Currency Quantity</td>
<td>0.01</td>
<td>--</td>
<td>(0,1)</td>
</tr>
<tr>
<td>number of Days</td>
<td>Travel Package</td>
<td>Integer</td>
<td>days</td>
<td>1</td>
<td>--</td>
<td>(0,1)</td>
</tr>
<tr>
<td>travel Restrictions</td>
<td>Travel Package</td>
<td>String</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(0,1)</td>
</tr>
</tbody>
</table>

Task 7: To define class attributes in detail. The aim of this task is to describe in detail all the class attributes already included in the concept dictionary by means of a class attribute table. Each row of the class attribute table contains a detailed description of the class attribute. Unlike instance attributes, which describe concept instances and take their values in instances, class attributes describe concepts and take their values in the class where they are defined. Class attributes are neither inherited by the subclasses nor by the instances. For each class attribute, the ontologist should fill the following information: name; the name of the concept where the attribute is defined; value type; value; measurement unit and value precision (in the case of numerical values); cardinality; the instance attributes whose values can be inferred with the value of this class attribute; etc.

Table 3.5 shows the class attributes company Name and number of Stars defined in our travel ontology. For example, the class attribute number of Stars takes value 5 in the class Five-stars Hotel. Note that, given that these attributes are not inherited, they are defined in different concepts.

Table 3.5: An excerpt of the Class Attribute Table of our travel ontology.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Defined at concept</th>
<th>Value type</th>
<th>Measurement unit</th>
<th>Precision</th>
<th>Cardinality</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>company Name</td>
<td>American Airlines Flight</td>
<td>String</td>
<td>--</td>
<td>--</td>
<td>(1,1)</td>
<td>AA</td>
</tr>
<tr>
<td>company Name</td>
<td>British Airways Flight</td>
<td>String</td>
<td>--</td>
<td>--</td>
<td>(1,1)</td>
<td>BA</td>
</tr>
<tr>
<td>company Name</td>
<td>Iberia Flight</td>
<td>String</td>
<td>--</td>
<td>--</td>
<td>(1,1)</td>
<td>IB</td>
</tr>
<tr>
<td>number of Stars</td>
<td>Five-star Hotel</td>
<td>Integer</td>
<td>star</td>
<td>1</td>
<td>(1,1)</td>
<td>5</td>
</tr>
<tr>
<td>number of Stars</td>
<td>Four-star Hotel</td>
<td>Integer</td>
<td>star</td>
<td>1</td>
<td>(1,1)</td>
<td>4</td>
</tr>
<tr>
<td>number of Stars</td>
<td>Three-star Hotel</td>
<td>Integer</td>
<td>star</td>
<td>1</td>
<td>(1,1)</td>
<td>3</td>
</tr>
<tr>
<td>number of Stars</td>
<td>Two-star Hotel</td>
<td>Integer</td>
<td>star</td>
<td>1</td>
<td>(1,1)</td>
<td>2</td>
</tr>
<tr>
<td>number of Stars</td>
<td>One-star Hotel</td>
<td>Integer</td>
<td>star</td>
<td>1</td>
<td>(1,1)</td>
<td>1</td>
</tr>
</tbody>
</table>
Task 8: To define constants in detail. The aim of this task is to describe in detail each of the constants defined in the glossary of terms. Each row of the constant table contains a detailed description of a constant. For each constant, the ontologist must specify the following: name, value type (a number, a mass, etc.), value, the measurement unit for numerical constants, and the attributes that can be inferred using the constant. Table 3.6 shows a fragment of the constant table of our travel ontology, where the constant maximum Number of Travelers in a Plane is defined. The attributes that can be inferred with the constant are omitted.

Table 3.6: An excerpt of the Constant Table of our travel ontology.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Value</th>
<th>Measurement unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum Number of Travelers in a Plane</td>
<td>Integer</td>
<td>200</td>
<td>person</td>
</tr>
</tbody>
</table>

Formal axioms and rules are important ontology modeling components in heavyweight ontologies, as we commented in Chapter 1. Formal axioms are logical expressions that are always true and are normally used to specify constraints in the ontology. Rules are generally used to infer knowledge in the ontology, such as attribute values, relation instances, etc. METHONTOLOGY proposes to describe formal axioms and rules in parallel once concepts and their taxonomies, ad hoc relations, attributes, and constants have been defined.

Task 9: To define formal axioms. To perform this task, the ontologist must identify the formal axioms needed in the ontology and describe them precisely. For each formal axiom definition, METHONTOLOGY proposes to specify the following information: name, NL description, the logical expression that formally describes the axiom using first order logic, the concepts, attributes and ad hoc relations to which the axiom refers, and the variables used.

Table 3.7: An excerpt of the Formal Axiom Table of our travel ontology.

<table>
<thead>
<tr>
<th>Axiom name</th>
<th>Train inside Europe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Every train that departs from a European location must arrive at another European location</td>
</tr>
<tr>
<td>Expression</td>
<td>forall(?X,?Y,?Z)</td>
</tr>
<tr>
<td></td>
<td>(<a href="?X">Train Travel</a> and <a href="?X,?Y">departure Place</a> and <a href="?X,?Z">arrival Place</a> and <a href="?Y">European Location</a> -&gt; <a href="?Z">European Location</a>)</td>
</tr>
<tr>
<td>Concepts</td>
<td>Train Travel</td>
</tr>
<tr>
<td></td>
<td>European Location</td>
</tr>
<tr>
<td>Refferred attributes</td>
<td>--</td>
</tr>
<tr>
<td>Ad hoc binary relations</td>
<td>departure Place</td>
</tr>
<tr>
<td></td>
<td>arrival Place</td>
</tr>
<tr>
<td>Variables</td>
<td>?X</td>
</tr>
<tr>
<td></td>
<td>?Y</td>
</tr>
<tr>
<td></td>
<td>?Z</td>
</tr>
</tbody>
</table>
As we have already commented, METHONTOLOGY proposes to express formal axioms in first order logic. Table 3.7 shows a formal axiom in our travel ontology that states that “every train that departs from a European location must arrive at another European location”. The row that corresponds to the referred attributes is empty because the axiom only mentions concepts and relations. The variables used are $?X$ for `Train Travel`, $?Y$ for the `departure Place` and $?Z$ for the `arrival Place`.

**Task 10: To define rules.** Similarly to the previous task, the ontologist must identify first which rules are needed in the ontology, and then describe them in the rule table. For each rule definition, METHONTOLOGY proposes to include the following information: name, NL description, the expression that formally describes the rule, the concepts, attributes and relations to which the rule refers, and the variables used in the expression.

METHONTOLOGY proposes to specify rule expressions using the template `if <conditions> then <consequent>`. The left-hand side of the rule consists of conjunctions of atoms, while the right-hand side of the rule is a single atom.

Table 3.8 shows a rule that states and establishes that “every ship that departs from Europe is arranged by the company Costa Cruises”. This rule would let us infer the company name of a ship given the fact that it departs from Europe. As shown in the figure, the rule refers to the concepts `Ship` and `European Location`, to the attribute `company Name`, and to the relation `departure Place`. The variables used are $?X$ for the `Ship`, and $?Y$ for the `European Location`.

Table 3.8: An excerpt of the Rule Table of our travel ontology.

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Costa Cruises rule</strong></td>
<td>Every ship that departs from Europe is arranged by the company Costa Cruises</td>
</tr>
<tr>
<td><strong>Expression</strong></td>
<td><code>if [European Location](?Y) and Ship(?X) and [departure Place](?X,?Y) </code></td>
</tr>
<tr>
<td></td>
<td><code>then [company Name](?X, &quot;Costa Cruises&quot;)</code></td>
</tr>
<tr>
<td><strong>Concepts</strong></td>
<td>Ship</td>
</tr>
<tr>
<td></td>
<td><code>European Location</code></td>
</tr>
<tr>
<td><strong>Referred attributes</strong></td>
<td>company Name</td>
</tr>
<tr>
<td><strong>Ad hoc binary relations</strong></td>
<td>departure Place</td>
</tr>
<tr>
<td><strong>Variables</strong></td>
<td>$?X$</td>
</tr>
<tr>
<td></td>
<td>$?Y$</td>
</tr>
</tbody>
</table>

**Task 11: To define instances.** Once the conceptual model of the ontology has been created the ontologist might define relevant instances that appear in the concept dictionary inside an instance table. For each instance, the ontologist should define: its name, the name of the concept it belongs to, and its attribute values, if known.

Table 3.9 presents some instances of the instance table of our travel domain:

---

7 We suppose that travel agencies work only with travel, accommodation, etc., in Europe and America.
AA7462_Feb08_2002 and AA7462_Feb16_2002). Both of them are instances of the concept American Airlines Flight, as defined in the concept dictionary, and they have some attribute and relation values specified: company Name, departure Date, arrival Date, and single Fare. These instances could have more than one value for the attributes whose maximum cardinality is higher than one.

Table 3.9: An excerpt of the Instance Table of our travel ontology.

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Concept Name</th>
<th>Attribute</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA7462_Feb08_2002</td>
<td>AA7462</td>
<td>company Name</td>
<td>American Airlines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>departure Date</td>
<td>02/08/2002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arrival Date</td>
<td>02/08/2002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>single Fare</td>
<td>300</td>
</tr>
<tr>
<td>AA7462_Feb16_2002</td>
<td>AA7462</td>
<td>company Name</td>
<td>American Airlines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>departure Date</td>
<td>02/16/2002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arrival Date</td>
<td>02/16/2002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>single Fare</td>
<td>300</td>
</tr>
</tbody>
</table>

Embedded in the conceptualization activity itself, there are a series of controls for verifying that each IR is used correctly and that the knowledge represented is valid, that is, the semantics of the conceptualized knowledge is what it should be. For example, in the formal axiom expression that appears in Table 3.7, we use the attribute arrival Place, applied to an instance of the concept Train Travel. One of the verifications proposed by the conceptualization activity is to check that this attribute can be applied to that instance, which is true, since Train Travel is a subclass of the concept Travel, in which this attribute is defined.

It is important to mention here that different domain ontologies may have different knowledge representation needs. METHONTOLOGY suggests reducing or extending the set of intermediate representations according to the KR needs in a domain as well as modifying the fields of the intermediate representations by adding, removing or changing some of the fields previously presented. For instance, if an ontologist is building a lightweight ontology with only concepts, attributes and relations between concepts, (s)he does not need to use the intermediate representations that model rules and formal axioms.

METHONTOLOGY has been used at UPM to build the following ontologies:

- **Chemicals** (Fernández-López, 1996; Gómez-Pérez et al., 1996; Fernández-López et al., 1999), which contains knowledge within the domain of chemical elements and crystalline structures, as presented in Chapter 2. OntoGeneration (Aguado et al., 1998) is a system that uses the Chemicals ontology and the linguistic ontology GUM (Bateman et al., 1995) to generate Spanish text descriptions in response to queries of the Chemistry domain.

- **Monatomic Ions** (Rojas, 1998), which gathers information about monatomic ions, as explained in Chapter 2.

- **Environmental Pollutant** ontologies (Mohedano, 2000). They represent methods to detect pollutant components in various media: water, air, soil, etc., and the maximum concentrations of these components permitted taking into account the legislation in force. This ontology was also commented on Chapter 2.
• *Silicate* (Pinilla-Ponz, 1999). It models properties in the domain of minerals, paying special attention to silicates.

• The *Reference Ontology* (Arpírez et al., 1998) and the *OntoRoadMap* ontology, which extends the former. They are ontologies in the domain of ontologies that play the role of ontology yellow pages. They provide features for describing ontologies, ontology development methodologies, tools and languages for implementing ontologies, ontology-based applications, ontology events (conferences, workshops, etc.), etc. Two applications use these ontologies: *(Onto)*\(^2\)Agent (Arpírez et al., 2000) and the OntoRoadMap application\(^8\). *(Onto)*\(^2\)Agent is an ontology-based WWW broker about ontologies that uses the Reference-Ontology as a knowledge source and retrieves descriptions of ontologies that satisfy a given set of constraints. The OntoRoadMap application (developed as an evolution of *(Onto)*\(^2\)Agent) uses the *OntoRoadMap* ontology. It permits users to introduce, access and modify information about ontologies, ontology tools, ontology methodologies and ontology-based applications. It was developed in the framework of the OntoWeb thematic network\(^9\).

• The restructured version of the *(KA)*\(^2\) ontology (Blázquez et al., 1998), which contains knowledge about the scientific community in the field of Knowledge Acquisition, particularly scientists, research topics, projects, universities, etc. Also described in Chapter 2.

• *Knowledge management* ontologies (KM-LIA), which provide vocabulary in the domains of people, publications, lessons learned, hardware and software, etc., of the Laboratory of Artificial Intelligence.

• Ontologies developed at the MKBEEM project\(^10\) (Léger et al., 2000) on the domains of traveling, cloth catalogues, and lodging, which have been used in a multilingual platform for e-commerce.

• Ontologies for describing R&D projects and their structure, documents generated in R&D projects, persons and organizations participating in them, and meetings (administrative, technical, etc.) held during a project. The Esperonto project\(^11\) uses these ontologies on the Esperonto knowledge portal.

