2. Using Ontologies in Software Engineering and Technology

Francisco Ruiz

ALARCOS Research Group. Dept. of Information Technologies and Systems, Escuela Superior de Informática, University of Castilla-La Mancha, Spain,
francisco.ruizg@uclm.es

José R. Hilera

Computer Science Department, University of Alcalá, Spain, jose.hilera@uah.es

2.1 Introduction

In this chapter, the state of the art on the use of ontologies in software engineering and technology (SET) is presented. The chapter is organized into four parts. In the second and third sections, serving as a supplement to Chap. 1,[29] a wide review of the distinct kinds of ontologies and their proposed uses is presented respectively. In the fourth section, we offer a taxonomy for classifying ontologies in SET, in which two main categories are distinguished: (1) SET domain ontologies, created to represent and communicate agreed knowledge within some subdomain of SET, and (2) ontologies as software artifacts, with proposals in which ontologies play the role of an additional type of artifact in software processes. On the one hand, the former category is subdivided into those ontologies included in software engineering and those referring to other software technologies.

[29] Readers can find a more detailed study on the ontology notion in the books "Ontological Engineering" by Gómez-Pérez et al. [38] and "Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce" by Fensel [30].

On the other hand, the latter category is subdivided into development time and run time proposals according to the moment when ontologies are used. Then, in the last section, we analyze and classify (based on our taxonomy) a large number of recently published works. We also comment on and classify works which will be presented in later chapters of this book.

2.2 Kinds of Ontologies

Although the term "ontology" was introduced in the eighteenth century to refer to the general science of being ("onto" in ancient Greek), Ontology as a discipline has been practiced by philosophers since the dawn of history (previously a part of metaphysics). Etymologists may define ontology as the knowledge of beings, that is, all that relates to being. Just as we call those who study "students", we use the term "entity" to describe all things which "are". From this point of view, stones, animals or people are "entities". Mathematical objects, even those that are merely imagined, are also considered beings (be they fictitious or unreal).

All sciences and knowledges refer to or examine a type of entity: some are physical, as in the physical sciences, others abstract or mental, as in mathematics and the vast majority of the computational sciences, and still others living, as in biology.

In the scope of the computational sciences and technologies (computer science, software engineering, information systems, etc.), ontology has boomed as a field of research and application since the latter part of the twentieth century. Perhaps the principal cause of this boom has been the key role that it plays in the new generation of the advanced Web (Semantic Web).

Focusing exclusively on the scope of this publication, that is, SET, the first known proposals were presented by Gruber [40, 41], whereby ontologies are "an explicit specification of a conceptualization". Conceptualization is understood to be an abstract and simplified version of the world to be represented: a representation of knowledge based on objects, concepts and entities existing within the studied area, as well as the relationships existing among them. By "explicit" we mean that the concepts used and the restrictions applied to them are clearly defined. Later authors have considered it important to add to this definition two new requirements: that the said specification be (1) formalized and (2) shared. By "formalized" it is meant that a machine can process it. By "shared" it is understood that the knowledge acquired is the consensus of a community of experts [38]. In regards to this last requirement, common ontologies are used to describe ontological commit-

ments for a set of agents (people or artificial systems) so that they can communicate and interact with a domain of discourse. Additionally, an agent commits to an ontology if its observable actions are consistent with the definitions of the ontology. This idea of ontological commitments was proposed by Newell [68] from a knowledge-level point of view.

Some SET researchers view ontologies as "a vocabulary" for a specific domain representing conceptual elements and the relationships existing between them. However, the ontology is not the vocabulary itself, but what the vocabulary represents, since the translation of this vocabulary into another language will not change the ontology [16].

Other researchers defend the need for ontologies to be viewed as a "theory", that is, a formal vocabulary with a set of defining axioms. These axioms express new relationships between concepts and limit the possible interpretations [75, 98]. However, many experts have concluded that ontologies of the software systems application domain, or of its design and construction processes, are of great assistance in avoiding problems and errors at all stages of the software product life cycle: from the initial requirements analysis (facilitating the analyst–client interaction), through the development and construction phase, and finaly with the maintenance stage (assuring greater understanding of the modification requests, better understanding of the maintained system, etc.).

Additionally, numerous authors have viewed ontologies from distinct vantage points. Therefore it is not surprising that in the literature we find diverse classifications of ontologies with different focuses.

According to the generality level, Guarino considers that the following ontology types exist [43]:

- *High–level ontologies*: Describe general concepts such as space, time, material, object. They are independent of a specific domain or problem. Their purpose is to unify criteria between large communities of users.
- *Domain ontologies*: Describe the vocabulary related to a generic domain (for example, information systems or medicine), by means of the specialization of the introduced concepts of high–level ontologies.
- *Task ontologies*: Describe the vocabulary related to a generic task or activity (for example, development or sales), by means of specialization of the introduced concepts of high–level ontologies.
- *Application ontologies*: Describe concepts belonging simultaneously to a domain and a task, by means of specialization of the concepts of domain ontologies and task ontologies. They generally correspond to roles played by the domain entities when executing an activity.

On the other hand, Fensel [30] established the following alternative classification:

- *Generic or common-sense ontologies*: Capture general knowledge of the world. They provide basic notions and concepts for space, time, state, events, etc, and are valid for a variety of domains.
- *Representational ontologies*: Do not belong to any particular domain. They offer entities without establishing what they might represent. Therefore, they define concepts which express knowledge in an object- or framework- oriented approach.
- *Domain ontologies*: Capture the knowledge valid for a particular type of domain (for example, electronics, medicine, etc.).
- *Method and task ontologies*: The former offer terminology specific to problem resolution methods, while the latter provide terms for specific tasks. Both offer a reasonable point of view as to the knowledge of the domain.

In our opinion, the two previous authors' classifications may be aligned according to the following model as shown in Fig. 2.1.

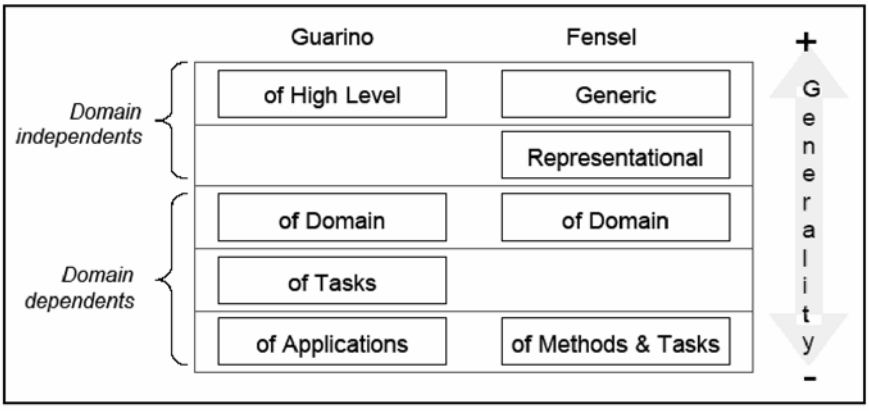

Fig. 2.1. Kinds of ontologies according to the generality level

In accordance with the type of conceptualization structure, Van Heijst and colleagues established the following kinds [94]:

- *Terminological ontologies*: Specify terms to be used to represent the knowledge of a studied domain. Then try to obtain a unified language

related to a specified field. An example of this type would be the ULMS (Universal Medical Language System).

- *Information ontologies*: Specify the structure of database records, determining a framework for the standardized storage of information. An example is the framework for modeling medical patient clinic records.
- *Knowledge representation ontologies*: Specify knowledge conceptualizations with an internal structure that exceeds those of the previous ones. They tend to be focused on a description of a particular knowledge use.

Another possible way of classifying ontologies is according to the nature of the real-world issue that is to be modeled. In this manner, Jurisica et al. have identified the following classes [55]:

- *Static ontologies*: Describe things that exist, their attributes and the relationships existing between them. This classification assumes that the world is made up of entities which are gifted with a unique and unchangeable identity. In these, we use terms such as entity, attribute, or relationship.
- *Dynamic ontologies*: Describe the aspects of the modeled world which can change with time. To model these it may be necessary to use finite state machines, Petri nets, etc. Process, state, or state transition are examples of terminology commonly included in this category.
- *Intentional ontologies*: Describe the aspects of the world of motivations, intentions, goals, beliefs, alternatives and elections of the involved agents. Some typical terms in these types of ontologies are aspect, object, agent, or support.
- *Social ontologies*: Describe social aspects such as organizational structures, nets or interdependences. For this reason they include terms such as actor, position, role, authority, responsibility or commitment.

Some authors believe that this linear way of classifying ontologies based on only a sole criterion does not allow for adequate reflection of the problem's complexity. Along these lines, Gómez-Pérez et al. [38] suggest a bi-dimensional classification, taking into account two criteria: the richness of the internal structure, and the subject of the conceptualization. The former criterion is based on a proposal of Lassila and McGuinness [59]. The latter proposes an extension of the Van Heijst et al. [94] classification previously described.

In this bi-dimensional proposal, every ontology belongs to one of the following categories, based on the level of richness of its internal structure:

- *Controlled vocabularies*: Formed by a finite list of terms.
- *Glossaries*: Lists of terms with their definitions offered in natural language.
- *Thesauruses*: Differentiated from the previous categories in that they offer semantic additions to the terms, including synonyms.
- *Informal hierarchies*: Hierarchies of terms which do not correspond to a strict subclass. For example, the terms "rental vehicle" and "hotel" could be modeled informally under the hierarchy "travel" as they are considered key parts of traveling.
- *Formal hierarchies*: In this case, a strict "is-a" relationship exists between instances of a class and of its corresponding superclass. For example, a teacher "is-a" people. Its objective is to exploit the inheritance concept.
- *Frames*: Ontologies which include such classes as properties, which can be inherited by other classes in lower levels of a formal "is-a" taxonomy.
- *Ontologies with value constraints*: Include value constraints. The most typical case is that of constraints dependent on the data type of a property (for example, a day of the month must be lower than 32).
- *Ontologies with generic logical constraints*: These are the most expressive ontologies which permit specific constraints between the terms of the ontology using first-order logic.