### 3.3.6 SENSUS-based method

In this section, we present a method for building the skeleton of the domain ontology starting from a huge ontology, the SENSUS ontology, described in Section 2.3.5. The method proposes to link domain specific terms to the huge ontology and to prune, in the huge ontology, those terms that are irrelevant for the new ontology we wish to build.

The result of this process is the skeleton of the new ontology, which is generated automatically using the process described below and the OntoSaurus tool described in Chapter 5.

---

8 [http://babage.dia.fi.upm.es/ontoweb/WP1/OntoRoadMap/index.html](http://babage.dia.fi.upm.es/ontoweb/WP1/OntoRoadMap/index.html)
According to this method, to build an ontology in a specific domain the following processes should be taken:

**Process 1: To identify seed terms** (key domain terms). For example, if we wish to build an ontology on flights we could take the terms presented in Figure 3.19 as seed: Europe–Africa flight, Europe–America flight, London–Liverpool flight and Madrid–Barcelona flight.

![Figure 3.19: Process 1. Identify seed terms.](image)

**Process 2: To link manually the seed terms to SENSUS.** As Figure 3.20 illustrates, the first two types of flight of the example are linked to SENSUS as hyponyms (subclasses) of international flight, and the other terms are linked as hyponyms of domestic flight. To perform this example we have used OntoSaurus.

**Process 3: To add paths to the root.** All the concepts in the path, from the seed terms to the root of SENSUS, are included. OntoSaurus provided the set of concepts presented in Figure 3.21.

**Process 4: To add new domain terms.** In this process, terms that could be relevant within the domain and that have not yet appeared are manually added. Then processes 2 and 3 are repeated to include concepts in the path going from the new terms to the root of SENSUS. In our example, we can add the terms origin and destination, as shown in Figure 3.22. We should also add the ascendants of the new identified terms.

**Process 5. To add complete subtrees.** In this process, the ontologist has to pay special attention to those nodes that have a large number of paths through them in the tree now generated. Then, for the subtree under each of these nodes, the ontologist should decide whether to add a node or not. The criterion taken into account in this process is that if many of the nodes in a subtree have been found to be relevant, then the other nodes in the subtree are likely to be relevant, too. As Figure 3.23 shows, we could add the entire subtree of flight trip and then the concepts redeye (night flight) and nonstop flight (a flight without intermediate landings between source and destination).

This process is done manually, since it seems that the ontologist requires some understanding of the domain to take the decision of adding new nodes. Obviously, very high level nodes in the ontology will always have many paths through them, but it is not appropriate to include the entire subtrees under these nodes.
Figure 3.20: Process 2. To link the seed terms to SENSUS.

Figure 3.21: Process 3. To add paths to the root.
Figure 3.22: Process 4. To add new domain terms.

Figure 3.23: Process 5. To add complete subtrees.
The approach adopted in SENSUS promotes knowledge shareability as the same base ontology is used to develop ontologies in specific domains. A major advantage of this approach is that if two ontologies are developed independently, the broad coverage of the ontology (SENSUS) acts as a “hinge” that couples the terminology and organization of one ontology with the other (Swartout et al., 1997).

SENSUS has also been applied for building an ontology for military air campaign planning. It contains an overview of the basic elements that characterize air campaign plans such as campaign, scenario, participants, commanders, etc. (Valente et al., 1999). It also includes ontologies of weapons, systems in general, fuel, etc.

3.3.7 On-To-Knowledge

The aim of the On-To-Knowledge project (Staab et al., 2001) is to apply ontologies to electronically available information for improving the quality of knowledge management in large and distributed organizations. Some of the partners of this project are the Institute AIFB of the University of Karlsruhe, the Vrije Universiteit of Amsterdam, and British Telecom. In this project, they developed a methodology and tools for intelligent access to large volumes of semi-structured and textual information sources in intra-, extra-, and internet-based environments. The methodology includes a methodology for building ontologies to be used by the knowledge management application. Therefore, the On-To-Knowledge methodology for building ontologies proposes to build the ontology taking into account how the ontology will be used in further applications. Consequently, ontologies developed with this methodology are highly dependent of the application. Another important characteristic is that On-To-Knowledge proposes ontology learning for reducing the efforts made to develop the ontology.

The methodology also includes the identification of goals to be achieved by knowledge management tools, and is based on an analysis of usage scenarios (Staab et al., 2001).

We consider On-To-Knowledge as a methodology because it has a set of techniques, methods, principles for each of its processes, and because it indicates the relationships between such processes (e.g., recommended order, interleaving, input/outputs). The processes proposed by this methodology are shown in Figure 3.24.

Process 1: Feasibility study. On-To-Knowledge adopts the kind of feasibility study described in the CommonKADS methodology (Schreiber et al., 1999). According to On-To-Knowledge, the feasibility study is applied to the complete application and, therefore, should be carried out before developing the ontologies. In fact, the feasibility study serves as a basis for the kickoff process.

Process 2: Kickoff. The result of this process is the ontology requirements specification document that describes the following: the domain and goal of the ontology; the design guidelines (for instance, naming conventions); available
knowledge sources (books, magazines, interviews, etc.); potential users and use cases as well as applications supported by the ontology.

Competency questions, discussed in Section 3.3.3, can be useful to elaborate the requirements specification document. The requirement specification should lead the ontology engineer to decide about the inclusion or exclusion of concepts in the ontology, and about their hierarchical structure. In fact, this specification is useful to elaborate a draft version containing few but seminal elements. This first draft is called “baseline ontology”. The most important concepts and relations are identified on an informal level.

In the kickoff process the developers should look for potentially reusable ontologies already developed.

**Process 3: Refinement.** The goal here is to produce a mature and application-oriented “target ontology” according to the specification given in the kickoff process. This refinement process is divided into two activities:

**Activity 1: Knowledge elicitation process with domain experts.** The baseline ontology, that is, the first draft of the ontology obtained in process 2, is refined by means of interaction with experts in the domain. When this activity is performed, axioms are identified and modeled. During the elicitation, the concepts are gathered on one side and the terms to label the concepts on the other. Then, terms and concepts are mapped. The On-To-Knowledge methodology proposes the use of intermediate representations to model the knowledge. In this aspect, it follows METHONTOLOGY’s basic ideas. If several experts participate in the building of the ontology, it is necessary to reach an agreement.
A complementary way to enrich the ontology is to use it as seed in an ontology learning process, as explained in Section 3.5.

Activity 2: Formalization. The ontology is implemented using an ontology language. Such language is selected according to the specific requirements of the envisaged application. To carry out the formalization, On-To-Knowledge recommends the OntoEdit ontology editor, which generates automatically the ontology code in several languages. Other ontology editors that perform similar functions can be also used, as described in Chapter 5.

Process 4: Evaluation. The evaluation process serves as a proof of the usefulness of the developed ontologies and their associated software environment. The product obtained is called ontology based application. During this process two activities are carried out:

Activity 1: Checking the requirements and competency questions. The developers check whether the ontology satisfies the requirements and “can answer” the competency questions.

Activity 2: Testing the ontology in the target application environment. Further refinement of the ontology can arise in this activity.

This evaluation process is closely linked to the refinement process. In fact, several cycles are needed until the target ontology reaches the envisaged level.

Process 5: Maintenance. It is important to clarify who is responsible for the maintenance and how this should be carried out. On-To-Knowledge proposes to carry out ontology maintenance as part of the system software.

On-To-Knowledge has been adopted to build virtual enterprises, to organize corporate memories, and to help desk functionality in call centers. This last application provides customers with appropriate information on products and services and decides about the ability of a company to establish successful and stable relationships with their clients.

3.3.8 Comparing ontology development methods and methodologies

Which method or methodology is the most useful to build your ontology? To help you answer this question we have elaborated a framework to compare the following methods and methodologies: the Cyc method, Uschold and King’s method, Grüninger and Fox’s methodology, the KACTUS method, METHONTOLOGY, the SENSUS method, and the On-To-Knowledge methodology.

3.3.8.1 Comparison framework
The comparison framework takes into account the ontology construction strategy of methodologies and methods, their software support, the ontology development processes that they propose, and how they have been used to develop ontologies or in applications, projects, etc. This framework is based on the one presented by Fernández-López and Gómez-Pérez (2002a).
To compare the *construction strategy* of methods and methodologies we propose the following set of criteria, summarized in Table 3.10:

- **Life cycle proposal.** The life cycle identifies the set of stages through which the ontology moves during its life time. It also describes which activities have to be performed at each stage and how the stages are related, such as relations of precedence, simultaneity, etc. Examples of life cycles are:
  - Incremental life cycle (McCracken and Jackson, 1982). According to this approach, the ontology would grow by layers, allowing the inclusion of new definitions only when a new version is planned. This model prevents the inclusion of new definitions if they are not planned, but it does permit an incremental development.
  - Evolving prototypes (Kendall and Kendall, 1995). According to this approach, the ontology grows according to the needs. This model permits modifying, adding, and removing definitions in the ontology at any time.
- **Strategy according to the application.** This criterion is related to the degree of dependency of the ontology with the application using it. Considering this criterion, the methodologies and methods can be classified into the following types:
  - Application dependent. Ontologies are built on the basis of the applications that use them.
  - Application semi-dependent. The possible scenarios of ontology use are identified in the specification stage.
  - Application-independent. The process is totally independent of the uses of the ontology in applications.
- **Use of core ontologies.** In this criterion we analyze whether it is possible or not to use a core ontology as a starting point in the development of the domain ontology.
- **Strategy to identify concepts.** There are three possible strategies for identifying concepts: from the most concrete to the most abstract (bottom-up), from the most abstract to the most concrete (top-down), or from the most relevant to the most abstract and most concrete (middle-out). A more precise explanation of this criterion can be found in Section 3.3.2, where we described Uschold and King’s method.

Concerning the *technological support*, it is important to know which tools give full or partial support to the methodologies and methods. Table 3.11 enumerates these tools and tool suites.

Table 3.12 summarizes the *ontology development processes* in all the methods and methodologies. Each cell of the table can be filled with three types of values. The value ‘described’ means that the method or methodology describes how to perform each task in the proposed activity, when to do it, who has to do it, etc. The value ‘proposed’ means that the methodology just identifies the process. The value ‘NP’ means that public documentation does not mention the non-considered activity.
Table 3.10: Summary of the construction strategies.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cyc</th>
<th>Uschold &amp; King</th>
<th>Grüninger &amp; Fox</th>
<th>KACTUS</th>
<th>METHONTOLOGY</th>
<th>SENSUS</th>
<th>On-To-Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life cycle proposal</td>
<td>Evolving prototypes</td>
<td>Non-proposed</td>
<td>Evolving prototypes or incremental</td>
<td>Evolving prototypes</td>
<td>Evolving prototypes</td>
<td>Non-proposed</td>
<td>Incremental and cyclic with evolving prototypes</td>
</tr>
<tr>
<td>Strategy with respect to the application</td>
<td>Application independent</td>
<td>Application independent</td>
<td>Application semi-dependent</td>
<td>Application independent</td>
<td>Application semi-dependent</td>
<td>Application dependent</td>
<td>Application dependent</td>
</tr>
<tr>
<td>Strategy to identify concepts</td>
<td>Not specified</td>
<td>Middle-out</td>
<td>Middle-out</td>
<td>Top-down</td>
<td>Middle-out</td>
<td>Not specified</td>
<td>Top-down Bottom-up Middle-out</td>
</tr>
<tr>
<td>Use of a core ontology</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Depends on the available resources</td>
<td>Yes</td>
<td>Depends on the available resources</td>
</tr>
</tbody>
</table>

Table 3.11: Technological support of the approaches.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cyc</th>
<th>Uschold &amp; King</th>
<th>Grüninger &amp; Fox</th>
<th>KACTUS</th>
<th>METHONTOLOGY</th>
<th>SENSUS</th>
<th>On-To-Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tools that give support</td>
<td>Cyc tools</td>
<td>No specific tool</td>
<td>No specific tool</td>
<td>No specific tool</td>
<td>ODE WebODE OntoEdit Protégé-2000</td>
<td>No specific tool (usually OntoSaurus)</td>
<td>OntoEdit with its plug-ins.</td>
</tr>
</tbody>
</table>
Table 3.12: Summary of the ontology development process.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cyc</th>
<th>Uschold &amp; King</th>
<th>Grüninger &amp; Fox</th>
<th>KACTUS</th>
<th>METHONTOLOGY</th>
<th>SENSUS</th>
<th>On-To-Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology management activities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduling</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>Proposed</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Control</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>Proposed</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Quality assurance</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>Proposed</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Pre-development processes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment study</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>Proposed</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Feasibility study</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>Proposed</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Development processes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>NP</td>
<td>Proposed</td>
<td>Described in detail</td>
<td>Proposed</td>
<td>Described in detail</td>
<td>Proposed</td>
<td>Described in detail</td>
</tr>
<tr>
<td>Conceptualization</td>
<td>NP</td>
<td>NP</td>
<td>Described in detail</td>
<td>Proposed</td>
<td>Described in detail</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Formalization</td>
<td>NP</td>
<td>NP</td>
<td>Described in detail</td>
<td>Described</td>
<td>Described</td>
<td>NP</td>
<td>Described</td>
</tr>
<tr>
<td>Implementation</td>
<td>Proposed</td>
<td>Proposed</td>
<td>Described</td>
<td>Proposed</td>
<td>Described in detail</td>
<td>Described</td>
<td>Described</td>
</tr>
<tr>
<td>Post-development processes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintenance</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>Proposed</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Use</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>Proposed</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Knowledge acquisition</td>
<td>Proposed</td>
<td>Proposed</td>
<td>Proposed</td>
<td>NP</td>
<td>Described in detail</td>
<td>NP</td>
<td>Described</td>
</tr>
<tr>
<td>Evaluation</td>
<td>NP</td>
<td>Proposed</td>
<td>Described in detail</td>
<td>NP</td>
<td>Described in detail</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Integration</td>
<td>Proposed</td>
<td>Proposed</td>
<td>Proposed</td>
<td>Proposed</td>
<td>Proposed</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Configuration management</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>Described</td>
<td>NP</td>
<td>Proposed</td>
</tr>
<tr>
<td>Documentation</td>
<td>Proposed</td>
<td>Proposed</td>
<td>Proposed</td>
<td>NP</td>
<td>Described in detail</td>
<td>NP</td>
<td>Described</td>
</tr>
<tr>
<td>Merging and Alignment</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>Proposed</td>
<td>NP</td>
<td>NP</td>
</tr>
</tbody>
</table>

12 Each On-To-Knowledge description consists of (a) an explanation of the general ideas of how to carry out the activity; (b) references to authors that have written about such an activity; and (c) the activity software support.

13 Cyc proposes the integration of different micro-theories.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Cyc</th>
<th>Uschold &amp; King</th>
<th>Grüninger &amp; Fox</th>
<th>KACTUS</th>
<th>METHONTOLOGY</th>
<th>SENSUS</th>
<th>On-To-Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projects where they have been used</td>
<td>High Performance Knowledge Bases (HPKB)</td>
<td>Enterprise Project</td>
<td>TOVE</td>
<td>KACTUS</td>
<td>MKBEEM, OntoWeb, Esperonto, ContentWeb, Environmental Ontologies.</td>
<td>Military air campaign planning (DARPA project)</td>
<td>On-To-Knowledge, OntoWeb, SemiPort, AIFB Website, COST action</td>
</tr>
<tr>
<td>Ontologies created</td>
<td>Cyc</td>
<td>Enterprise Ontology</td>
<td>TOVE</td>
<td>Electrical network ontologies</td>
<td>- Chemicals, Monatomic Ions, Silicate and Environmental Pollutants - Reference and OntoRoadMap - (KA)^3 ontologies - KM ontologies - MKBEEM ontologies - Esperonto ontologies</td>
<td>Military air campaign planning ontologies</td>
<td>- Skills Management - Virtual Organization - OntoShare - OntoWeb Portal - AIFB Portal</td>
</tr>
<tr>
<td>Ontology domains</td>
<td>Cyc’s different micro-theories</td>
<td>Enterprise</td>
<td>Enterprise</td>
<td>Electrical network</td>
<td>- Chemical and Environment - Ontologies - Knowledge management - Computer Science - Travel - etc.</td>
<td>Military air campaign</td>
<td>See above</td>
</tr>
<tr>
<td>Applications that use such ontologies</td>
<td>- Heterogeneous DB integration - Searching of Captioned Information - Thesauri Integration</td>
<td>Enterprise Toolset</td>
<td>Enterprise Design Workbench Supply Chain Management Project</td>
<td>Diagnosis and service resumption in electrical networks</td>
<td>- OntoGeneration - OntoRoadMap - (Onto)^3 Agent - ODEClean - MKBEEM prototype - Esperonto Web Site - ODESeW</td>
<td>Unknown</td>
<td>See above</td>
</tr>
</tbody>
</table>
Finally, and in relation with the use of ontologies built according to a method or methodology, we have analyzed the use of the ontologies in projects; the acceptance of the methodology or method by other groups different to the one that has elaborated it; the ontologies developed following the methodology or method; the domains of such ontologies; and the systems where they have been used. Table 3.13 summarizes all these features.

3.3.8.2 Conclusions
The conclusions reached from the analysis of the tables are:

- The prevailing life cycle model is evolving prototypes, although there are approaches that do not propose any life cycle model.
- There is a wide diversity of ontology development strategies. Some approaches consider the application-dependent strategy, others the application-semidependent, and others the application-independent.
- There is also a wide diversity of strategies to identify concepts in the taxonomy, although the middle-out approach is the most commonly used.
- None of the approaches covers all the processes involved in ontology building. However, the following scale can be established between the methodologies and methods presented, from the most to the least complete.
  - METHONTOMETRY is the approach that provides the most accurate descriptions of each activity.
  - The On-To-Knowledge methodology describes more activities than the other approaches.
  - The strength of Grüninger and Fox’s methodology is its high degree of formality. However, the ontology life cycle is not completely specified. Several ontologies have been developed with this methodology and some applications use these ontologies, but the methodology has only been tested on the business domain.
  - Uschold and King’s method has the same omissions as Grüninger and Fox’s and it is less detailed.
  - The SENSUS-based method does not mention the ontology life cycle and it has some of the shortcomings of the above approaches. However, it easily allows generating skeleton ontologies from huge ontologies.
  - Bernarbas and colleagues’ method has been used to build only a few ontologies and applications and it has also the aforementioned omissions.
- Most of the approaches are focused on development activities, specially on the ontology implementation, and they do not pay too much attention to other important aspects related to the management, evolution and evaluation of ontologies. This is due to the fact that the ontological engineering field is relatively new. However, a low compliance with the criteria formerly established does not mean a low quality of the methodology or method. As de Hoog (1998) states, a not very specified method can be very useful for an experienced group.
- Most of the approaches present some drawbacks in their use. Some of them have not been used by external groups and, in some cases, they have been used in a single domain.
Most of the approaches do not have a specific tool that gives them technology support. Besides, none of the available tools covers all the activities necessary in ontology building.

### 3.4 Method for Re-engineering Ontologies

Ontologists can come across an ontology that might be reused in the ones that they are developing. It might happen, however, that this ontology has not been developed according to the modeling criteria and the ontological commitments that the ontologists are following. It might also be possible that the domain experts do not completely agree with the contents of the ontology to be reused. Or it might happen that the conceptual model of the ontology, or even its design, is missing and the only thing available is the ontology implementation in a formal language. In all these cases, a re-engineering process is needed.

Ontological re-engineering (Gómez-Pérez and Rojas, 1999) is defined as the process of retrieving a conceptual model of an implemented ontology and transforming this conceptual model to a more suitable one, which is then re-implemented in the same language or in another ontology language. Figure 3.25 presents the proposal of a method for re-engineering ontologies, which adapts Chikofsky and Cross II’s (1990) software re-engineering schema to the ontology domain. This adaptation is an extension of METHONTOLOGY for ontology re-engineering.

![Figure 3.25: Ontological re-engineering process.](image)

Three main activities are identified in the ontology re-engineering process:

**Activity 1. Reverse Engineering.** Its objective is to derive the ontology conceptual model from its implementation code. To build this conceptual model, the set of intermediate representations proposed by METHONTOLOGY might be used, as well as any ontology editor able to import ontologies in that specific language or format too.

For example, suppose that during the construction of our travel agency ontology we want to reuse an ontology of places written in Ontolingua, as shown in Figure 3.26. If we do not have the ontology conceptualization, we must obtain it from its Ontolingua code.
Activity 2. Restructuring. The goal here is to reorganize the initial conceptual model into a new one. The restructuring activity is carried out in two phases, analysis and synthesis. The analysis phase includes evaluation, whose general purpose is to evaluate the ontology from a technical point of view (Gómez-Pérez et al., 1995). The synthesis phase seeks to correct the ontology after the analysis phase and to document all the changes made. Hence, activities related to configuration management have appeared in this context. The aim of configuration management is to keep a record of the ontology evolution as well as a strict change control. Figure 3.27 shows an example of a control report of a change made in our travel ontology.

| Description of the Change: | Modify the taxonomy shown in Figure 3.26 as all the classes are subclasses of the root class. |
| Need for the Change: | It is advisable to create an intermediate layer with concepts that distinguish stations and accommodations. |
| Effects of the Change: | The taxonomy has been modified and two new classes are included. |
| Alternatives: | The only alternative is the one shown in Figure 3.28. |
| Date of change: | 03/03/2002. |
| Changes made: | Changes are shown in Figure 3.28. |

In our example, a problem in the ontology structure has been detected during the analysis phase: there is no taxonomic organization identifying intermediate
Ontological Engineering

concepts between the concept Place and the next level of the concept taxonomy. That is, all the stations and all the accommodations are at the same level. In the synthesis phase, we can modify the conceptual model of the ontology of locations through a specialization of concepts. The objective here is to create general concepts that are superclasses of existing concepts in the old taxonomy. For example, the concepts Bus station and Train station can be defined as subclasses of the concept Station, and the concepts Hotel, House and Shelter can be defined as subclasses of the concept Accommodation, as shown in Figure 3.28.

Figure 3.28: Taxonomy of places restructured.

Activity 3. Forward Engineering. The aim of this activity is to output a new ontology implementation on the basis of the new conceptual model. For example, if we need the ontology in OWL, we can implement the new conceptual model automatically with many of the ontology tools described in Chapter 5.

We tested this re-engineering method when we developed the environmental ontologies described in Section 2.4.5, because we had to re-engineer the Standard-Units ontology of the Ontolingua Server, as presented by Gómez-Pérez and Rojas (1999).

We also applied this method to re-engineer ontologies in the context of the IST project MKBEEM (Léger et al., 2000). The MKBEEM ontologies were developed by different groups and with two different ontology tools: CONE (Kankaanpää, 1999), a frame-based ontology editor, and WebODE (Arpírez et al., 2003), based on a combination of frames and first order logic as will be described in Chapter 5. These ontologies had to be transformed into the description logic language CARIN (Levy and Rousset, 1998) because we were interested in using the inference services provided by PICSEL (Goasdoué et al., 2000), which worked with that language. To exchange ontologies between both tools we decided to use an XMLization of CARIN, called X-CARIN. Figure 3.29 shows how ontologies developed with CONE were transformed into X-CARIN and then imported through a reverse engineering process into WebODE. There, the ontologies were re-engineered, integrated with the other ontologies, and transformed back into X-CARIN using WebODEs X-CARIN translator.
3.5 Ontology Learning Methods

Although one of the main purposes of ontologies is to reduce the effort during the knowledge acquisition process, acquiring knowledge for building an ontology from scratch, or for refining an existing ontology requires much time and many resources. There are, however, several approaches for the partial automatization of the knowledge acquisition process. This automatization can be carried out by means of natural language analysis and machine learning techniques. Alexander Maedche and Steffen Staab (2000a) distinguish the following ontology learning approaches:
1) ** Ontology learning from texts. ** Ontology learning from texts is based on the use of text corpora. A *corpus of texts* is a set of texts that should be representative of the domain (complete), prepared to be processed by a computer, and accepted by the domain experts (Enery and Wilson, 2001). To understand the different methods and techniques identified by Maedche and colleagues for ontology learning from texts, it is important to distinguish between the linguistic level and the conceptual level. The main characteristic of the linguistic level is that knowledge is described through linguistic terms, while in the conceptual level knowledge is described through concepts and relations between concepts (Brachman, 1979; Guarino, 1994). According to this distinction, the learnt concepts and relations are at the conceptual level, and the texts used during the learning are, obviously, at the linguistic level. Therefore, the different techniques of ontology learning from texts are based on how linguistic level structures are ‘projected’ on the conceptual level. These kind of techniques are classified into:

1.1) **Pattern-based extraction** (Hearst, 1992). Relations at the conceptual level are recognized from sequences of words in the text that follow a given pattern.

   For example, in English a pattern can establish that if a sequence of \( n \) names is detected, then the \( n-1 \) first names are hyponyms of the \( n^{th} \). According to this pattern, the term *Spain location* could be used to obtain the hyponymy relationship between the term *Spain location* and the term *location*. This relation at the linguistic level is projected on the conceptual level as the *Subclass-Of* relation between the concept associated to the term *Spain location* and the concept associated to the term *location*.

   This kind of technique is applied in the ontology field by Maedche and colleagues (Kietz et al., 2000), as well as by Aussenac-Gilles and colleagues (2000a; 2000b).

1.2) **Association rules.** Association rules were initially defined on the database field as follows (Agrawall et al., 1993): “*Given a set of transactions, where each transaction is a set of literals (called items), an association rule is an expression of the form X implies Y, where X and Y are sets of items. The intuitive meaning of such a rule is that transactions of the database which contain X tend to contain Y*”. Association rules are used on the data mining process to discover information stored in databases provided that we have a rough idea of what we are looking for (Adriaans and Zantinge, 1996).