Simultaneously, depending on the subject of the conceptualization, an ontology falls into one of the following types:

- *Knowledge representation ontologies*: Capture representation primitives used to formalize knowledge under a concrete paradigm of knowledge representation.
- *Common or generic ontologies*: Represent common-sense knowledge reusable in distinct domains, for example, vocabulary related to things, events, time, space, etc.
- *High-level ontologies*: Describe very general concepts and notions by which they can be related to root terms of all ontologies. An unresolved problem is that many of these high-level ontologies differ in their way of classifying general concepts. This makes it difficult to integrate and exchange ontologies.

- *Domain ontologies*: Reusable ontologies of a particular domain (for example, medicine, engineering, etc.). They offer a vocabulary for concepts related to the domain and its relationships.
- *Task ontologies*: Describe the vocabulary related to some generic activity. They provide a systematic vocabulary of terms used to solve problems that may or may not belong to the same domain.
- *Domain task ontologies*: Unlike the previous ontologies, these are reusable in a given domain, but not among different domains.
- *Method ontologies*: Provide definitions of relevant concepts and their relationships. They are applicable to a reasoning process specifically designed to carry out a particular task.
- *Application ontologies*: Are dependent on the applications. Often, they extend and specialize the vocabulary of one domain ontology or task ontology for a particular application.

In the bibliography of ontologies, the adjectives formal, informal and semi-formal are also used. In this case, the formality of the language used to represent the ontologies is being indicated. This way, the ontologies expressed using natural language are considered to be totally informal, whereas those represented using first-order logic are formal [92]. In an intermediate situation, there is the ontology represented using UML class diagrams, as UML is considered semi-formal. In this case, the level of formality may be raised using OCL to model constraints. In relation to languages and techniques used to represent ontologies, SET experts may be interested in reading "Modeling ontologies with software engineering techniques" and "Modeling ontologies with database techniques", both of which are sections included in the first chapter of the book by Gómez-Pérez et al. [38].

These numerous and varying ways of thinking about ontologies have been clarified by some researchers who have looked for an integral definition which would serve for the different fields of application (knowledge engineering, databases, software engineering, etc.), and so as to be understood by non-experts. In this manner, Uschold and Jasper elaborated the following characterization (not definition) [92]:

> *An ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning. This includes definitions and an indication of how concepts are interrelated which collectively impose a structure on the domain and constrain the possible interpretations of terms.*

With the same goal, Gómez-Pérez et al. [38] conclude that "ontologies aim to capture consensual knowledge in a generic way, and that they may be reused and shared across software applications and by groups of people".

2.2.1 Heavyweight Versus Lightweight Ontologies

In the ontological engineering community it is common to hear of light- and heavyweight ontologies. This distinction is a simplification of the classification based on the level of richness of their internal structure (as previously commented), whereby lightweight ontologies will be principally taxonomies, while heavyweight ontologies are those which model a certain knowledge "in a deeper way and provide more restrictions on domain semantics" [38]. The former include concepts, concept taxonomies, relationships between concepts, and properties that describe these concepts. The latter add axioms and constraints, in order to clarify the meaning of the terms.

In Fig. 2.2 we have represented linearly the continuum from lightweight to heavyweight ontologies. In the upper part of the line, we find the lightweight ontologies which include controlled vocabularies, glossaries, and thesauruses; while at the bottom we find the heavyweight ontologies with value constraints and general logic constraints. In between are the informal hierarchies, formal hierarchies and frames. These intermediates have some of the characteristics of the heavyweight ontologies but not all authors consider them to fall within this general category.

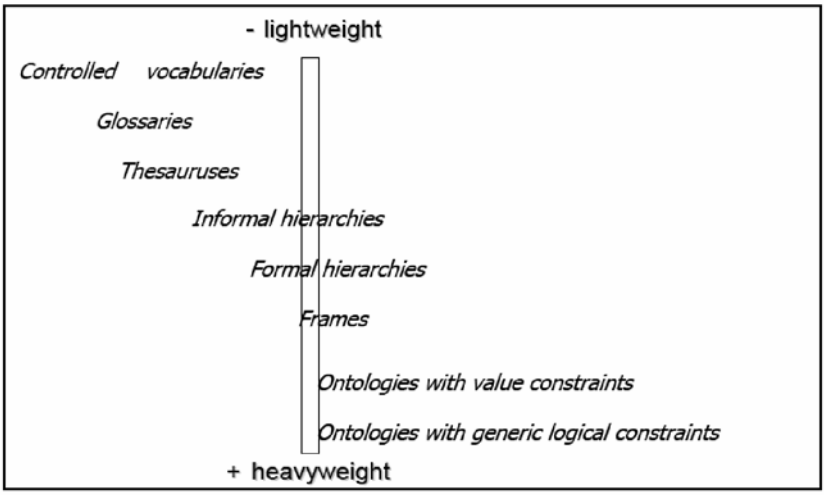

Fig. 2.2. A continuum from lightweight to heavyweight ontologies

This continuum from lightweight to heavyweight can be viewed as the two arms of a balance. The first has the advantage of being simple and the second, of being powerful. It is not possible to possess both advantages at the same time, and there is no way of determining which is better than the other, lightweight or heavyweight. It all depends on one's goals and necessities based on the particular case at hand. For example, the lightweight ontologies are more useful when the objective is, simply, to share knowledge of one domain between people. On the other hand, if it is necessary to execute some sort of logical inference or automatic calculation, it will be necessary to utilize the heavyweight ontologies. In any case, the following advice might serve the SET stakeholders: "use the lightest ontologies possible which can serve the necessities of the project at hand".

2.3 A Review of the Uses in SET

Of the utilities of ontologies in any field of human activity, we recognize the following to be principal:

- *Clarify the knowledge structure:* During the ontological analysis the domain concepts and relationships between them are defined in such a way that the adequate execution of this step eases the clear specification of the nature of the concepts and terms being used, with respect to the body of knowledge that is to be constructed [15].
- *Reduce conceptual and terminological ambiguity*: Ontological analysis provides a framework for the unification between people (and/or agents-systems) with differing necessities and/or points of view, depending on their particular context [91].
- *Allow the sharing of knowledge*: By means of an appropriate ontological analysis, it is possible to achieve a set of conceptualizations of a specific domain, and the set of terms which support it. With an adequate syntax, these conceptualizations and the relationships between them are expressed and codified in an ontology, which can be shared with any agent (person or system) having similar needs for the same domain [59].

Focusing exclusively on the scope of this book, many authors have studied and categorized the possible uses of ontologies in the software engineering and information systems disciplines. In these fields, it is possible to use ontologies of varying levels of generality. For example, the domain-level ontologies are especially useful for the development of reusable, high-quality software, as they provide a unambiguous terminology which can be shared

by all the development processes. Furthermore, thanks to ontologies, the eliciting and modeling of the requirements phase can be carried out in two steps [35]: in the first, general knowledge of the domain is elicited and specified in one or more ontologies. In the second step the obtained ontologies are used as inputs to develop the specified applications. The constructed ontology also serves as the basic vocabulary to speak about the domain and is a base for the development of the specific conceptualizations for the applications that are to be constructed.

Next we will summarize the results of some of the best known surveys.

For Pisanelli et al. the most important characteristics that ontologies offer the field of software engineering are [75]:

1) an explicit semantic and taxonomy;
2) a clear link between concepts, their relationships, and generic theories;
3) lack of polysemy within a formal context;
4) context modularization;
5) minimal axiomatization to pinpoint differences between similar concepts;
6) a good politic of name choice; and
7) a rich documentation.

Uschold, Gruninger and Jasper identified the following functions [91, 92]:

Communication: Ontologies allow for the reduction of conceptual and terminological ambiguity, as they provide us with a framework for unification. They allow us to share knowledge and facilitate the communication between people and/or systems as even those having differing necessities and viewpoints, a function of their contexts and particular interests. Furthermore, in any organization, there is implicit knowledge (for example, the normative models and the network of relationships between people) that can be made explicit through ontological means. Ontologies also permit an increased consistency, eliminating ambiguity and integrating distinct user viewpoints. For person-to-person communication, an informal, unambiguous ontology may be sufficient.

Interoperability: When different users or systems need to exchange data or when different software tools are used, the concept of interoperability has some important repercussions. In this sense, the ontologies can act as an "interlingua", that is, they can be used to support the translation between different languages and representations, as it is more efficient to have a translator for each part involved (with an exchange ontology) than to design a transla-

tor for each pair of involved parts (languages or representations). A paradigm case would be the use of ontologies in the Semantic Web to look for irrelevant language factors, that is, to obtain the same results when using the term "author" or "autor" (in Spanish).

System/software engineering: The application of ontologies to support the design and development of systems, specifically software, may have the following objectives:

- Specification: The role that ontologies play in specification depends on the level of formality and automization within the methodology of the system design. From an informal perspective, ontologies assist in the requirements identification process and in the understanding of the relationships between components. This is particularly important when there are different sets of designers working in different domains. From a formal perspective, an ontology offers a declarative specification of a system, allowing designers to argue over "why" the system is being designed instead of "how" to support its functionality.
- Confidence: The informal ontologies can improve the confidence of the system by serving as a basis for the manual checking of the design, while the formal ontologies allow for the semi-automized consistency check of a software system with respect to the declarative specification that the ontology presumes.
- Reusability: To increase its usefulness, an ontology should be able to support the import and export of modules (parts of the ontology). By characterizing the domain classes and tasks within these subdomains, the ontologies can provide a framework to determine the aspects of the ontology that can be reused between different domains and tasks. The objective is, therefore, to achieve libraries of ontologies that are reusable and adaptable to different classes of problems and environments.
- Search: An ontology can be used as metadata, serving as an index for a repository of information.
- Reliability: The consistency checking may be (semi-)automatic if a formal representation of knowledge exists.
- Maintenance: One of the main efforts made during the software system's maintenance phase is the studying of the system. For this reason, using ontologies allows an improvement of the documentation and a reduction in maintenance costs. Maintenance effort is also reduced if an ontology is used as a neutral authoring language because

it only has to be maintained in one site instead of in multiple places, one for each target language.
- Knowledge acquisition: In the process of building knowledge-based systems, speed and reliability may be increased when an existing ontology is used as the starting point and guide for the knowledge acquisition.

Several years after its publication, a study was re-performed by Gruninger and Lee. Greatly abbreviating, the results were the following [42]:

- Communication:
  - Between computer systems, for example, in the exchange of data between distinct software tools.
  - Between humans, for example, for the acquisition of a vocabulary that unifies concepts of a specific domain.
  - Between humans and computer systems, for example, an ontology may be deployed in a window so that the user can use it to better and more easily understand the vocabulary used in the application.
- Computational inference:
  - For the internal representation and management of plans and planning information.
  - For analysis of internal structures, algorithms, system inputs and outputs, in conceptual and theoretic terms.
- Knowledge reuse and organization:
  - For the structuring and organization of libraries or repositories of plans, and planning and domain information.

In addition to these previous possible uses of ontologies, Uschold and Jasper [92] have described scenarios for applying ontologies. These scenarios are abstractions of specific applications of ontologies following the same idea of Jacobson's use cases. Each scenario includes an overview with the intended purpose of the ontology, the role of the ontology, the main actors and the supporting technologies. These authors have established four categories which include all of the identified scenarios:

- *Neutral authoring*: An information artifact is authored in a single language and is converted into a different form for use in multiple target systems. Knowledge reuse, improved maintainability and long-term knowledge retention are the main benefits of this scenario.
- *Specification*: An ontology is created and/or used as a basis for specification and possibly also for the development of some software.

Benefits of this scenario include documentation, maintenance, reliability and knowledge reuse.

- *Common access to information*: When information is required by one or more persons or systems, but is expressed using unfamiliar vocabulary or in an inaccessible format, an ontology can help to render the information intelligible by providing a shared understanding of the terms, or by mapping between sets of terms. Interoperability and more effective use and reuse of knowledge resources are the main benefits of this scenario.
- *Search*: An ontology is used for searching an information repository for desired resources (for example, documents, Web pages, names of experts). The chief benefit of this scenario is faster access to needed information resources. The technology of the Semantic Web has this same goal, using the entire Web as a repository. Because of this, ontologies play a key role in this new technology.

Other authors have studied the impact of ontologies on information systems (ISs). For example, Guarino identified two dimensions that should be considered [43]:

- a temporal dimension, concerning whether an ontology is used at development or at run time (that is "for" an information system or "within" an information system), and
- a structural dimension, concerning the particular way an ontology can affect the main IS components.

With respect to the moment in which they are utilized, the use of the ontologies can take place during the development stage or during run time. On the one hand, when the ontology is used by the IS at run time, it is referred to as an "ontology-driven information system" proper. On the other hand, when it is used during development time, it is referred to as an "ontology-driven development of the information system".

By using ontologies *at development time*, two situations might occur: (1) that we have a set of reusable ontologies organized in libraries of domain or task ontologies; or (2) that we have a generic ontology (with less detailed distinctions at a domain level between the basic entities, and meta-level distinctions as for class and relationships types), with a more limited reusability grade. In the first case, the semantic content of the ontologies can be converted into a system component, reducing the cost of analysis and assuring the ontological system correctness (given that the ontology is correct). In the second scenario, which is more realistic, the quantity of ontological knowl-

edge available is more limited, but its quality may assist the designer in the conceptual analysis task.

When using an ontology *at run time*, one must distinguish between an "ontology-aware information system" and an "ontology-driven information system". In the first case, a system component has knowledge of the existence of a potential ontology and may make use of it with a specific proposal, while in the second case, the ontology is an additional component (generally, local to the system) which cooperates at run time in order to achieve the system's goals and functionality. One reason why ontologies are used at run time is to ease the communication between software agents, which communicate by means of messages containing expressions elaborated in accordance with the ontology.

With respect to the structural dimension, the three principal component types analyzed by Guarino for their impact are [43]:

- *Components of database*: To use an ontology at development time for the database component seems to be the most obvious use, because, in practice, an ontology has a great likeness to a database schema. In fact, some authors have created proposals whereby the ontologies play a key role during the phases of analysis and conceptual modeling [94]. The resulting conceptual model can be represented in a format understood by a computer and from there be projected to a concrete platform. During run time, there are various ways in which ontologies and databases can work together. For example, the explicit ontologies' availability as an information resource is basic in the mediation-based focus of information integration.
- *Components of user interface:* In this type, the ontologies have been used successfully in order to generate interfaces based on forms that perform data control by means of type violation constraints. Another example of use, in this case during run time, consists of deploying an ontology in a help window so that the user may use it as part of the system, for example, to understand the given vocabulary.
- *Components of application program:* The application programs tend to have much implicit knowledge about the domain, for example, in the type or class declarations, in regards to business rules or policies, etc. At development time, it is possible to generate the static part of a program with the help of an ontology. Further, ontologies which are integrated with linguistic resources may be used to assist in the development of object-oriented software, as expressed with the databases. At run time, it is possible to represent in explicit form (with an ontology) the knowledge that the program holds implicitly,

converting the program into a knowledge-based system. This could improve the maintenance, the extensibility and the flexibility of the system.

In the following sections of this chapter, we present a state of the art review in which the reader can find the most developed examples of these and other ways to use ontologies in SET.

2.3.1 Ontology Versus Conceptual Model

In the SE and IS communities, perhaps due to the historical importance of conceptual modeling, there is frequent confusion between ontology and conceptual models. In some sense, an ontology has a similar function to a database schema because the first provides meta-information that describes the semantics of the terms or data, but there are several important differences between these concepts [44, 63]:

- Languages for defining and representing ontologies (OWL, etc.) are syntactically and semantically richer than common approaches for databases (SQL, etc.).
- The knowledge that is described by an ontology consists of semi-structured information (that is, texts in natural language) as opposed to the very structured data of the database (tables, classes of objects, etc.).
- An ontology must be a shared and consensual conceptualization because it is used for information sharing and exchange. Identifiers in a database schema are used specifically for a concrete system and do not have the need to make an effort to reach the equivalent of ontological agreements.
- An ontology provides a domain theory and not the structure of a data container.

With didactic intention, Mylopoulos [67] explains with samples that an ontology is not a conceptual schema. This researcher uses the following sample situation. On one hand, there may be a university ontology defining and associating concepts such as student, course, lectures, etc. On the other hand, a conceptual schema, say, for the scholarship IS at the University of the World, may use these concepts but they are specialized in meaning. For example, the student concept may be meant to have as instances only 2005–2006 University of the World students. An ontology is meant to be reusable, whereas a conceptual schema is less so.

Spyns et al. [86] establish that the main difference between the data models and ontologies is that while the former are task specific and implementation oriented, the latter should be as much generic and task independent as possible. In this manner, to the benefits of reusability and reliability mentioned by Ushold and King [93] when ontologies are used in software and system engineering, we can also add shareability, portability and interoperability. These characteristics are identified as the common notion of "genericity".

2.3.2 Ontology Versus Metamodel

There also exists some confusion between ontologies and metamodels, which in our opinion is motivated principally because of the fact that both are frequently represented by the same languages, although their characteristics and goals are different.

Bertrand and Bezivin [7] have analyzed the relationship between low-level ontologies and metamodels, and have arrived at the conclusion that while metamodels look to improve the rigor of similar but different models, an ontology does the same but for knowledge models. Devedzic [24] noted another difference: without an ontology, different knowledge representations of the same domain can be incompatible even when using the same metamodel for their implementation.

The existing confusion is also generated due to the lack of agreement as to the definition of both terms. In the case of ontologies, we have already commented sufficiently on this fact. Similarly, for metamodels there exists no other universal consensus than the mere etymological description that a metamodel is a "model of models".

In our opinion, if one uses the definition of ontology proposed by Gruber [40] and the Object Management Group definition of metamodel, proposed in the "Model-driven Architecture" [71], the clearest distinction between them is that of intention: while an ontology is descriptive and belongs to the domain of the problem, a metamodel is prescriptive and belongs to the domain of the solution.

In Chap. 9 of this book, the reader is provided with a detailed proposal of the different roles played by ontologies and metamodels in the framework of a model-driven engineering paradigm. Also, a new idea, that of the megamodel, is introduced.

2.3.3 Ontologies in Software Engineering Environments

Other application fields for ontologies are the SEE (SEEs), which integrate diverse types of tools in order to assist the engineers in completing the software engineering processes. To begin with, in the SEE, knowledge is embedded in one or various tools or assistants but this makes it virtually impossible to be shared or reused.