   In the ontology field (Maedche and Staab, 2000b) association rules have been used to discover non-taxonomic relations between concepts, using a concept hierarchy as background knowledge, and statistics of co-occurrences of terms in texts. Thus, for example, if the word *train* frequently co-occurs with the word *travel* in the texts to be analyzed, then we could add to the ontology a relation between the concept associated to *train* and the concept associated to *travel*. 
1.3) *Conceptual clustering* (Michalsky, 1980; Faure et al., 2000). A set of concepts are taken as input and then are grouped according to the semantic distance between them. A way to group concepts is to consider that two concepts belong to the same group if their semantic distance is lower than a predefined threshold. A way to calculate the distance between concepts is based on the use of the syntactical functions that the terms associated to such concepts play in the text.

For example, if *train* and *car* appear with the same syntactical function (e.g., the subject) in sentences with the same verb (e.g., *Peter travels by train, John travels by car*, etc.), then the concepts associated to *train* and *car* are considered semantically as close concepts and should be grouped.

Conceptual clustering is explicitly proposed in the two ontology learning methods that we will present in this section.

1.4) *Ontology pruning* (Kietz et al., 2000). The objective of ontology pruning is to elicit a domain-ontology using a core ontology and a corpus, or even several corpora of texts. A method that uses ontology pruning is presented in Section 3.5.1.

1.5) *Concept learning* (Hahn and Schulz, 2000). A given taxonomy is incrementally updated as new concepts are acquired from real-world texts. To update the taxonomy, techniques already considered in other approaches (*pattern based extraction, ontology clustering*, etc.) could be applied. A method that uses concept learning is also presented in Section 3.5.1.

2) *Ontology learning from instances* (Morik and Kietz, 1989). Particular instances, for example, those taken from files, are used to generate the ontology.

3) *Ontology learning from schemata*. Relational database schemas, Entity/Relationship models, and XML schemas can be used to generate ontologies by a re-engineering process.

4) *Ontology learning for interoperability*. Semantic mappings between similar elements from two ontologies are learnt. The learned mappings can be used for ontology merging, and also when a system needs to operate with the two ontologies.

Up to now, the most well-known ontology learning methods use techniques that belong to the category of ontology learning from texts. Two of these methods will be presented in this section: Maedche and colleagues’ and Aussenac-Gilles and colleagues’.
3.5.1 Maedche and colleagues’ method

Maedche and colleagues’ method (Kietz et al., 2000) assumes that documents from a domain describe most of the domain concepts and relations to be included in an ontology as well as the domain terminology. This method proposes to learn the ontology using as a base a core ontology (SENSUS, WordNet, etc.), which is enriched with the learned concepts.

New concepts are identified using natural language analysis techniques over the resources previously identified by the user. The resulting ontology is pruned and then focused on a specific domain by means of several approaches based on statistics. Finally, relations between concepts are established applying learning methods. Such relations are added to the resulting ontology.

Figure 3.30: Activities followed in Maedche and colleagues’ method for ontology learning.

Figure 3.30 shows the five activities proposed in this method:

- **Activity 1. Select sources.** In this method, sources are either ontologies or documents. The process starts with the selection of a core ontology, which is used as a base in the learning process. This ontology should contain generic and domain specific concepts. In this first activity the ontologist should specify which documents should be used in the steps to follow to refine and extend the previous ontology. By its own nature, documents are heterogeneous in their formats and contents. They can be free text, semi-structured text, domain text, and generic text documents. Documents will make up two corpora: one with domain specific terms, and another with general terms.
**Activity 2. Concept learning.** Its goal is to acquire new generic and domain specific concepts. Both types of concepts are extracted from texts by means of mainly natural language processing (NLP) tools that use pattern-based extraction and conceptual clustering. The selection of the tools depends on the languages to be processed (Spanish, English, German, etc.). The method suggests linking the learned concepts to the core ontology using, above all, the Subclass-Of relation.

**Activity 3. Domain focusing.** Its purpose is to prune the enriched core ontology and remove general concepts. The results of the analysis of the term frequency in the generic and the specific corpora are used to prune the ontology. The terms appearing more frequently in the domain-specific corpus than in the generic corpus should be proposed to the ontologist for deciding whether they should be kept in the whole enriched ontology.

**Activity 4. Relation learning.** ad hoc relations between concepts of the domain are learnt by means of pattern-based extraction and association rules.

**Activity 5. Evaluation.** Its goal is to evaluate the resulting ontology (the core ontology enriched and pruned in the previous activities) and to decide whether it is necessary to repeat the process again.

This method is supported by the tool Text–To–Onto (Maedche and Volz, 2001), and has been applied inside the project On-To-Knowledge, whose methodology was described in Section 3.3.7.

### 3.5.2 Aussenac-Gilles and colleagues’ method

This is a method for ontology learning based on knowledge elicitation from technical documents, as described by Aussenac-Gilles and colleagues (2000a, 2000b). The method allows creating a domain model by analyzing a corpus with NLP tools. The method combines knowledge acquisition tools based on linguistics with modeling techniques to keep links between models and texts. During the learning process, it is assumed that (Aussenac-Gilles et al., 2002):

1. The ontology builder should have a comprehensive knowledge about the domain, so that (s)he will be able to decide which terms (nouns, phrases, verbs or adjectives) are domain terms and which concepts and relations are labeled with these domain terms.
2. Concerning the output, a similar implicit assumption is that the ontology builder knows well how the ontology will be used.

After selecting a corpus, the method proposes to obtain linguistic knowledge (terms, lexical relations, and groups of synonyms) at the linguistic level. This linguistic knowledge is then transformed into a semantic network in the normalization activity. The semantic network includes concepts, relationships between concepts and attributes for the concepts, and is implemented in the formalization activity. Figure 3.31 illustrates the activities proposed by the method:
Activity 1. Corpus constitution. Texts are selected from among the technical documentation available taking into account the ontology requirement specification document. The authors recommend that the selection of texts be made by an expert in texts of the domain, and that the corpus should be complete, that is, it should cover the entire domain specified by the application. To perform this activity it is very useful to have a glossary of the domain terms. Thus, the expert selects texts containing the terms of the glossary. If we compare this activity with the first activity proposed by Maedche and his colleagues, we can observe that they differ in: (1) the kind of sources used (this method does not require a core ontology to build the new ontology), and (2) the kind of texts to be analyzed, because the method shown in Section 3.5.1 is not necessarily focused on technical texts.

Activity 2. Linguistic study. This activity consists in selecting adequate linguistic tools and techniques (for example, pattern-based extraction and conceptual clustering), and applying them to the texts being analyzed to obtain domain terms, lexical relations, and groups of synonyms.

Activity 3. Normalization. The result of this activity is a conceptual model expressed by means of a semantic network. This conceptual model is rather informal and can be easily understood by the ontology builder. Normalization includes two steps: a linguistic step and a conceptual modeling one.
• During the **linguistic step**, the ontologist has to choose the relevant terms and the lexical relations (hyperonym, hyponym, etc.). According to the authors this choice is mainly subjective. Terms and lexical relations are kept when they seem important both for the domain and for the application where the ontology will be used. Also in this linguistic step, the ontologist adds a natural language definition to these terms according to the senses they have in the source texts. If there are terms with several meanings, the most relevant senses are taken.

• During the **conceptual step**, terms and lexical relations are transformed into concepts and semantic relations, which are defined in a normalized form using the terms associated to concepts and relations.

The result of the normalization activity has to be checked in accordance with differentiation rules that require that for any given concept the following information should be made explicit in the model:

• The concept must have, at least, one common attribute or relation with its father concept (generally an inherited attribute or relation).

• The concept must have, at least, one specific attribute or relation that makes it different from its father concept.

• The concept must have, at least, one property that makes it different from its sibling concepts.

**Activity 4. Formalization.** Concepts and relations of the semantic network are implemented in a formal language.

Several tools give support to different activities of this method. *Lexter* (Bourigault et al., 1996) is used in the linguistic study activity to obtain domain terms. *Géditerm* (Aussenac-Gilles, 1999) and *Terminae* (Biébow and Szulman, 1999) are used in the normalization activity to choose terms, and *Terminae* is also used in the formalization activity. This method was applied, as an experiment, to organize the concepts of the Knowledge Engineering domain in French (Aussenac-Gilles et al., 2000a).

### 3.6 Ontology Merging Methods and Methodologies

Ontologies aim to capture consensual knowledge of a given domain in a generic and formal way, to be reused and shared across applications and by groups of people. From this definition we could wrongly infer that there is only one ontology for modeling a domain. However, we can find on the literature several ontologies that model the same knowledge in different ways. For instance, as we saw in Section 2.2, there are several top-level ontologies that differ in the criteria followed to classify the most general concepts of the taxonomy, and in the e-commerce field there are several standards and joint initiatives for the classification of products and services (UNSPSC, e-cl@ss, RosettaNet, NAICS, SCTG, etc.). This heterogeneity of ontologies also happens in many domains (medicine, law, art, sciences, etc.).
Noy and Musen (1999) defined ontology alignment and merging as follows: (1) ontology alignment consists in establishing different kinds of mappings (or links) between two ontologies, hence preserving the original ontologies; and (2) ontology merging proposes to generate a unique ontology from the original ontologies. This section will be focused on ontology merging, though some ideas presented in this section can be used for aligning ontologies. In fact, before merging two ontologies there has to be a correspondence between terms of the ontologies to merge. This section presents the following methods: ONIONS (Steve et al., 1998; Gangemi et al., 1999), FCA-Merge (Stumme and Maedche, 2001) and PROMPT (Noy and Musen, 2000).

3.6.1 ONIONS

One of the most elaborated proposals for ontology merging is ONIONS, developed by the Conceptual Modeling Group of the CNR in Rome, Italy. With this method we can create a library of ontologies originated from different sources. In the first stages of the integration, the sources are unified without considering the generic theories that will be used in the modeling (part-whole or connectedness theories, for example). Later, these theories are incorporated as a base of the library and they are integrated as faces of the same polyhedron. In this way, the ontologies of the library are connected through the terms of the shared generic theories.

The specific activities proposed by ONIONS are shown in Figure 3.32:

**Activity 1. The creation of a corpus of validated sources.** The selection of sources (electronic dictionaries, informal is-a hierarchies, etc.) is made taking into account their diffusion and validation inside the domain community, for example, traveling.

Figure 3.32: Activities followed in the ONIONS method for ontology merging.
**Activity 2. Taxonomic analysis.** The taxonomy contained in each single source is inferred.

**Activity 3. Local definition analysis.** A unified definition in natural language is given for each concept of the corpus. When the unified definition is created, it is important to establish the differences between sibling concepts. For example, if we have the sibling concepts domestic flight and international flight, we have to distinguish them by introducing in the definitions expressions such as “the departure and the arrival places belong to the same country” or “the departure and the arrival places belong to different countries”. It is also important to detect and remove ambiguities and modal expressions (“it is usually characterized by...”). This activity is called “local” because the definitions are not yet connected through generic theories.

**Activity 4. Multi-local definition analysis: triggering paradigms.** In this activity, each term is formalized without including formal axioms. The definition can contain terms like Subclass-Of, Part-Of, Transmitted-By, etc., which have to be searched in theories containing them. This activity is called “multi-local” because the links between the ontology definitions and the generic theories have to be made explicit. For instance, in a definition that contains the term Part-Of, we have to specify that such a term should be defined in a mereology theory.

**Activity 5. The building of an integrated ontology library.** The definitions are enriched with formal axioms, and domain and generic ontologies are integrated in the same library. Sometimes, the generic ontologies may have to be extended and integrated in the library since they are not intended to be used in all the different domain ontologies.

**Activity 6. The classification of the library.** The ontology libraries are implemented in a language with automatic classification capabilities, such as LOOM.

In this process, we should make a clear distinction between the linguistic level and the conceptual level. So, polysemy, homonymy and synonymy appear:

- **Polysemy.** Different concepts have to be defined for any set of polysemic senses of a term. Polysemy occurs when two concepts with overlapping or disjoint meanings have the same name. For example, flight can be: an instance of traveling by plane; a scheduled trip by plane between designated airports; etc., as we explained in Section 1.5. Therefore, a different concept for each meaning of flight has to be created.

- **Homonymy.** It occurs when two disjoint concepts have the same name. For example, flight in the sense of instance of traveling by plane, and flight meaning a set of steps between one floor or landing. The homonymy has to be solved using a different name for each concept, or preventing the homonyms to be included in the same module namespace.

- **Synonymy.** It is the converse of homonymy, and occurs when two concepts with different names has the same meaning. Synonyms can be preserved linked to the concept.
When the library is built, the activities to incorporate new ontologies are (Gangemi et al., 1999):

Activity 7. Ontology representation. Each ontology is implemented in the language of the library. For example, the authors of this method use the LOOM language.

Activity 8. Text analysis and formalization. If texts on the domain of the ontologies are available, they are analyzed and formalized.