The exchange of knowledge between humans is one of the major problems in software engineering projects. It has been shown that this is due in great part to the fact that the project participants have distinct domains of problem knowledge and/or use different languages, both problems which could be mitigated by using ontologies. This is why some authors have proposed the use of ontologies as the backbone of the tools and SEE [22]. For the same reason, there exist proposals of SEE architectures based on ontologies [28].

Two of these proposals will be commented on in the following subsections.

2.3.3.1 MANTIS Environment

An MANTIS is "eXtended Software Engineering Environment" for the management of software maintenance projects. By using the nomenclature "extended SEE" the intention is to emphasize the idea of integrating and widen the concepts of methodology and SEE [79]. All the MANTIS components are considered as tools of three different categories: conceptual, methodological and technical (CASE tools). A summary of the components that make up the MANTIS environment is shown in Fig. 2.3.

Conceptual tools are used in MANTIS to represent and to manage the inherent complexity of software maintenance projects. A level-based conceptual architecture is necessary to be able to work with different abstract levels. A software life cycle process framework is useful for knowing which are the other software processes related to the maintenance process. To make sure that all the concepts are correctly defined, used and represented, a set of ontologies was defined. The Maintenance Ontology represents the static aspects. They describe the concepts related to maintenance and consist of a subontology for products, another for activities, a third for processes and the fourth for describing the different agents involved in the maintenance process [79]. The intentional and social aspects are considered within the same subontology, Agents, since they are closely related. The dynamic part is represented by an ontology called Workflow Ontology, where three relevant aspects of the maintenance process are defined: decomposition of activities, temporal constraints between activities, and control of the execution of ac-

tivities and projects during the process enactment. A third ontology called a Measure Ontology represents both static and dynamic aspects related to the software measurement. This ontology was included because of the importance of measurement within the software process.

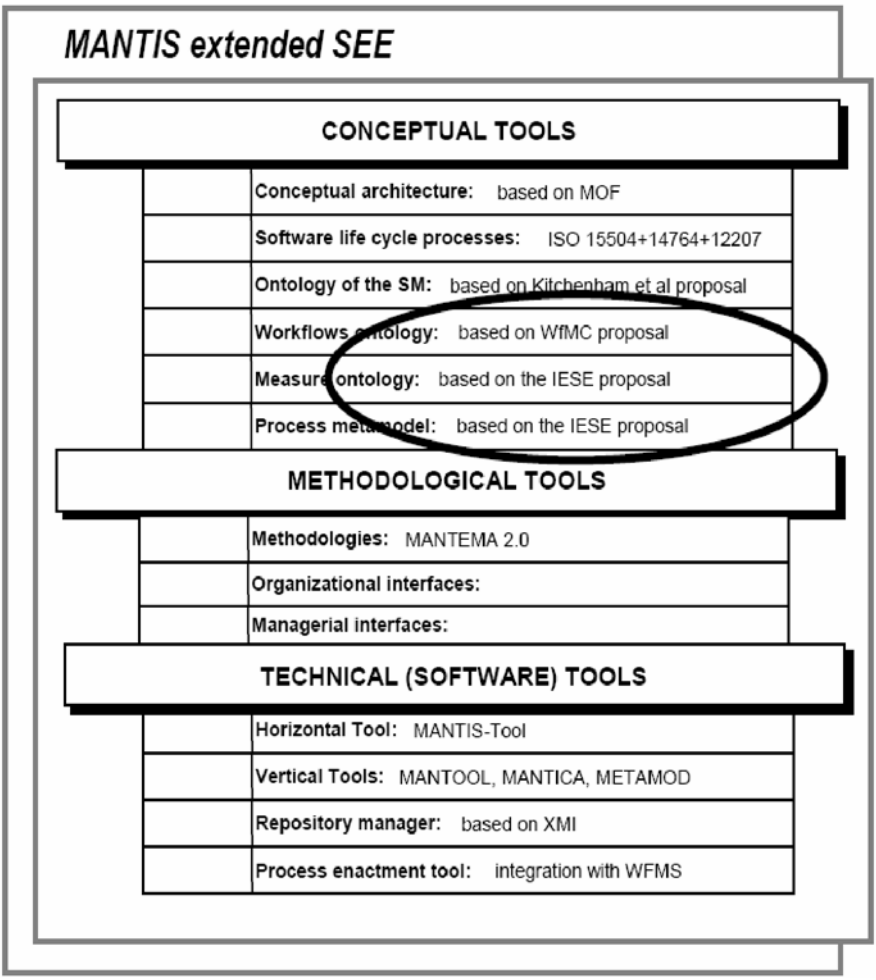

Fig. 2.3. Ontologies as conceptual tools in the MANTIS environment

The uses of the ontologies proposed in the MANTIS environment are two of the three identified by Gruninger and Lee [42]: communication (especially between humans participating in maintenance projects, and between humans and the software system of the MANTIS environment), and knowledge reuse and organization. On the other hand, the computational inference has not

been included in this SEE. The importance of ontologies' use as a support for maintenance activities (particularly for the sharing and reuse of knowledge about the product and its characteristics) has been recognized by other authors as well [21].

In MANTIS the ontologies have been represented using an adaptation of the REFSENO method (see later section).

2.3.3.2 TABA Workstation

TABA Workstation is a meta-SEE, capable of generating, by means of instancing, specific SEEs adequate for the particularities of a software process, of an application domain or of a specific project [28]. Given that the meta-Environment, the created SEE instance and the tools in the TABA Workstation need to handle knowledge of the software development process, this system includes an ontology whose end is "to support the acquisition, organization, reuse, and sharing of Software Process knowledge". This software development process ontology consists of various subontologies: of activities, of procedures and of resources.

For the graphic representation of these ontologies, GLEO (Graphical Language for Expressing Ontologies) is used along with a set of axioms defined in first-order logic. Also, for each ontology, the vocabulary used is defined in a table created by two columns, one with the concept name, the other with descriptions of its function and relationship with other concepts.

2.3.4 Representing Ontologies Using Software Engineering Techniques

There are many languages, techniques and tools for the representation, design and construction of ontologies (see Chap. 1). But the great majority of these have been created for and by the knowledge engineering community. Because of this, the use of ontologies by SET professionals and researchers can be seen as an additional learning experience, and in some cases, of considerably great effort.

To avoid this problem, UML has been proposed and analyzed as a language of ontological representation in software engineering [97]. Further, the ontological fundamentals of this option have been studied by Guizzardi et al. [45]. Other potential advantages of this choice is that the extension possibilities of UML can be used: descriptive or restrictive stereotypes, and regular or restrictive extensions of the UML metamodel [82].

For more detail regarding the use of UML as a representation language of ontologies, the reader may refer to “Modelling ontologies with software engineering techniques” in Chapt. 1 of the book “Ontological Engineering” [38].

2.3.4.1 REFSENO

Some SET researchers have made an effort to approximate previous proposals in the area of artificial intelligence, to the software engineering community. A significant case of this type is that of REFSENO (Representation Formalism for Software Engineering Ontologies) [90], a proposal created by the Fraunhofer Institute for Experimental Software Engineering (IESE) in Germany, which includes a methodology in order to develop the ontologies, together with a guide for their representation, through tables and diagrams.

REFSENO provides constructs (primitives) to describe concepts where each concept represents a class of experience items. Besides concepts, its properties (named terminal attributes) and relationships (non-terminal attributes) are represented. One relevant feature of REFSENO is that it enables us to describe similarity functions, which are used for similarity-based retrieval. In this way, the implementation of retrieval components is facilitated. This similarity extends the formalism of Ostertag et al. [73] by additional integrity rules and by clearly separating the schema definition and characterization. On the other hand, REFSENO also incorporates integrity rules such as cardinalities and value ranges for attributes, assertions and preconditions.

In the hope of better adapting the characteristics and interests of software engineers, and in contrast with the usual codified knowledge in knowledge-based systems, REFSENO represents the knowledge in the form of documents having a set of templates of tables and diagrams. This election is based on the studies of Althoff et al. [1] in which an important reduction in learning effort is achieved by the storage of experiences in the form of documents.

The methodology proposed by REFSENO is an improved adaptation of METHONTOLOGY [29, 37], which imitates the software life cycle proposed by the IEEE 1074 standard. Consequentially, the main steps are:

1. Planning.
2. Specification of the ontology requirements.
3. Conceptualization. This stage is similar to the phase of design in a software system, so it is not the ontology itself.

4. Implementation. This refers to the representation and storage of the previous conceptualization through use of computer tools.

REFSENO has been used for the creation and representation of diverse ontologies. For example, in [80] an ontology for software maintenance projects management, developed by a group of software engineers and researchers, is represented using REFSENO, changing specific diagrams for UML class diagrams and with other minor adjustments. According to the authors, they chose REFSENO for the following reasons:

- It allows for the modeling of software engineering knowledge in a precise and complete manner, by using alternate representations. The ontologies specified using REFSENO are precise, since the semantic relationships are defined and are complete, in the sense that all conceptual knowledge necessary to instantiate an experience base are provided.
- It has a clear terminology, differentiating between conceptual and context-specific knowledge, thus enabling the management of knowledge from different contexts.
- It guarantees a consistent ontology since consistency criteria must be fulfilled.

2.3.5 Experiences and Lessons Learned in Software Engineering Research

In this section we present some lessons learned about the usefulness of the ontologies in software engineering research. In these, we have reflected on the experience of the Alarcos Research Group (University of Castilla-La Mancha, Spain), which has been achieved through the development of various research and development (R&D) projects. In our opinion, these conclusions and commentaries can be extended to ISs and database research, and in part to the professional work of the software engineer.

At the origin of the use of these conceptual tools were two challenges encountered in the research projects: the integration of knowledge and the automation-oriented approach by means of software tools.

The first challenge arose from the common daily difficulties in human relationships (between memberships of our group, other groups and other stakeholders), causing a waste of time and energy, due to lack of explicit or tacit shared knowledge. The second challenge arose because the great majority of projects confronted involved the design of advanced support

tools for software engineering activities, which should offer the greater functionality that is possible at lower development cost.

In facing these challenges, the following two questions arose:

1. How can we achieve proposals, methods, or tools which offer more general solutions, that is, more useful for all, in research problems?
2. How can we more easily share knowledge of the different participants (researchers, groups, clients, users, managers, etc.)?

The conceptual architectures including meta-metamodels and ontologies have been the two conceptual tools best answering these questions.

The second question had the best solution when using ontologies. Of the many applications of ontologies that are identified in the bibliography [42], and that have already been commented on, for our software engineering R&D project they have been especially useful in:

1. Sharing problem domain knowledge and allowing the use of common terminology between all stakeholders (and not just the researchers).
2. The "filtering" of knowledge upon defining the models and metamodels.

This first use is evident, but its importance was considerable in the problems faced. This importance arose due to the need for communication as a main activity (in duration and importance) in R&D projects (as well as in any other type of work in software engineering or computer science) and because the ambiguity of the natural language implies errors, misunderstandings and unproductive efforts. It has been shown that this is due in great part to the project participants having differing knowledge of the domain of the problem, as well as the use of different languages, both problems which an ontology can mitigate.

The second more important use that we have found with ontologies is the filtering of knowledge (Fig. 2.4). The models and metamodels (models of models) are representations or images of reality that, by definition, only include a part of this reality. However, this is not a problem, but an assistance, as this precise factor allows for the filtering capability of undesired characteristics. In this sense, an ontology is also of assistance in deciding what should be taken out of the real systems in order to construct the model(s) of a system (correspondents at the M1 level in a conceptual architecture such as that defined in MDA-MOF [71]), or what should be taken into account in order to define metamodels (level M2 of MDA-MOF).

Although a formal and implemented ontology in a computer-adapted format may serve for knowledge inference, the characteristics of our R&D projects (with software engineering and not knowledge engineering goals) have led us to limit the use of ontologies to those of knowledge sharing and filtration. Therefore, the decision to use lightweight and non-formal (or semi-formal) ontologies has been due to the scope of projects which have been undertaken until today.

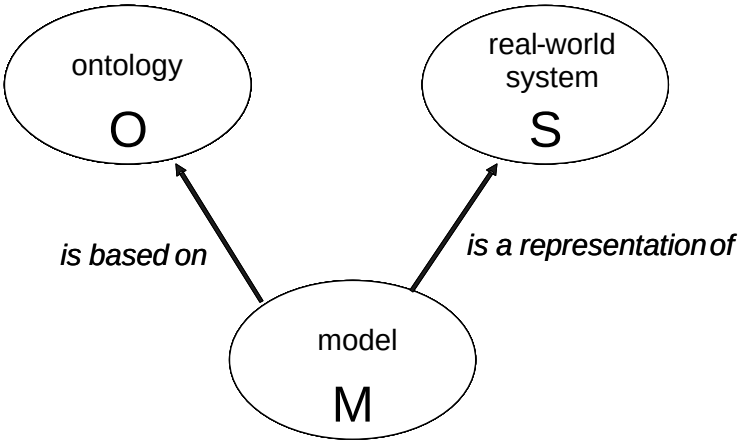

Fig. 2.4. Ontologies as filters of knowledge when defining models and metamodels

2.3.5.1 Examples

In the Alarcos Research Group we have carried out several R&D projects for software maintenance. For example, several years ago, we developed, in collaboration with the international company Atos ODS (previously Sligos), the MANTEMA methodology [76], specifically for the maintenance of software. In these projects, it was very useful to define an ontology for "managing project maintenance" [80] that solved previous misunderstanding and discussions due to, for example, not all participants (researchers, clients, maintainers) having equal understanding of the "modification request" concept.

On the other hand, in 2003, various groups from Spain and diverse countries of America held a meeting in order to define a metamodel which would permit the representation and implementation or any type of software measure. After several days of debating, it was evident to all that there did not even exist any agreement on the concepts and terms that the different researchers or groups used. Without this prior step, it was very difficult to con-

tinue advancing. For this reason, a work group was created in order to elaborate a "Software Measurement Ontology" [34]. Thanks to this ontology, the diverse groups have available a conceptual "filtering" tool to help them to create specific metamodels and models for their research. Further, it has been possible to more easily debate and truly center oneself in the reasonable differences due to distinct points of view or work philosophies. As a continuation of this work, a study was undertaken of the different standards and international norms, and it has been discovered that we are far from reaching this explicit and shared conceptualization (ontology). Even for the core concepts of "metric", "measure" and "indicator" there is no international consensus (aggravated by the inconsistencies and lack of the ISO and IEEE official standards) [33].

This absence of a prior ontology is a very common problem in software engineering and in computer science in general. Therefore, when trying to work with the new proposal of the SQL:2003 standard (14 parts and approximately 2000 pages), the Alarcos Group had major misunderstandings, because the metamodel indicated in part 11 "SQL/Schemata" [51], represented in the form of relational schemas, is illegible and has inconsistencies with other parts of the standard. These problems can be solved, or at least considerably reduced, if all the parts and thousands of pages of this standard are based on a previous ontology which makes clear the concepts and their relationships, independent of syntax and implementation aspects.

With this goal, we have begun to construct an SQL:2003 ontology, although due to this great challenge, we have opted to divide this task into various stages. Firstly, we have elaborated the ontology of the object-relational features [14] that we have represented by means of UML 2.0 class diagrams and texts organized in the form of tables. Additionally, in order to increase the level of formality, we have used OCL well-formedness rules. The ontology has been checked by mean of instantiation of an example in which most of the new object-relational features of the SQL:2003 standard are presented. It is of interest to remark that during the development process of the ontology, some inconsistencies were detected in the SQL:2003 standard.

In addition to improving didactics and easing the understanding and learning that this ontology has provided, it also has allowed us to start exploring some new research tracks the first of these being the ontology-based formalization of a set of complexity measures for object-relational schemas [5].

To conclude, in our experience with ontologies in reference to standards, we believe that ease of reading and understanding the standards would be greatly improved if the typical lists of terminology were substituted with an ontology containing the relationships between there terms (if possible, using some type of ontological diagram), its most significant properties and the

main semantic constraints. Furthermore, ontology would be a tool of great use for the verification and validation of standards.

2.4 A Proposal of Taxonomy

In previous parts of this chapter, we have examined distinct types of ontologies and the possible ways of employing them. In this section, we present a taxonomy especially oriented to assist SET professionals and researchers. Although we use the previously described classification ideas, we believe that SET community viewpoints and interests require a new and specific taxonomy. Concretely, we claim, without being experts in subjects such as ontological engineering, the Semantic Web, or knowledge engineering, that this taxonomy will assist in answering the following questions:

> *What ontologies exist to better understand the knowledge of a determined SET issue or subdomain?*
>
> *Why and how can we use ontologies in software development or maintenance projects?*
>
> *What proposals of new methodologies or previous adaptations exist for the construction of ontology-driven software?*
>
> *What types of software artifacts can be formed by or include ontologies?*

When attempting to establish a relationship between ontologies and SET, the former are typically considered to be another technique or artifact to be applied in the software life cycle processes; however, although less typical, it is also possible to use this type of conceptual tool for the representation of SET domain knowledge. This should not be forgotten when establishing a taxonomy or classification of the possible combinations between both fields. Thus, at a basic level, we propose that the ontology taxonomy for SET be formed by the following two generic categories:

- ***Ontologies of domain***: Describe knowledge of the SET domain.
- ***Ontologies as software artifacts***: Used as artifacts of diverse types, in some software process.

Following a description of the fundamental characteristics of ontologies belonging to these categories, and also the subcategories that we propose in both cases, is presented.

2.4.1 Ontologies of Domain

This generic category refers to the ontologies whose main goal is to represent (at least partially) knowledge of a certain subdomain within SET matter. The existence of a universal ontology to fully conceptualize this domain of knowledge would assist in the resource annotation and localization, for example, in the Semantic Web, and would avoid the ambiguities and inconsistencies which are commonly produced when computer science academics, researchers and professionals use varying terms and concepts.

As previously indicated, there are various forms of classifying the ontologies of a domain of knowledge; however, with SET ontologies, we believe that the classification should be based on norms, recommendations and standards published by prestigious organizations and associations (such as the IEEE or ACM), having been accepted and very well known by the international community dedicated to this discipline. In order to establish the hierarchy of subcategories, we have adopted the following:

- Firstly, to distinguish software engineering from software technology, as established in the "Overview Report" of the Computing Curricula 2005 [3].
- Within software engineering, to distinguish between the generic proposals that include the complete scope of this discipline and the specifics focused on some part of it.
- For these last ontologies, to employ the classification in 10 knowledge areas defined in the 2004 version of the "Software Engineering Body of Knowledge" (SWEBOK) [49].
- Within the field of software technology, to use the extended taxonomy of the "ACM Computing Classification System" [50] to identify the subcategories with two breakdown levels.

The ACM taxonomy categories and subcategories that have been considered are those whose content refers to the field of software technology. That is to say, those identified by the letters D (software, but without software engineering because this topic corresponds to the previous category), E (data in

general) and H (information technologies and systems, especially databases and Web systems).