Activity 9. Integration of intermediate products. The intermediate products obtained in the previous activities are integrated in the library. To carry out this activity the difference between each of the sibling concepts is obtained, and this difference is formalized by axioms that reuse (if available) concepts and relations of the library. The new concepts are linked to the library through the Subclass-Of and Superclass-Of relations. Moreover, the aforementioned synonym, homonym and polysemy terms have to be dealt with. A term of a new source can have some of these relations with another term of a new source, or with a term of the library.

Activity 10. Ontology mapping. The source ontologies are explicitly mapped in the integrated ontologies to allow interoperability (maybe partial). The only explicit mappings are equivalent and coarser equivalent. The equivalent mapping is $C_1 = C_2$, whereas the coarser equivalent mapping is $C_1 = C_2 \cup C_3$, where $C_2$ and $C_3$ are disjoint.

The library ON.9.2 has been created with ONIONS and it unifies medical ontologies like GALEN or UMLS. These medical ontologies have been presented in Chapter 2. Nowadays, ONIONS is evolving from the versions presented in the mid-1990s. The last version is more tightly integrated with foundational ontologies and linguistic processing. We do not present it here because it has not been published yet. However, it is being applied to several industrial projects (banking regulations, service-level agreements, fishery terminologies, etc.).

3.6.2 FCA-Merge

FCA-Merge (Stumme and Maedche, 2001) was developed at the Institute AIFB of the University of Karlsruhe, Germany. This approach is very different from the other approaches presented in this section. FCA-Merge takes as input the two ontologies to be merged and a set of documents on the domains of the ontologies, as shown in Figure 3.33. The merging is performed by extracting, from the documents, instances that belong to concepts of both ontologies. Thus, if the concept $C_1$ of the ontology $O_1$ has instances in the same documents as the concept $C_2$ of the ontology $O_2$, then $C_1$ and $C_2$ are candidates to be considered the same concept. To establish this relation between concepts and documents, we have created a table for each ontology. Each table relates each concept $C$ of the associated ontology with the documents where instances of $C$ appear. A lattice structure is generated from the tables and, finally, the merged taxonomy is obtained from the structure. At present, this method works only for lightweight ontologies.
An algebra \((L;\wedge,\vee)\) is called a lattice\(^{14}\) if \(L\) is a nonempty set, \(\wedge\) and \(\vee\) are binary operations on \(L\), both \(\wedge\) and \(\vee\) are idempotent, commutative, and associative, and they satisfy the absorption law. The lattices used by Stumme and Maedche (2001) can be expressed:

- Each element of \(L\) is a pair of sets \(\{\{d_1,d_2,...,d_n\},\{c_1,c_2,...,c_m\}\}\), where each \(d_i\) is a document, and each \(c_i\) is a concept. \(\{c_1,c_2,...,c_m\}\) represents the concept \(c_1\wedge c_2\wedge...\wedge c_m\).
- \((A,B) \wedge (A',B') = (A\cap A',B'')\), where \(B''\) contains every concept of \(B\) and every concept of \(B'\).
- \((A,B)\vee(A',B') = (A\cup A',B'')\), where the concept represented by \(B''\) is the union of the concept represented by \(B\) and the concept represented by \(B'\).
- Two elements \((A,B)\) and \((A',B')\) of \(L\) are graphically linked if and only if \(A \subseteq A'\).

\(^{14}\) http://mathworld.wolfram.com/Lattice.html
Suppose that we want to merge the two ontologies of Figure 3.34 with FCA-merge. To do this with the FCA-Merge method the following activities must be carried out:

Activity 1. To extract instances. We get a table for each ontology to merge, by using NL analysis tools. Each table relates each document with the ontology concepts that have instances in the document. For example, Tables 3.14 and 3.15 show the concepts of the ontology $O_1$ and $O_2$ respectively, related to the documents where they have instances. Thus, the concept travel of $O_1$ has instances in the documents 1, 2, 3 and 4, and the concept travel of $O_2$ has instances in the same documents. For the sake of clarity, we will add a suffix to every term to distinguish the source ontologies where they are defined. Thus, for instance, the terms travel_1 and travel_2 are used to refer to the terms travel of $O_1$ and travel of $O_2$ respectively.

Table 3.14: Appearance of instances of the $O_1$ concepts in the documents considered.

<table>
<thead>
<tr>
<th></th>
<th>root</th>
<th>travel</th>
<th>travel agency</th>
<th>travel by train</th>
<th>travel by car</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>doc2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>doc3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>doc4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc5</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.15: Appearance of instances of the $O_2$ concepts in the documents considered.

<table>
<thead>
<tr>
<th></th>
<th>root</th>
<th>travel</th>
<th>traveler</th>
<th>travel by train</th>
<th>travel by plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc1</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>doc2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>doc3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>doc4</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>doc5</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Activity 2. To generate the pruned concept lattice structure. From the tables generated in the former activity, a lattice structure is automatically generated and pruned. To do this, the following tasks are carried out:

Task 2.1. Lattice structure generation. Stumme and Maedche (2001) define each lattice node as a combination of columns of the tables. A node is represented by a pair composed by a set of documents and the set of concepts from the two ontologies that have instances in such documents. It is represented as:

$\{(\text{document}_1, \text{document}_2, \ldots, \text{document}_n), (\text{concept}_1, \text{concept}_2, \ldots, \text{concept}_m)\}$

For example, in Figure 3.35 the node $\{(\text{doc2, doc3, doc4), (travel_agency_1, travel_by_plane_2)}\}$ represents those concepts that have instances in the documents 2, 3 and 4. A node $N_i$ is a
child of other node $N_2$ if and only if the set of documents of $N_1$ is included in the set of documents of $N_2$.

![Lattice structure generated by FCA-Merge for the ontologies of Figure 3.34 and the documents of Tables 3.14 and 3.15.](image)

The node \((\{\text{doc2, doc3, doc4}\}, \{\text{travel_agency_1, travel_by_plane_2}\})\) is a child of the node \((\{\text{doc1, doc2, doc3, doc4}\}, \{\text{travel_1, travel_2}\})\) because \(\{\text{doc2, doc3, doc4}\}\) is included in \(\{\text{doc1, doc2, doc3, doc4}\}\).

**Task 2.2. Lattice structure pruning.** Once the lattice structure has been generated, it is pruned. A node remains in the lattice structure if and only if it represents a concept of some of the two source ontologies, or has some child that represents a concept of some of the two source ontologies. Otherwise, it is pruned.

For example, as we can see in Figure 3.36, \((\{\text{doc2, doc3, doc4}\}, \{\text{travel_agency_1, travel_by_plane_2}\})\) remains in the pruned lattice, since it represents both the concept travel_agency_1 and travel_by_plane_2. However, the node \((\{\text{doc2, doc3}\}, \{\text{travel_agency_1, travel_by_plane_2, traveler_2, travel_by_car_1}\})\) should be removed from the lattice structure, since such a node represents the intersection travel_agency_1 \(\cap\) traveler_2 \(\cap\) travel_by_car_1 or the intersection
travel_by_plane_2 ∩ traveler_2 ∩ travel_by_car_1, but it does not represent a single concept of any ontology, and no concept of any ontology is represented by some child node.

The result of pruning the lattice of Figure 3.35 appears in Figure 3.36, where dashed lines mean that the edge and its origin node have been pruned, and where continuous lines are used for edges kept in the lattice.

Activity 3. To derive the merged ontology. Starting from the pruned lattice structure, the engineer obtains the target ontology. This activity can be software aided, although the role of the ontologist is fundamental. Some of the decisions to be made for each node are:

- If the node really represents a single concept. For example, the node 
  
  $$\{(doc2, doc3, doc4), \{travel_agency_1, travel_by_plane_2\}\}$$

  should be revised, since the ontologist may suspect that the terms travel_agency and travel_by_plane do not represent the same concept. This may happen because only a few documents have been used for merging the two ontologies. The Figure 3.37 shows that we have decided to generate two concepts from this node: travel_agency and travel_by_plane.

- If the node could represent a relation. Relations can be obtained from the intersection of concepts. For example, the same node 
  