In this chapter and book, we do not consider "Artificial Intelligence" within the umbrella of "Software Engineering and Technology". For this reason, in our proposed taxonomy, ACM category "1.2 Artificial Intelligence" (under the generic category of "I. Computing Methodologies") is not included, despite the fact that it includes such topics as "Ontology Design" and "Ontology Languages" within subcategory "1.2.12 Intelligent Web Services and Semantic Web".

Taking these factors into consideration, the taxonomy of the "ontologies of domain" is as follows:

- **Software Engineering (SE)**
  - *Generic (all-domain)*
  - *Specific (sub-domain)*
    - Software Requirements
    - Software Design
    - Software Construction
    - Software Testing
    - Software Maintenance
    - Software Configuration Management
    - Software Quality
    - Software Engineering Tools & Methods
    - Software Engineering Process
    - Software Engineering Management
- **Software Technology (ST)**
  - *Software*
    - Programming Techniques
    - Programming Languages
    - Operating Systems
  - *Data*
    - Data Structures
    - Data Storage Representations
    - Data Encryption
    - Coding and Information Theory
    - Files
  - *Information Technology and Systems*
    - Models and Principles
    - Database Management
    - Information Storage and Retrieval
    - Information Technology and Systems Applications

- Information Interfaces and Representation (HCI)

In order to simplify, in the SWEBOK-based discussion, only one level (knowledge areas) has been described, and in the ACM taxonomy section, we only detail two levels (generic categories and categories).

"*Software Engineering generic ontologies*", also denominated as "Software Engineering all-domain ontologies", has the ambitious objective of modeling the complete software engineering body of knowledge. Therefore, it can be based on three different source types: (1) glossaries (of the IEEE, for example), (2) body of knowledge guides (as SWEBOK), and (3) books of reference in the matter (Pressman [77], etc.). On the other hand, "*Software Engineering specific ontologies*" only attempts to conceptualize one part (subdomain) of this discipline, of interest for a determined goal, collective, or moment. As might be expected, there are many more proposals in this category than in the previous one.

On the other hand, some ontologies of SET subdomains are elaborated, taking into account the possibility of their integration with others, in order to extend the knowledge that is represented in a common way. This can be a good idea, as it follows the well-known strategy of synthesis to design complex systems. Taking this to the extreme, the combination of ontologies of all subdomains included in the proposed taxonomy would result in an ontology of the complete SET domain. Unfortunately, the reality is that this goal is extremely laborious, not only due to its size but also due to the numerous problems of ontology integration and merging (for example, overlapping of concepts) and, as yet, satisfactory solutions do not exist for them. Although similarities are found with the problem of database views integration, the ontology literature states that the merging ontology process is more difficult, labor intensive and error prone [87]. In the literature, we find the experiences of SET community members, not being experts in knowledge engineering–ontology, commenting on their problems in carrying out the merging of ontologies [96].

2.4.2 Ontologies as Software Artifacts

In addition to the ontologies that conceptualize the knowledge of SET (sub)domains, there are other types of proposals that use ontologies as artifacts, with varying characteristics and functionalities, during the construction or functioning of software systems. Many authors have researched the usefulness of using ontologies in this way, even basing the software development process on this technology, and giving way to what Guarino [43] has termed "*Ontology-driven Information System development*". Au-

thors such as Pisanelli et al. [75] have assured that in the future, software will not be designed without using an ontological approach, given the shown effectiveness of this choice, particularly when adequate tools are available. And, as already presented in prior sections, there exists a great potential in the use of ontologies as knowledge's artifacts, for facilitating communication among project stakeholders and for avoiding the ambiguities of natural language, as well as for filtering knowledge when defining models and metamodels of systems to be developed [80, 96]. A pending task which may prove very interesting is the comparative study of the paradigm "Ontology-driven development" proposed by Guarino [43] and the new paradigm "Model-Driven Engineering" (MDE) [83].

Among these uses of ontologies, the World Wide Web Consortium (W3C), a main precursor in the use of ontology for the Semantic Web, also endorses the use of ontologies for software development, having recently created a work group to evaluate, among other possibilities, the potential for "Ontology Driven Software Engineering", "Ontology Driven Architectures" (ODAs) and the crossover between ontology engineering and software engineering [99].

When it comes to proposing a taxonomy or classification of the ontologies that have been used as software artifacts in recent years, it seems reasonable to do so as a function of the ontology's use as an artifact (requirements specification, system conceptual modeling, etc.). Given that the software artifacts can be employed either at development or at run time, we have opted for the first-level classification proposed by Guarino [43], where analyzing the usefulness of ontologies in the IS field distinguished between those artifacts used at system development and those used during system execution.

The first of these categories, that is, "**Ontologies as software artifacts at development time**", has been divided based on function of the software life cycle processes in which it is principally used. The process groups that we have used are defined in the ISO/IEC 15504-2 [53] and ISO/IEC 12207 [52] standards. To simplify, we have covered only two breakdown levels (process groups and process categories), without achieving a bottom level of individual processes. Basically, the distinction consists of taking into account whether the ontologies are used as artifacts in the strictly engineering processes (software development and maintenance) or in other complementary processes: support activities, project management, knowledge reuse, etc. The reference model of these standards groups the processes into three life cycle groupings which contain five categories. Table 2.1 summarizes this information.

In the case of the category referred to as "**Ontologies as software artifacts at run time**", following the same reasoning as Guarino [43], we have determined two different situations:

1. *Ontologies as architectural artifacts*: When ontologies are part of the system software architecture, as an additional component, ,cooperating with the rest of the system at run time to attain the software objective (ontology-driven software).
2. *Ontologies as (information) resources*: Are used by the software during run time for a specific purpose, as an information resource, normally remote, upon which the software operates (ontology-aware software), carrying out, for example, specific queries.

Table 2.1. Groups and categories of processes in ISO/IEC 15504-2 [53]

Group	Category	Description of included processes
Primary	Customer–Supplier	Directly impacts the customer, support development and transition of the software to the customer, and provides for the correct operation and use of the software product and/or service.
	Engineering	Directly specifies, implements, or maintains the software product, its relationship to the system and its customer documentation.
Supporting	Support	May be employed by any of the other processes (including other supporting processes) at various points in the software life cycle.
Organizational	Management	Contains practices of a generic nature which may be used by anyone who manages any type of project or process within a software life cycle.
	Organization	Establishes the business goals of the organization and development process, product, and resource assets which, when used by the projects in the organization, will help the organization achieve its business goals.

Taking into account all of the prior considerations, the taxonomy of "Ontologies as software artifacts" that we propose is the following:

- **At Development Time**
  - For Engineering Processes
    - Development process
    - Maintenance process
  - *For Other Processes*
    - Customer-Supplier processes
    - Support processes
    - Management processes
    - Organization processes
- **At Run Time**
  - *As Architectural Artifacts*

– *As (Information) Resources*

At development time and for engineering processes, the ontologies may be used as artifacts for requirements specification, conceptual modeling, programming, database design, or automatic generation of code. Use cases of ontological artifacts in other complementary processes are communication, software process management, configuration management, reuse, quality assurance, documentation, etc.

Examples of scenarios in which ontologies at run time can be used as architectural artifacts are ontology-driven software architecture, software agent architecture, Web service architecture, Web server architecture. On the other hand, ontologies as information resources at run time could be used in scenarios such as ontology-aware systems, ontology databases, software agents communication, Web services use, search engines or workflow execution.

2.5 Review and Classification of Proposals in the Literature

In this last section of the chapter, we present a summary of a large collection of SET ontology proposals. Each reference is briefly commented upon and situated within the taxonomy previously presented.

When classifying each analyzed work, its main contribution was taken into account. For instance, for the proposals using ontologies at run time of an application, it is evident that this ontology could have been created at development time. However, it has been just considered and classified in the first category, due to its great interest.

In a few cases, the proposal's characteristics have led us to make the decision to include it in more than one taxonomy category. The most typical cases of this type, although not the only ones, are proposals included both in the general category of ontologies of domain and in the ontologies as software artifacts. This happens when, for example, in Ruiz et al. [80], the same proposal includes an ontology as artifact as well as an ontology of software engineering or software technology domain, which complements the first.

2.5.1 Proposals of Ontologies of Domain

Since the late 1990s, different proposals have been published in order to elaborate ontologies of a part or of a complete knowledge domain of SET. In Tables 2.2 and 2.3 below, some of the most well known of these are presented along with their authors and taxonomy category (and subcate-

gory) classification, according to the conceptualized knowledge domain of the ontology.

The majority of the ontologies included in this subsection have not been developed with the sole objective of representing a conceptualization of a SET domain, but, rather, they have been created by their authors to obtain or to be part of systems based on semantic technology.

2.5.1.1 Software Engineering Ontologies

Based on the SWEBOK guide, prototypes of ontologies for the representation of the complete software engineering domain have been created [49]. This includes those of Mendes and Abran [64], consisting of an almost literal transcription of the SWEBOK text, with over 4000 concepts. Another proposal included is that of Sicilia et al. [85], which established an ontology structure based on a *description part*, to characterize artifacts and activities as created and enacted by current software engineering practice, as well as a *prescriptive part*, dealing with a different aspect of reality, which comprises the approaches or rules to concrete practical activities that are "commonly accepted" as considered in the SWEBOK. In Chap. 3 of this book, the authors of both works summarize the different approaches to develop an ontology of the SWEBOK.

Another less ambitious ontology, but one that also conceptualizes the entire software engineering domain, is *OntoGLOSE*, created and based on the "Glossary of Software Engineering Terminology" published by the IEEE [48]. It basically deals with a terminological ontology including over 1500 concepts, corresponding to 1300 glossary terms with their differing meanings [47].

The remaining ontologies presented in Table 2.2 are partial representations of the software engineering domain. Falbo et al. [28] and Larburu et al.[58], for example, have proposed ontologies to model the knowledge related to the **software process**, including concepts such as *Life Cycle Model*, *Software Process*, *Activity*, *Procedure*, *Task*, *Role*, or *Artifact*, among others.

In the second case, the ontology referred to as *SPOnt* and its authors have reused concepts included in other ontologies related to decision support systems, establishing links to concepts such as *Problem* (of the *MCDA* ontology) or *Guideline* (from the *GLIF* ontology).

Table 2.2. Proposals of ontologies of domain (software engineering subdomain)

Category / subcategory	Proposal	Author(s) and reference
Generic	Issues in the development of an ontology for an emerging engineering discipline	Mendes and Abran [64]
	The evaluation of ontological representation of the SWEBOK as a revision tool	Sicilia et al. [85]
	OntoGLOSE: a light weight software engineering Ontology	Hilera et al. [47]
Specific / Software Requirements	Conceptual design model-based requirements analysis in the Win–Win framework for concurrent requirements engineering	Bose [9]
	A generic ontology for the specification of domain models	Girardi and Faria [35]
	An ontology about ontologies and models: a conceptual discussion	Sánchez et al. [81]
	OpenCyc.org: formalized common knowledge	Cyc [19]
Specific / Software Design	An ontology about ontologies and models: a conceptual discussion	Sánchez et al. [81]
	OpenCyc.org: formalized common knowledge	Cyc [19]
	XCM: a component ontology	Tansalarak and Claypool [89]
Specific / Software Maintenance	A concept-oriented approach to support software maintenance and reuse activities	Deridder [21]
	Organizing the knowledge used in software Maintenance	Dias et al. [25]
	An ontology for the management of software maintenance projects	Ruiz et al. [80]
	Merging software maintenance ontologies: our experience	Vizcaino et al. [96]
	Towards an ontology of software maintenance	Kitchenham et al. [56]
Specific / Software Quality	Identifying quality requirements conflicts	Boehm and In [8]
	An ontological approach to domain engineering	Falbo et al. [27]
Specific / SE Process	Using ontologies to improve knowledge integration in software engineering environments	Falbo et al. [28]
	Towards the implementation of a tool for supporting the software development process (in Spanish)	Larburu et al. [58]
	An ontology for software development methodologies and endeavours	González-Pérez and Henderson-Sellers [39]
	Building a knowledge base of IEEE/EAI 12207 and CMMI with ontology	Lin et al. [60]
Specific / SE Management	Towards a consistent terminology for software measurement	García et al. [33]
	REFSENO: a representation formalism for software engineering ontologies	Tautz and Greese [90]

In Chap. 4 of this book, a software development methodology ontology is described, as proposed by González-Pérez and Henderson-sellers [39], which

includes a comprehensive metamodel plus a three-domain architecture that can be used to specify methodologies and then apply them to real endeavors.

Related to the knowledge domain of software process, Lin et al. [60] have realized initiatives for the creation of ontologies for the IEEE 12207 standard [76] that provides a guide for software life cycle processes, and CMMI [84] that can be applied in an organization to inspect and improve the capability of software process maturity. The goal of these authors is to combine these two ontologies in order to integrate the two knowledges (in the case of CMMI, only that relative to the maturity level 3) into a knowledge base capable of being applied by an organization in order to develop software more efficiently and correctly.

Ontologies of knowledge associated with the **software maintenance** process, having different focuses, such as the proposal by Deridder [21] or Dias et al. [25], have also been developed. In the latter case, the authors organized the ontology into five subontologies, in order to represent the knowledge related with the software systems in general, with the necessary skills required for software maintainers, with the activities of the maintenance process, with the organizational issues of the maintenance, and with the concepts and tasks that constitute any application domain. In Chap. 5 of this book there is a detailed description of the latest version of this ontology.

The ontology created by Ruiz et al. [80] is also structured in subontologies, although in this case, there are four: subontology of the products, of the activities, of the process organization, and of the agents. There is also an ontology created by Vizcaíno et al. [96] based on a combination of the previous ones. All these ontologies are based on the earlier work of Kitchenham et al. [56].

As for **software quality**, one of the first known ontologies was that used by Boehm and In [8] which included concepts related to the quality attributes of software systems, and information about the influence of software architectures and the development processes on these attributes. For example, the ontology includes relationships among the concepts of *portability, layered system architecture* and *prototyping*, in order to represent the following knowledge: "the *portability* quality attribute can be achieved when using a *layered system architecture* and a *prototyping*-based development". Falbo et al. [27] also proposed an ontology for the quality linked to concepts related to the software process, which were previously modeled in the form of a different, previously mentioned ontology [28].

Regarding the domain of **software measurement**, García et al. [33, 34] have created an ontology which attempts to establish a well-defined terminology for this field, with 21 interrelated terms that are based on four fundamental concepts: *measurement approach, measurement, measure and measurement results.* In Chap. 6, the principal characteristics and elements of this

ontology are presented. Similarly, Tautz and Wangenheim [90] have developed a highly detailed ontology of the GQM paradigm *(Goal Question Metric)*, exemplifying the use of the REFSENO notation to represent SET ontologies (to model the knowledge of the software measurement planning domain). In Table 2.2, these ontologies appear classified within the subcategory of "Software Engineering Management", as the SWEBOK breakdown of this area suggests.

Related to the domain of **requirements engineering**, there have also been proposed ontologies, such as that of *Win–Win* [9], carried out by Bose in order to represent the knowledge gained from the model of the same name (established by Boehm in order to manage the necessary collaboration and negotiation produced by those involved in this software life cycle stage).

There also exist ontologies which attempt to conceptualize knowledge with respect to **system modeling**, from a software engineering point of view. Sánchez et al. [81] created an ontology to reflect upon the different meanings of the term *model*, through the incorporation of different concepts related with this term, in their ontology, such as: *model as concept* and *model as original*. In Table 2.2 this proposal is found within the subcategory of "Software Requirements", as based on the SWEBOK breakdown which includes in this knowledge area the topic "Requirements Analysis" and within this, the subtopic "Conceptual Modeling".

The subontology of UML, integrated in the upper ontology *OpenCyc* [19], is more thorough than the previous ones, as it includes over 100 concepts and their definitions, and over 50 relationships, as well as some 30 instances, in order to represent the knowledge associated with this modeling technique. Some of the concepts included in *OpenCyc* are: *UMLModelElement*, *UMLClassifier*, *UMLClass* and *UMLStateMachine.* In Table 2.2 this ontology appears classified within the subcategory "Software Requirements", for the same reason as mentioned previously. But it also falls under the subcategory of "Software Design" as the SWEBOK breakdown includes the knowledge area topic "Software Design Notations".

Another proposal related to the domain of software requirements is *ONTODM*, an ontology created by Girari and Faria [35] for the construction of domain models to be reused in the development of multi-agent applications. ONTODM represents the knowledge of techniques for the specification of the requirements of a family of multi-agent systems in an application domain. It is being used as a CASE tool to assist in the elicitation and specification of domain models.

With respect to **component-based software engineering,** there exists an ontology known as *XCM*, created by Tansalarak and Claypool [89], with its purpose being "to provide a standard for the definition of components that crosscuts the different component models and unifies the variances between

the different models". This ontology includes concepts such as *Component*, *Method*, *Even*, *UndelayingComponent* or *Aggregation-basedComposition*. In Table 2.2, this proposal appears classified in the sub-category "Software Design" according to the SWEBOK breakdown, as this knowledge area includes the topic "Software Design Strategies and Methods" and, within this, the subtopic "Component-based Design".

2.5.1.2 Software Technology Ontologies

Among the software technologies that have been conceptualized through ontologies, we find the technologies related to the **programming languages**. For example, as part of the *SIMILE* project (Semantic Interoperability of Metadata and Information in unLike Environments) of MIT, for the creation of a collection of public ontologies [66], we find a very simplistic ontology of the Java language, which represents structural dependencies between the concepts of *Class* and *Package* of this language. Also related to Java is the proposal of Liu and Lo [61] who created an ontology based on the software architecture on which the J2EE technology is implemented.

Other interesting work linking ontologies and programming languages is that of Zimmer and Rauschmayer [101], who used a generic ontology of source code, with concepts such as *Code*, *Identifier* or *CodeAssociation*, in order to create programs as instances of these concepts (for example, a *Java class* would be an instance of the *Code* concept).

In the scope of **Web engineering**, very detailed ontologies have been developed for Web technologies, such as **Web services**, of which the *OWL-S* ontology stands out, being created in order to describe the properties and capabilities of Web services in an unambiguous, computer-interpretable form [62]. More recently, there is the *WSMO* ontology (Web Service Modeling Ontology), expressed in a more specialized language than the OWL, and referred to as WSML [78]. These two, previously mentioned, are characterized by their high level of detail; however, there are other ontology-oriented proposals about Web services that are more limited, such as that of Pahl [74], focused on the representation of conceptual elements necessary in order to consider Web services as a type of software component.

Also there are **software agents** technology ontologies, such as that of Brandão et al. [10], denominated as *MAS* (Multi-Agent System), that define the concepts and properties that can be used to represent dynamic models of applications based on software agents.