  $$\{(doc2, doc3, doc4), \{travel_agency_1, travel_by_plane_2\}\}$$

  could derive the following ad hoc relation: a travel agency recommends a travel by plane. This relation is shown in Figure 3.37.
As in any process that uses texts to extract knowledge, a crucial decision in FCA-Merge is which documents should be selected.

```
Methodologies and Methods for Building Ontologies

As in any process that uses texts to extract knowledge, a crucial decision in FCA-Merge is which documents should be selected.

Figure 3.37: Resulting ontology from merging the ontologies of Figure 3.34.

3.6.3 PROMPT

The PROMPT method (Noy and Musen, 2000) has been elaborated by the Stanford Medical Informatics group at Stanford University. The main assumption of PROMPT is that the ontologies to be merged are formalized with a common knowledge model based on frames. A plug-in of Protégé-2000 merges ontologies according to the PROMPT method. In fact, this section is intertwined with Section 5.3.1, where we describe how to use the PROMPT plug-in.

Figure 3.38: Activities followed in the PROMPT method for ontology merging.
As we can observe in Figure 3.38, this method proposes first to elaborate a list with the operations to be performed to merge the two ontologies (e.g., merge two classes, merge two slots, etc.). This activity is carried out automatically by the PROMPT plug-in. Afterwards, a cyclic process starts. In each cycle the ontologist selects an operation of the list and executes it. Then a list of conflicts resulting from the execution of the operation must be generated, and the list of possible operations for the following iterations is updated. Some of the new operations are included in the list because they are useful to solve the conflicts.

Let us suppose that we want to merge the ontologies of Figure 3.39 using the PROMPT method.

![Figure 3.39: Sample ontologies to be merged through PROMPT.](image)

The set of activities to be performed are:

1. **Activity 1. To make the initial list of suggested operations.** The set of ontology-merging operations here identified includes both the operations normally performed during traditional ontology editing and the operations specific to merging and alignment and these are\(^{15}\):
   - Creating a new class in the new ontology by merging classes of the original ontologies.
   - Creating a new slot in the new ontology by merging slots of the original ontology.
   - Creating a new binding between a slot and a class in the new ontology by merging bindings between a slot and a class of the original ontologies.
   - Creating a new class in the new ontology by performing a deep copy of a class from one of the original ontologies. A deep copy consists in copying all the parents of a class up to the root of the hierarchy, and all the classes and slots it refers to.

\(^{15}\) This catalogue of possible operations might grow as PROMPT authors gain more experience.
Creating a new class in the new ontology by performing a shallow copy of a class from one of the original ontologies. A shallow copy consists in copying only the class and not its parents or the classes and slots it refers to.

The initial list of possible operations for the two ontologies of Figure 3.39 would be the one that appears in Table 3.16. We propose to merge those classes that have similar names in both ontologies (e.g., Travel of Ontology 1, and Travel of Ontology 2). We also propose to make a deep or shallow copy of the classes that are not common to both ontologies, depending on whether they have subclasses (e.g., Flight of Ontology 1) or not (e.g., Ship of Ontology 2) respectively.

Table 3.16: Initial list of possible operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Travel Ontology 1</th>
<th>Travel Ontology 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge</td>
<td>Travel</td>
<td>Travel</td>
</tr>
<tr>
<td>Merge</td>
<td>Train Travel</td>
<td>Train Travel</td>
</tr>
<tr>
<td>Deep copy</td>
<td>Flight</td>
<td>--</td>
</tr>
<tr>
<td>Shallow copy</td>
<td>--</td>
<td>Ship</td>
</tr>
<tr>
<td>Shallow copy</td>
<td>--</td>
<td>Travel in High-Speed Train</td>
</tr>
<tr>
<td>Shallow copy</td>
<td>American Airlines Flight</td>
<td>--</td>
</tr>
<tr>
<td>Shallow copy</td>
<td>British Airlines Flight</td>
<td>--</td>
</tr>
</tbody>
</table>

The following process is repeated until the two ontologies are merged:

Activity 2. To select next operation. The ontologist chooses an operation of the list. This decision might be based on different aspects: what (s)he wants to focus on, what (s)he considers important, etc. For example, the ontologist may decide that (s)he is only interested in merging parts of the ontology and just ignore other parts. Let us suppose that, in our travel domain, the ontologist chooses to merge the class Travel of both ontologies.

Activity 3. To perform the selected operation. The PROMPT plug-in executes the operation selected by the ontologist in the previous activity. After this point, the ontologist might order the PROMPT plug-in to carry out additional changes associated to the proposed operation. For instance, when the merging of the class Travel of both ontologies is carried out, all the original slots of Travel in Ontology 1, and of Ontology 2 can be included in the resulting class.
Activity 4. To find conflicts. This activity is performed by analyzing the resulting ontology. The following conflicts might appear in the resulting ontology after the execution of these operations:

- naming conflicts (more than one frame with the same name);
- dangling references (a frame refers to another frame that does not exist);
- redundancy in the class hierarchy (more than one path from a class to a parent other than root); and
- slot-value restrictions that violate class inheritance.

Following with our example, when we create the class Travel in the resulting ontology, the attributes arrival Place, departure Place, arrival Date and departure Date must be included though the classes defining their datatypes (Location and Date) are not.

Activity 5. To update the list of operations. For updating the list the PROMPT plug-in does not only consider the linguistic-similarity but also the structure of the ontology. This activity could be carried out with the algorithm AnchorPROMPT (Noy and Musen, 2001), though this is not the only option. AnchorPROMPT finds mappings between classes of two ontologies by means of the structure of such ontologies.

Besides, the PROMPT plug-in analyzes the relationships and definitions of the concepts that were involved in the latest operation executed.

Table 3.17 shows the updated list of possible operations after merging the class Travel of both ontologies. As we can see, the proposal of copying the data types Date and Location is found from the conflict list.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Travel Ontology 1 term</th>
<th>Travel Ontology 2 term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>Data type Date</td>
<td>Data type Date</td>
</tr>
<tr>
<td>Copy</td>
<td>Data type Location</td>
<td>Data type Location</td>
</tr>
<tr>
<td>Merge</td>
<td>Train Travel</td>
<td>Train Travel</td>
</tr>
<tr>
<td>Deep copy</td>
<td>Flight</td>
<td>--</td>
</tr>
<tr>
<td>Shallow copy</td>
<td>--</td>
<td>Ship</td>
</tr>
<tr>
<td>Shallow copy</td>
<td>--</td>
<td>Travel by High-Speed Train</td>
</tr>
<tr>
<td>Shallow copy</td>
<td>American Airlines Flight</td>
<td>--</td>
</tr>
<tr>
<td>Shallow copy</td>
<td>British Airlines Flight</td>
<td>--</td>
</tr>
</tbody>
</table>

After performing several iterations, the resulting ontology of our example might be the one appearing in Figure 3.40.

---

16 As with the merge operations, this catalogue of possible conflicts might grow as PROMPT authors gain more experience.
To perform the merge of our two travel ontologies, we have used the PROMPT plug-in, which gives support to this method.

![Ontology tree diagram](image)

**Figure 3.40:** Ontology resulting from merging the ontologies of Figure 3.39.

### 3.7 Co4: a Protocol for Cooperative Construction of Ontologies

One of the essential characteristics of ontologies is that they provide consensual knowledge in a given domain. However, most of the methods previously presented do not propose guidelines for reaching consensus when ontologies are built cooperatively. We will describe a protocol for reaching consensus between different knowledge bases.

Co4 (Rechenmann, 1993; Euzenat, 1995; 1996) is a protocol developed at INRIA for the collaborative construction of knowledge bases (KBs). Its goal is to allow discussing and committing to the knowledge in the KBs of a system.

The protocol takes as an input a set of KBs organized in a tree. The leaves of this tree are called user KBs, and the intermediate nodes, group KBs. The user KBs do not need to have consensual knowledge, while group KBs represent the consensual knowledge among its sons (called subscriber KBs). A KB can be subscribed to only one group KB. Besides, the knowledge in a KB can be transferred to another KB.

Let us think about two groups developing ontologies in Europe (group 1.1 and group 1.2) and two groups in America (group 2.1 and group 2.2) who are building
collaboratively our travel agency ontology. Figure 3.41 represents a possible KB tree. The European groups must reach consensus in KB.1, and the American groups must reach consensus in KB.2. Moreover, all of them must reach consensus in the KB.0.

To reach consensus for different KBs, every KB modification proposal must be submitted to the upper node in the tree, so that it can be analyzed by the rest of the sibling groups. The modification proposal is accepted only when all the sibling groups accept it.

Let us suppose that group 1.1 proposes a modification to be accepted by all the groups. Figure 3.42 shows the process followed until consensus is reached:

1. The KB 1.1 sends a message to the KB 1.
2. The KB 1 sends a call for comments to all its subscriber KBs (in our case, the KB 1.2).
3. The KB 1.2 replies.
4. The KB 1 sends the decision to all its subscribers.

If the modification proposal is accepted, the European groups have reached consensus, and the process follows on:

5. The KB 1 sends the proposal to the KB 0.
6. The KB 0 transmits a call for comments to the KB 2.
7. The KB 2 sends the call for comments to its subscriber KBs (KB 2.1 and KB 2.2).
8. The KB 2.1 and the KB 2.2 send their replies to the KB 2.
9. The KB 2 replies to the KB 0.
If the proposal is accepted, that is, if there is a general agreement, the KB 0 propagates the acceptance decision to all its subscriber KBs. This decision is sent recursively until the leaves of the KB tree are reached, as shown at the bottom of Figure 3.42.

![Diagram of agreement following the Co4 model](image)

In the process described above, we have supposed that all the reply messages from subscriber KBs to their corresponding group KB accept the modification proposal. However, any of them could have rejected it, what means lack of agreement.

One of the characteristics of this protocol is that the absence of agreement about a modification proposal does not prevent other users from accepting the modification. For example, if the KB 1.1 modification proposal is only accepted by the KB 1.2 and the KB 2.1, then the modification will be carried out in all the European KBs and in the American KB 2.1, but not in the KB 0 nor in the KB 2.2.

Co4 also considers the possibility of subscribing a new KB when the construction of the rest of KBs has already started. In this case, all the knowledge that has been already agreed by the KB group to which the KB subscribes is included in this KB.

Moreover, independent KBs can take part as “observers”, so they can see any modifications in the tree but cannot submit them.
Co4 originated from two experiences with knowledge bases in the domain of molecular genetics. The first, ColiGene (Perrière and Gautier, 1993), describes the regulation mechanism of gene expression in the *E. coli* bacterial genome. The second allows the description and manipulation of mammal genomic maps at the cytogenetic, genetic, and physical levels. The Co4 protocol is supported by the Co4 system (Alemany, 1998).

### 3.8 Methods for Evaluating Ontologies

As any other resource used in software applications, the content of ontologies should be evaluated before (re)using it in other ontologies or applications. In that sense, we could say that it is unwise to publish an ontology or to implement a software application that relies on ontologies written by others (even yourself) without evaluating first its content, that is, its concepts definitions, taxonomy and axioms. A well-evaluated ontology will not guarantee the absence of problems, but it will make its use safer.

The first works on ontology content evaluation started in 1994 (Gómez-Pérez, 1994a; 1994b). In the last two years the interest of the Ontological Engineering community in this issue has grown. The main efforts to evaluate ontology content were made by Gómez-Pérez (1996; 2001) in the framework of METHONTOLOGY, and by Guarino and colleagues (Welty and Guarino, 2001) with the OntoClean method.

This section summarizes both approaches and presents a few definitions of the most common terms used in this area. However, we will not evaluate specific ontologies implemented in a given language, nor how well-known ontology development tools perform content evaluation. We will present two approaches for evaluating taxonomies.

#### 3.8.1 Ontology evaluation terminology

A study of the knowledge based systems (KBS) evaluation ideas served as a precedent for evaluating ontologies and also for learning from KBS successes and mistakes (Gómez-Pérez, 1994b). The main ideas taken from this study were committed to prepare a framework for ontology evaluation, which includes: (1) the division of evaluation into two kinds: technical (carried out by developers) and users’ evaluation; (2) the provision of a set of terms for ontology evaluation process and the standard definitions of such terms; (3) the definition of a set of criteria to carry out the user’s and the technical evaluation processes; (4) the inclusion of evaluation activities in methodologies for building ontologies; (5) the construction of tools for evaluating existing ontologies; and (6) the inclusion of evaluation modules in tools used to build ontologies.

With regard to terminology and its definitions, the main terms (borrowed from the KBS evaluation field) are (Gómez-Pérez et al., 1995): “evaluation”, “verification”, “validation” and “assessment”. 
Ontology evaluation (Gómez-Pérez, 1996) is a technical judgment of the content of the ontology with respect to a frame of reference\(^{17}\) during every phase and between phases of their lifecycle. Ontology evaluation includes ontology verification and ontology validation. Ontology evaluation should be carried out on the following components at the ontology:

- Every individual definition and axiom.
- Collections of definitions and axioms stated explicitly in the ontology.
- Definitions imported from other ontologies.
- Definitions that can be inferred from other definitions and axioms.

Ontology Verification refers to building the ontology correctly, that is, ensuring that its definitions\(^{18}\) implement correctly the ontology requirements and competency questions, or function correctly in the real world.

Ontology Validation refers to whether the ontology definitions really model the real world for which the ontology was created. The goal is to prove that the world model (if it exists and is known) is compliant with the world modeled formally.

Finally, Ontology Assessment is focused on judging the ontology content from the user’s point of view. Different types of users and applications require different means of assessing an ontology.

In summary, the goal of the evaluation process is to determine what the ontology defines correctly, what it does not, and what it does incorrectly. To evaluate the content of a given ontology, the following criteria were identified (Gómez-Pérez, 1996): consistency, completeness and conciseness.

Consistency refers to whether it is possible to obtain contradictory conclusions from valid input definitions. A given definition is consistent if and only if the individual definition is consistent and no contradictory knowledge can be inferred from other definitions and axioms.

Completeness. Incompleteness is a fundamental problem in ontologies, even more when ontologies are available in such an open environment as the Semantic Web. In fact, we cannot prove the completeness of an ontology nor the completeness of its definitions, but we can prove the incompleteness of an individual definition, and therefore we can deduce the incompleteness of an ontology if at least one definition is missing in the established reference framework. So, an ontology is complete if and only if:

- All that is supposed to be in the ontology is explicitly stated in it, or can be inferred.

\(^{17}\) A frame of reference can be: requirements specifications, competency questions (Grüninger and Fox, 1995) the real-world, etc.

\(^{18}\) A definition is written in natural language (informal definition) and in a formal language (formal definition).
Each definition is complete. This is determined by figuring out: (a) what knowledge the definition specifies and if it defines explicitly the world; and (b) for all the knowledge that is required but not explicit, it should be checked whether it can be inferred from other definitions and axioms. If it can, the definition is complete; otherwise, it is incomplete.

**Conciseness.** An ontology is concise if: (a) it does not store any unnecessary or useless definitions, (b) explicit redundancies between definitions of terms do not exist, and (c) redundancies cannot be inferred from other definitions and axioms.

### 3.8.2 Taxonomy evaluation

In addition to providing the terminology for ontology evaluation described in Section 3.8.1, Gómez-Pérez has identified and classified different kinds of errors in taxonomies. Such an identification can be used as a check list for taxonomy evaluation. The following list presents a set of possible errors that can be made by ontologists when modeling taxonomic knowledge in an ontology under a frame-based approach. They are classified in: inconsistency, incompleteness, and redundancy errors, as shown in Figure 3.43. The ontologist should not postpone the evaluation until the taxonomy is finished; the control mechanisms should be performed during the construction of the taxonomy.

It is important to mention that we have slightly modified the terminology presented by Gómez-Pérez (2001) in order to be consistent with the vocabulary used in other sections of this book.

![Figure 3.43: Types of errors that might be made when developing taxonomies with frames.](image-url)
Inconsistency Errors can be classified in circularity errors, semantic inconsistency errors, and partition errors.

- **Circularity errors.** They occur when a class is defined as a specialization or generalization of itself. Depending on the number of relations involved, circularity errors can be classified as circularity errors at distance zero (a class with itself), circularity errors at distance 1, and circularity errors at distance \( n \). Figure 3.44 illustrates these errors. For example, if we say that Traveler is a subclass of Person, and that Person is a subclass of Traveler, then the ontology has a circularity error of distance 1.

- **Semantic inconsistency errors.** They usually occur because the ontologist makes an incorrect semantic classification, that is, (s)he classifies a concept as a subclass of a concept to which it does not really belong; for example, (s)he classifies the concept Airbus as a subclass of the concept Train. The same might happen when classifying instances.

- **Partition errors.** Concept classifications can be defined in a disjoint (disjoint decompositions), a complete (exhaustive decompositions), and a disjoint and complete manner (partitions). The following types of partition errors are identified:
  - **Common classes in disjoint decompositions and partitions.** These occur when there is a disjoint decomposition or a partition \( \text{class}_p_1, \ldots, \text{class}_p_n \) defined in a class \( \text{class}_A \), and one or more classes \( \text{class}_B_1, \ldots, \text{class}_B_k \) are subclasses of more than one \( \text{class}_p_i \).

  Figure 3.45a shows an error of this type in a disjoint decomposition. The classes Air Transport, Sea Transport and Ground Transport form a disjoint decomposition of the class Transport. This error appears if we define the class Seaplane as a subclass of the classes Air Transport and Sea Transport.
Figure 3.45b shows an error of this type in a partition. The classes *International Flight* and *Domestic Flight* form a partition of the class *Flight*. This error occurs if the class *Non Stop Flight* is defined as a subclass of the classes *International Flight* and *Domestic Flight* simultaneously.

![Disjoint Decomposition and Partition Diagram]

**Figure 3.45:** Common classes in disjoint decompositions and partitions.

- **Common instances in disjoint decompositions and partitions.** These errors happen when one or several instances belong to more than one class of a disjoint decomposition or partition.

  Figure 3.46a shows an error of this type in a disjoint decomposition. The classes *Air Transport*, *Sea Transport* and *Ground Transport* form a disjoint decomposition of the class *Transport*. This error happens if we define *Seaplane HA-14* as an instance of the classes *Air Transport* and *Sea Transport*.

  Figure 3.46b shows an error of this type in a partition. The classes *International Flight* and *Domestic Flight* form a partition of the class *Flight*. This error occurs if a specific flight is defined as an instance of both classes.

![Disjoint Decomposition and Partition Diagram]

**Figure 3.46:** Common instances in disjoint decompositions and partitions.
• **External instances in exhaustive decompositions and partitions.** These errors occur when we have defined an exhaustive decomposition or a partition of the base class (Class_A) into the set of classes class_p1,...,class_p_n, and there are one or more instances of the class_A that do not belong to any class class_p_i of the exhaustive decomposition or partition.

Figure 3.47a shows an error of this type in an exhaustive decomposition. The classes Economy Trip, Business Trip, and Luxury Trip form an exhaustive decomposition of the class Travel Package. This error happens if we define an instance John’s Package as an instance of the class Travel Package.

Figure 3.47b shows an error of this type in a partition. The classes International Flight and Domestic Flight form a partition of the class Flight. This error occurs if we define the flight AA7462 that departs on February 8, 2002 as an instance of the class Flight instead of as an instance of the class International Flight.

Detecting incompleteness on taxonomies. Errors appear when the superclasses of a given class are imprecise or over-specified, and when information about subclasses that belong to disjoint decompositions, exhaustive decompositions, and partitions is missing. Very often, when a concept taxonomy is built, the ontologist only uses the Subclass-Of relation to build the taxonomy and omits disjoint knowledge between classes. We present here common omissions when building taxonomies.

• **Incomplete Concept Classification.** Generally, an error of this type is made whenever concepts are classified without accounting for them all, that is, concepts existing in the domain are overlooked. On the traveling domain, an error of this type occurs if we classify locations only taking into account beach locations and mountain locations, and we do not consider cultural locations, skiing locations, etc.

• **Partition Errors.** They could happen when the definition of disjoint and exhaustive knowledge between classes is omitted. We have identified two types of errors:
  - **Disjoint knowledge omission.** The ontologist identifies the set of subclasses of a given class, but omits to model, in the taxonomy, that the subclasses...
are disjoint. An example would be to define the classes South American Location and North American Location as subclasses of American Location, without specifying that they form a disjoint decomposition of the class American Location.

- **Exhaustive knowledge omission.** The ontologist defines a decomposition of a class, omitting the completeness constraint between the subclasses and the base class. An example would be to define South American Location and North American Location as a disjoint decomposition of an American Location without specifying that they form an exhaustive decomposition.

**Redundancy detection.** It occurs in taxonomies when there is more than one explicit definition of any of the hierarchical relations, or when we have two classes or instances with the same formal definition.

- **Redundancies of Subclass-Of relations** occur between classes that have more than one Subclass-Of relation. We can distinguish direct and indirect repetition. A Direct repetition exists when two or more Subclass-Of relations between the same source and target classes are defined. An Indirect repetition exists when two or more Subclass-Of relations between a class and its indirect superclasses are defined. For example, Figure 3.48 shows an indirect repetition, because the class AA2010 is defined as a subclass of AmericanAirlinesFlight and Flight, and AmericanAirlinesFlight is defined as a subclass of Flight.

![Figure 3.48: Indirect repetition.](image)

- **Redundancies of Instance-Of relations.** As in the above case, we can distinguish between direct and indirect repetition.

- **Identical formal definition of classes.** Different classes have the same formal definition, although they have different names.

- **Identical formal definition of instances.** Different instances have the same definition, although they have different names.
3.8.3 OntoClean

OntoClean is a method elaborated by the Ontology Group of the CNR in Padova (Italy). Its goal is to remove wrong Subclass-Of relations in taxonomies according to some philosophical notions such as rigidity, identity and unity. These notions are applicable to properties, but can be extended to concepts, as we did in Chapter 2. The notions of rigidity and identity were explained in Section 2.2.1. Let us remember them:

- **Rigidity.** A property is rigid (+R) if and only if it is necessarily essential to all its instances. For example, the concept person is rigid, since every person is necessarily a person in every possible situation. On the other hand, a property is anti-rigid (~R) if and only if it is not essential for all its instances (for example, student). Finally, a property is non-rigid (-R) if and only if it is not essential for some of its instances. For example, the concept red is non-rigid, since there are instances of red which are essentially red (Drop of Blood), and instances that are not essentially red (My Pullover).

- **Identity.** A property (for example being a person) carries an identity criterion (IC), which is denoted by the symbol (+I), if and only if all its instances can be (re)identified by means of a suitable “sameness” relation (for example, DNA). The IC must be applicable to every instance in every moment. A property supplies an identity criterion (+O) if and only if such a criterion is not inherited by any subsuming property belonging to the ontology.

It might occur that a given property could inherit an identity criterion from any of its parents, and the property might have its own identity criterion at the same time. In such a case, the property carries two identity criteria and supplies its own identity criterion, but not the inherited criterion from its parents.

Sometimes it is hard to find and make explicit a suitable IC even though we know that the IC exists. Suppose that a non-expert in art makes a taxonomy about pictures, and (s)he wants to identify an identity criterion for Picasso’s painting. In that case, it is quite hard for the non-expert to identify which is the identity criterion of the property Picasso’s painting. However, we know that experts in art could identify easily Picasso’s paintings. So, the identity criterion exists, and we could tag the property Picasso’s painting with (+I).

We will now provide simplified and approximated definitions related to the notion of unity, considering that for understanding the idea of unity, it is necessary to know the idea of whole previously. To have more precise definitions, you can consult (Gangemi et al., 2001).

- **Whole.** We can say that an individual $w$ is a whole if and only if it is made up of a set of parts unified by a relation $R$ that links every pair of parts of $w$, and no part of $w$ is linked by $R$ to something that is not part of $w$, as we can see in Figure 3.49.

Let us suppose that we want to represent the concept Piece of Metal and that a piece of metal is composed of different sections. As we can see in Figure 3.50, every pair of sections of a piece of metal is intimately
connected with the other pieces, which means that the metal of the different sections is linked at the molecular level. However, we can say that two or more pieces of metal in mere contact are not intimately connected in the sense we have mentioned before. Since each section could be considered as a part, every piece of metal is a (topological) whole because every part is related through the relation intimately connected to any of the others, and there is no part of the piece of metal intimately connected to something that is outside such a piece.

Figure 3.49: The notion of whole.

Figure 3.50: An example of the notion of whole: relations between parts of a piece of metal.
Unity. A property $P$ is said to carry unity (+U) if there is a common unifying relation $R$ such that all the instances of $P$ are wholes\(^9\) under $R$. According to the previous example, we can say that the concept Piece of Metal carries unity (+U) because every piece of metal is a whole under the relation intimately connected. This relation is named unity criterion (UC). As it happens with the identity criterion, the unity criterion can be inherited from any of its parents.

If a property does not carry unity, it can be tagged with (-U). There are two cases of properties that do not carry unity (-U):

Properties carrying anti-unity. A property carries anti-unity (~U) if and only if all its instances can possibly be non-wholes. That is, there is not any instance that can be unified by a relation $R$ in any situation (permanently). To explain anti-unity, we will think of the concept Metal. We may find a relation $R$ that unifies a particular instance of metal in a particular moment, for example, we can find the relation metal belonging to the same car; however, this relation is not permanent, since a car can be dismantled, and the metal of such a car does not belong to the same car any more. Therefore, metal carries anti-unity (~U).

Properties that do not apply a common unity criterion to all its instances. This case differs from the other in that it is possible that some, or even every instance $i$ has a permanent relation $R_i$ that unifies it. However, this relation is not the same for all the instances. Thus, some (or even all) instances can be wholes, but with different unity criteria. An example is Legal Agent (Welty and Guarino, 2001), which could include people, companies, etc.

Bearing in mind these meta-properties, wrong Subclass-Of relations in a taxonomy can be cleaned. According to CNRs proposal, the OntoClean method proposes five activities for cleaning taxonomies (Welty and Guarino, 2001). Let us take the taxonomy of Figure 3.51 as an example.

![Diagram of taxonomy](image)

Figure 3.51: Example of taxonomy to be evaluated.

---

\(^9\) In the actual definition, the OntoClean authors use essential wholes instead of wholes.
Activity 1. Assigning meta-properties to every property. This activity proposes to assign values to the meta-properties for each concept in the taxonomy. Thus, the ontologist tags all the concepts with the following symbols: is rigid (+R); is anti-rigid (~R); is non-rigid (-R); carries identity criterion (+I); supplies identity criterion (+O); carries unity (+U); and carries anti-unity (~U). With regard to (~U), although it could be used for any concept that does not carry unity, it will be reserved for those concepts that carry neither unity nor anti-unity, as explained previously with the concept Legal Agent.

The top-level ontologies of universals and particulars presented in Section 2.2.1 can be used to set up these meta-properties. However, how to carry out this is out of the scope of this book. Figure 3.52 shows the result of assigning meta-properties to the taxonomy of Figure 3.51. For each concept in the taxonomy, we have followed this order: rigidity, identity and unity. This assignment has been made without taking into account the top-level ontologies.

**Figure 3.52: Result of assigning meta-properties to the taxonomy.**

Rigidity. We can say that the concepts Thing, Metal, Metallic Building, Building, Hotel and Airport are rigid (+R) because any instance of any of these concepts has to be an instance of this concept in any situation. For example, Building is rigid since every building is necessarily a building in every possible situation. The concepts Departure and Destination are anti-rigid (~R), since a departure (or a destination) can stop being a departure, for instance, due to political or economical reasons.
Identity. Let us start with the concept on top of the taxonomy. There is not a unique identity criterion valid for everything. We can say that Thing neither carries (-I) nor supplies identity criterion (-O). Let us think now of the case of Building. Which features allow us to identify a building? We know that we can identify a building by its shape, materials, the little details of its front, etc. Therefore, Building carries an identity criterion (+I), though it may be very complex to define it as we said before when explaining the identity notion. As the parent of Building (Thing) does not carry identity criterion, the identity criterion of building is not inherited, therefore, we can say that Building supplies identity criterion (+O).

Let us suppose that when we built our taxonomy we wanted to model hotel from a legal point of view. Now, when we are assigning meta-properties, the identity criterion that better reflects what we had in mind is the hotel legal code. Hence, the hotel legal code is one of its identity criterion, and Hotel carries identity criterion (+I). Since the hotel legal code criterion is not inherited from Building, we can say that Hotel supplies identity criterion (+O).

As in the case of Building, we know that Metal can be identified by an identity criterion and that it carries identity (+I). As the parent of Metal (Thing) does not carry identity criterion, the identity criterion of metal is not inherited, therefore, we can say that Metal supplies identity criterion (+O).

Metallic Building inherits its identity criterion from Building and from Metal, hence, it just carries two identity criteria (+I), but it does not supply them (-O).

Let us think now of Departure and Destination, and let us suppose that airports, bus stops, train stations, etc., played the role of departures and destinations when we built the taxonomy. In this case, there is not a common identity criterion, neither for Departure nor for Destination because if the identity criterion existed, it could be applied to all of them (airports, bus stops, etc.). Therefore, Departure and Destination do not carry identity criterion (-I). And since they do not carry identity criterion (they do not own nor inherit it from thing), they do not supply it (-O).

With regard to Airport, the airport legal code can be considered as one of its identity criteria. So, Airport carries identity criterion (+I). As the identity criterion is not inherited from Departure and Destination, we can say that Airport supplies identity criterion (+O).

To sum up, Metal, Building, Hotel and Airport carry (+I) and supply (+O) identity criterion, Metallic Building carries identity criterion (+I) but it does not supply (-O) it, and Thing, Departure and Destination neither carry (-I) nor supply (-O) identity criterion.
Unity. Sometimes it is not easy to find the unity relation. However, it is possible to know that a property has a unity criterion, although we do not know exactly which.

Concerning Thing, there are things unified by a functional unity criterion, other things unified by a topological unity criterion, etc. Therefore, there is not a unique unity criterion valid for everything. As a consequence, Thing neither carries unity nor anti-unity, for which it is tagged with (-U).

With regard to the case of Building, a building is constructed according to its function (to lodge people, to store goods, etc.). Different parts of the building are designed according to such a function. So, the unity criterion depends on the function of the building. On the other hand, the different parts of the building are topologically connected. Consequently, a building is a whole under a unity criterion that is a combination of a functional and topological unity. Therefore we can say that Building carries a unity criterion (+U) that is not only functional but also topological.

Let us consider now the case of Hotel. For a hotel to carry out its function, workers, utilities, services, etc. must work properly. Therefore, the different parts of the hotel are unified to carry out the function of lodging people. As a consequence, we know that Hotel has to carry a functional unity criterion (+U).

As we said when dealing with the unity notion, Metal carries anti-unity (~U). Metallic Building does not inherit the unity criterion from metal but from Building. So Metallic Building carries unity (+U).

In the case of Departure and Destination, let us suppose that when we built the ontology we only had in mind departures and destinations as places from where passengers depart from and arrive at. So, departures and destinations are unified to carry out a function. Therefore, Departure and Destination carry a functional unity criterion (+U).

As for the case of Airport, applying a similar argument as the one used for Hotel, we can say that it carries a functional unity criterion (+U).

In summary, Hotel, Building, Metallic Building, Airport, Destination, and Departure carry unity (+U); Metal carries anti-unity (~U), and Thing carries neither unity nor anti-unity (-U).

Activity 2. Focusing only on the rigid properties. The goal of this activity is to identify which are the essential parts of the taxonomy. A taxonomy with only rigid properties is called backbone taxonomy, which is the base of the rest of the taxonomy. In our example, in this activity, we have focused on Thing, Metal, Metallic Building, Building, Hotel and Airport, and we have disregarded Departure and Destination at this moment, as Figure 3.53 illustrates.
**Activity 3. Evaluating the taxonomy according to principles based on the meta-properties.** To perform this evaluation, we will present three rules (Welty and Guarino, 2001) that are related with the identification of conflicts between identity criteria, unity criteria and between anti-unity and unity.

**Rule 1: Checking identity**

The rule for detecting conflicts between identity criteria, inspired in Lowe (1989), states that properties carrying (+I) incompatible identity criteria are necessarily disjoint. Following this rule, two properties carrying incompatible identity criteria cannot have common individuals. If two concepts with incompatible identity criteria had common individuals, the individuals would inherit two identity criteria which are incompatible. So the candidate concepts that could breach the rule are:

- Concepts that are not disjoint.
- Concepts that carry identity (+I).

Besides, if there is incompatibility between the identity criteria carried by both concepts, we can state that the two concepts do not satisfy the rule.

![Backbone taxonomy](image)

Figure 3.53: Backbone taxonomy.

We can see in Figure 3.53 that **Hotel** is a subclass of **Building**. So, they are not disjoint. Besides, both **Hotel** and **Building** carry identity (+I); therefore, they are “suspects” of breaching the rule. Now, we have to search for possible incompatible ICs between them. According to what we said when we tagged the taxonomy, **Hotel** has its own identity criterion and inherits the IC of **Building** because **Building** supplies identity criterion (+O). This means that every hotel could be identified by the features of its building. However, given that **Hotel** is considered here from the legal point of view, a hotel can legally exist even when it does not have a building. Consequently, the features of its building cannot be used to identify the hotel. Hence, there exists a conflict between the identity criteria when we say that **Hotel** is a subclass of **Building**. So, the inherited identity criterion from building is incompatible with its own identity criterion; therefore, we remove the **Subclass-Of** relation.
between Hotel and Building. The solution proposed in this example is to link directly Hotel with Thing using the Subclass-Of relation, but this solution does not represent that a hotel has a building. Thus, we propose to create the relation has Building between Hotel and Building. By doing this, we model that a hotel has a building, but it is not a building. With this solution we can represent hotels that do not have a building.

Rule 2: Checking unity

The rule for detecting conflicts between unity criteria states that properties carrying incompatible unity criteria (UCs) are necessarily disjoint. According to this rule, two properties carrying incompatible unity criteria cannot have common individuals. If two concepts with incompatible unity criteria had common individuals, the individuals would inherit two unity criteria which are incompatible. So the candidate concepts that could breach the rule are:

- Concepts that are not disjoint.
- Concepts that carry unity (+U).

If there is incompatibility between the unity criteria carried by both concepts; then we can state that the two concepts do not satisfy the rule.

If we analyze Figure 3.54, we can say that rule 2 has not been breached. To show how to work with this rule, we will take the taxonomy as it appeared in Figure 3.53. The unity notion can be used to reinforce that hotel cannot be a subclass of Building. Looking at Figure 3.53, we can notice that hotel and building are not disjoint and both carry unity (+U). So, they are “suspects” of breaching the rule. Now, we have to search for possible incompatible UCs between them. As we have said before, Building carries a functional and topological unity criterion. However, Hotel carries only a functional unity criterion and not a topological one. That is, a topological unity criterion is not applicable to Hotel. Therefore, there is a breach of rule 2.

Figure 3.54: Result of restructuring the backbone taxonomy after applying the identity rule.
This violation is problematic because if the hotel has several buildings, we should have several instances for the same hotel. The solution shown in Figure 3.54 also solves this incompatibility.

**Rule 3: Checking unity and anti-unity**

The rule for detecting conflicts between properties with anti-unity (~U) and properties carrying unity (+U) states that a property carrying anti-unity (~U) has to be disjoint with a property carrying unity (+U). If a property $P_1$ carries anti-unity (~U) and a property $P_2$ carries unity (+U), why cannot $P_1$ have common individuals with $P_2$? That is, why must they be disjoint? If we know that $P_1$ carries anti-unity (~U), no common individual would be a whole under a common unifying relation. If we know that $P_2$ carries unity (+U), all common individuals would be a whole under a common unifying relation (which is the unity criterion). In other words, any common individual would be a whole and would not be a whole, which is a contradiction. So the concepts that could violate the rule are:

- Concepts that are not disjoint, and
- One of the concepts that carries anti-unity (~U) and another that carries unity (+U).

For instance, we can see in Figure 3.54 that Metallic Building and Metal are not disjoint, since Metallic Building is a subclass of Metal. Besides, Metallic Building is tagged with +U, whereas Metal is tagged with ~U. Therefore, there is a violation of the rule. A solution against the violation of rule 3 is to say that a metallic building is constituted by metal, as we can see in Figure 3.55.

![Figure 3.55: Result of restructuring the backbone taxonomy after applying the unity and anti-unity rule.](image)

**Activity 4. Considering non-rigid properties.** When the backbone taxonomy has been examined, the ontology developer has to evaluate non-rigid (~R) and anti-rigid (~R) properties. A useful rule to perform such an evaluation is the following:
Rule 4. Checking rigidity (+R) and anti-rigidity (~R)
The rule for detecting conflicts between rigid and anti-rigid properties states that an anti-rigid property cannot subsume a rigid property, which means that a rigid concept cannot be a subclass of an anti-rigid one. So the concepts that could violate the rule are:
- The parent concept is tagged with ~R.
- The child concept is tagged with +R.

We can see in Figure 3.55 that Airport is a subclass of Departure and Destination. Besides, Airport is rigid (+R) and Departure and Destination are anti-rigid (~R). Therefore, there are two violations of rule 4: Airport can be neither a subclass of Departure nor of Destination.

This violation causes problems because any instance of airport is a departure and a destination according to the taxonomy presented in Figure 3.56. However, it might happen that for any reason the airport stopped being a departure or a destination for our travel agency. To solve this problem, we have two options. The first consists in considering Departure and Destination as subclasses of Airport but not the opposite. And the second consists in introducing the concept Travel and converting the concepts Departure and Destination in relations, as Figure 3.56 illustrates.

This last option forces us to extend the scope of the ontology and to carry out more knowledge acquisition, modeling, evaluation, etc.

![Diagram](image)

Figure 3.56: Result of considering non-rigid properties and looking for missing concepts.

Activity 5. Completing the taxonomy with other concepts and relations. We have several reasons to introduce new concepts. One is the transformation of concepts into relations, for example, when we have added the concept Travel. Given that we have introduced new concepts, we must repeat the process starting from the activity 1.

As we can see, OntoClean evaluation is not only useful to clean the concept taxonomy, but also to make explicit the ontological commitments assumed in the definitions of the ontology terms. Thus, for example, the assignment of meta-
properties to Departure and Destination has forced us to reflect on the exact sense of such terms in our ontology.

OntoClean is being used in the Eureka European project, where WordNet is being restructured; it is also used in the construction of a domain ontology in the financial field; and in the TICCA Italian project, where an ontology of social interaction is being created; etc.

At the time of writing this book WebODE is the unique ontology platform that includes a module for evaluating ontologies according to OntoClean method (Fernández-López and Gómez-Pérez, 2002b).

### 3.9 Conclusions

Throughout this chapter, we have presented in detail the ontology development process and the methods and methodologies that support the building of ontologies from scratch. Also, we have discussed particular methods that allow carrying out specific activities already identified in the ontology development process. Special attention must be given to the re-engineering method; to the ontology learning methods that help reduce the effort during the knowledge acquisition process; to the merging of ontologies that generates a unique ontology from several ontologies; to the ontology alignment that establishes different types of mapping between ontologies (hence preserving the original ones); and to the ontology evaluation methods used for evaluating the ontology content. For each methodology and method, we have given an example on the traveling domain.

The main conclusion (taken from the analysis of Tables 3.10 to 3.14) is that none of the approaches presented in Section 3.3 cover all the processes identified on the ontology development process. Most of the methods and methodologies for building ontologies are focused on the development activities, specially on the ontology conceptualisation and ontology implementation, and they do not pay too much attention to other important aspects related to management, learning, merge, integration, evolution and evaluation of ontologies. In fact, such types of methods should be added to the methodologies for ontology construction of Section 3.3. An attempt to include some methods into a methodology occurs in METHONTOLOGY, where re-engineering methods and the two evaluation methods presented in Section 3.8 are being integrated.

In this chapter, we have seen that although all the methods and methodologies presented in Section 3.3 propose to reuse existing ontologies, the SENSUS method automates the construction of domain ontologies from the reuse of the knowledge contained on huge ontologies (the SENSUS ontology). Specific methods for merging and alignment ontologies presented in Section 3.6 also favor the reuse of existing knowledge resources.

Now, we conclude with the specific methods presented in this chapter:

- **Ontology re-engineering methods (Section 4).** The method presented is just a sound initial approach to carry out the aforementioned process although it must
be improved in further studies with more complex ontologies. In order to increase the reusability of the ontology to be re-engineered, we need new guidelines and criteria to achieve a higher degree of reusability in the restructuring process.

- **Ontology learning methods (Section 5).** Ontology learning is a novel research area, and most of its approaches are based on extracting knowledge from text using a corpus that guides the natural language analysis. We propose to integrate ontology learning methods into the knowledge acquisition activity of ontology building methodologies.

- **Ontology merging methods and methodologies (Section 6).** The strategies presented for merging ontologies are quite diverse and are able to merge concept taxonomies, attributes and relations. However, they should include new methods for merging other ontology components, such as axioms.

- **Cooperative construction of ontologies (Section 7).** Co4 is a protocol for collaborative construction of consensual knowledge bases. Specific methods are needed for reaching consensus on ontology building.

- **Ontology evaluation methods (Section 8).** The field of ontology evaluation is just emerging. From the methodological perspective, evaluation activities should be introduced in more detail into some activities (e.g., conceptualisation and implementation) of the ontology development process.

Finally, we would like to point out that Ontologies are dynamic entities that evolve over time. The management of ontology evolution and the relationships between different versions of the same ontology are crucial problems to be solved. In this section we have not summarized the work carried out in this area, but we have identified the need to create and integrate robust methods inside the current methodologies to distinguish and recognize versions, with procedures for updates and changes in ontologies.

### 3.10 Bibliographical Notes and Further Reading

To those readers who want to read more about the contents of this chapter, we recommend the deliverable D.1.4 “A survey on methodologies for developing, maintaining, integrating, evaluating and re-engineering ontologies” of the OntoWeb thematic network. This deliverable contains detailed descriptions and references of these and other methods and methodologies for ontology construction. The descriptions can also be consulted and updated in the OntoRoadMap application ([http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html](http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html)).

In addition, the following readings, grouped by topics, are recommended:

- **Ontology development methods and methodologies.** Extended information on the methods and methodologies presented in Section 3.3 can be found for: the Cyc method ([http://www.cyc.com/publications.html](http://www.cyc.com/publications.html)), the Uschold and King’s
Methodologies and Methods for Building Ontologies

- Methodology for re-engineering ontologies. Papers on this issue are also available in the METHONTOLOGY’s URL.
- Ontology learning methods. We recommend reading the book “Ontology Learning for the Semantic Web”, from Alexander Maedche (2002), which contains descriptions of ontology learning algorithms and implementations such as the Text-To-Onto system. We also recommend the deliverable D1.5 “A survey on ontology learning methods and techniques” of the OntoWeb thematic network.
- Ontology merging methods and methodologies. ONIONS publications can be found at http://saussure.irmkant.rm.cnr.it/onto/publ.html. Maedche and colleagues’ ontology merging method publications can be found in http://wwwneu.fzi.de/wim/eng/publications.php. Concerning the PROMPT method, all the related publications can be found in Natasha Noy publications’ Web page (http://smi.stanford.edu/people/noy/publications.html).
- Co4: a protocol for cooperative construction of ontologies. The papers of Co4 can be found in http://www.inrialpes.fr/sherpa/publi.html.
- Ontology evaluation methods. Publications related to ontology evaluation can be found at http://delicias.dia.fi.upm.es/papers/evaluation.html. Papers about OntoClean and its theoretical foundations are available at http://ontology.ip.rm.cnr.it/Publications.html. Finally, there are also interesting papers related to ontology evaluation in Yannis Kalfoglou’s home page http://www.ecs.soton.ac.uk/~yk1/publications.html.