Other computer application types that have been the object of conceptualization are the **ubiquitous and pervasive applications**, which seamlessly integrate into the life of everyday users, providing them with services and in-

formation in an "anywhere, anytime" fashion. In this knowledge domain, Chen et al. [17] have proposed *SOUPA* (Standard Ontology for Ubiquitous and Pervasive Applications), which offers developers a shared ontology that combines many useful vocabularies from different consensus ontologies. Their objective is to assist the ubiquitous and pervasive applications developers who are inexperienced in knowledge representation, to quickly begin to building ontology-driven applications. SOUPA includes concepts such as *Agent* (to represent human users, with properties such as *Believes*, *Desires*, or *Intends*), *Action*, *Time*, *Device*, or *Location*.

All of the previous ontologies appear in Table 2.3 classified, according to the established taxonomy, in the category of "Software"; the first are under the subcategory "Programming Languages" and those related to Web technology and ubiquitous computing are found in the "Programming Techniques" category.

Table 2.3. Proposals of ontologies of domain (software technology subdomain)

Category / subcategory	**Proposal**	**Author(s) and reference**
Software / Programming Techniques	OWL-based Web service ontology	Martin [62]
	Web Service Modeling Ontology (WSMO)	Roman et al. [78]
	Ontology-based description and reasoning for component-based development on the Web	Pahl [74]
	Ontologies as specifications for the verification of multi-agent systems design	Brandão et al. [10]
	SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications	Chen et al. [17]
Software / Programming Languages	RDF ontology collection. SIMILE project	MIT [66]
	The study on ontology integrating and applying the ontologies of IEEE/EIA 12207, CMMI, Workflow and J2EE to Web service development environment	Liu and Lo [61]
	Tuna: ontology-based source code navigation and annotation	Zimmer and Rauschmayer [101]
Data / Data Encryption	Security mechanisms ontology	Denker [20]
Information Technology and Systems / Database Management	An ontological approach to the SQL:2003	Calero and Piattini [13]
Information Technology and Systems / Information Interfaces (HCI)	An ontology based method for universal design of user interfaces	Furtado et al. [31]
	A proposal of a knowledge model aimed at the use of questionnaires in the usability evaluation (in Spanish)	García [32]

Among the proposals that are found in the "Data" category of the taxonomy are those related to the **data encryption** techniques, such as the ontology of Denker [20], whose objective is to provide notations that will allow interfacing among the various standards for security and trust. This ontology includes concepts such as *SecurityMechanism*, *KeyFormat*, *Encryption*, *Signature*, *Protocol*, or *KeyProtocol*.

In the category "Information Technology and Systems", there are, among others, the proposals of different **database** technologies, such as those presented in detail in Chap. 7 of this book, related to the object-relational features of the SQL:2003 standard [13].

Within this category, but forming part of the subcategory "Information Interfaces and Representation", are the ontology proposals about technology related to **Human Computer Interaction** (HCI) and, particularly, within the domain of user interface. The proposal of Furtado et al. [31] of an ontology-driven interface design, includes the definition of ontologies at three levels: conceptual, logical and physical. On the other hand, García [32] has developed a detailed ontology of this domain which, in addition to the representation of general concepts of user interface design, also incorporates others related to the usability of the interfaces and their evaluation.

2.5.2 Proposals of Ontologies as Software Artifacts

The proposals for ontology use as software artifacts that can be found in the literature are more abundant than those oriented towards the conceptualization of the SET knowledge domain previously described. The importance of this new approach in software development shows how, recently, special events were being established in order to present such proposals, as in the case of the *Workshop on Ontologies as Software Engineering Artifacts*, hosted in 2004 as a specific event within the *International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA). The majority of the works presented at this event are described in this section.

In Tables 2.4 and 2.5 we present the analyzed proposals, organizeda and based on the taxonomy presented in Sect. 2.2.4, that is, as a function of ontologies as artifacts used: (1) at software development time (for the realization of the stated engineering processes or for other auxiliary processes); or (2) at software run time, as architectural artifacts (ontology-driven software) or as information resources (ontology-aware software).

Table 2.4. Proposals of ontologies as software artifacts at development time

Category / subcategory	Detail	Proposal	Author(s) and reference
Engineering / Development process	All phases	Ontology-driven software development in the context of the semantic web: an example scenario with Protegé/OWL	Knublauch [57]
	Analysis, design, coding	The role of ontologies in schema-based program synthesis.	Bures et al. [12]
	Analysis, design, coding	Building ontologies in a domain oriented software engineering environment	Mian and Falbo [65]
	Analysis, design, coding	Use of ontologies in software development environments	Oliveira et al. [72]
	Analysis, design, coding	An ontology based method for universal design of user interfaces	Furtado et al. [31]
	Analysis, design	Data modelling versus ontology engineering	Spyns et al. [86]
	Analysis, design	Ontology-based description and reasoning for component-based development on the Web	Pahl [74]
	Analysis	The use of ontologies as a backbone for use case management	Wouters et al. [100]
	Analysis	Simplifying the software development value chain through ontology-driven software artifact generation	Jenz [54]
	Analysis	Conceptual design model based requirements analysis in the Win–Win framework for concurrent requirements engineering	Bose [9]
	Analysis	Ontologies, metamodels and model-driven paradigm	Assmann et al. [2]
	Analysis	Improving analysis patterns reuse: an ontological approach	Hamza [46]
	Design, coding	Ontology-oriented programming: static typing for the inconsistent programmer	Goldman [36]
	Coding	Tuna: ontology-based source code navigation and annotation	Zimmer and Rauschmayer [101]
Engineering / Maintenance process		An ontology for the management of software maintenance projects	Ruiz et al. [80]
Non-engineering / Support processes	Quality assurance	ODE: Ontology-based software Development Environment	Falbo et al. [26]
	Verification, validation	The use of ontologies as a backbone for software engineering tools	Deridder and Wouters [22]
	Documentation	The use of an ontology to support a coupling between software models and implementation	Deridder et al. [23]

Table 2.4. *(continued)*

Category / subcategory	Detail	Proposal	Author(s) and reference
Non-engineering / Management processes		Ontology-based retrieval of software process experiences	Nour et al. [69]
		Toward the implementation of a tool for supporting the software development process (in Spanish)	Larburu et al. [58]
		ODE: Ontology-based software Development Environment	Falbo et al. [26]

2.5.2.1 Ontologies as Software Artifacts at Development Time

For Engineering Processes

Among the proposals found in this category is that of Knublauch [57], who defined a complete ontology-driven software development methodology oriented to Semantic Web applications, in which ontologies are used throughout the life cycle of an application, from development through execution.

The rest of the proposals for using ontologies at development time do not establish their use throughout the development process, but, rather, are limited to certain phases such as requirements analysis, design, or coding. The majority of the proposed works apply a domain-oriented software development to the software projects, based on the use of application domain knowledge to guide software developers across the several phases of the software process, facilitating the understanding of the problem during development. Authors such as Bures et al. [12], Mian and Falbo [65] and Oliveira et al. [72] propose to carry out the **domain analysis** through the creation of an ontology which, shortly, will be mapped in design models to be ultimately used to generate code in a determined programming language. This approach assumes the integration of ontology editors in the Domain-Oriented Software Development Environments (DOSDEs).

Bures et al. [12] proposed the automatic generation of code directly through a high-level specification, formed by models constructed from concepts of a given ontology that help to assure the consistency of the generated code.

On the other hand, Furtado et al. [31] established a design method for a user interface at three levels of abstraction, beginning with the creation of an ontology of the domain of discourse (conceptual level), and the subsequent elaboration of models (logical level) that capture instantiations of concepts identified in this ontology for producing multiple user interfaces for one design situation, and that exhibit different presentation styles, dialogues and

structure. These models are subsequently transformed into code for their execution in a determined technological platform (physical level).

Spyns et al. [86] used ontologies as an alternative to traditional **data modeling** for database design, defining a method called "DOGMA ontology engineering" (and using the *DogmaModeler* tool), which adopts a classical database model-theoretic view, in which conceptual relations are separated from domain rules; but in this case, through an ontological approach, by means of an "ontology base", which contains multiple intuitive conceptualizations of a domain, and "ontological commitments", where each commitment contains a set of domain rules.

Other authors have established analysis and design methods, based on ontologies, for the development of **component**-oriented software. In this way, Pahl's proposal [74] established the convenience of using ontologies not only for modeling the domain knowledge that corresponds to the components, but also for modeling the software-related knowledge, referred to the behavior of operations or services offered by the components to be developed. In this last case, the ontological concepts would represent descriptions of service properties, while the properties or roles would be the services themselves. WSMO [78] is another ontology thought for suitable this type of development, but oriented towards components implemented in the form of Semantic Web services.

The following group of proposals shown in Table 2.4 is formed by those which refer to the use of ontologies only during **requirements analysis.** In this way, Wouters et al. [100] established a method of requirements specification based on case models represented in UML, but complemented by an annotation mechanism based on an ontology of the application domain, in order to facilitate the management of large sets of use case, improving its browseability, maintainability and scalability. Jenz [54] suggested the creation of a business process ontology with concepts such as *BusinessActivity*, *BusinessRule*, or *BusinessDocument*, having two principal goals: to allow the sharing of knowledge between domain experts and people engaged in software development, and to serve as a requirements specification from which a number of software artifacts can be automatically generated, for example, UML class diagrams.

Another proposal is that of Bose [9], using an ontology of the *Win–Win* technique domain (previously commented), and with the objective of facilitating the semi-automatic transition of the system requirements, according to the mentioned technique, to the corresponding abstract design model. The author proposes the expansion of this ontology by including the conceptualization of the elements that constitute these high-level design models, creating a mapping between these and the elements used in the *Win–Win* requirements model.

Also within this group we find the proposal of Assmann et al. [2] for using ontologies in the case of **model-driven development**, to describe the domain of a system (see Chap. 9 of this publication). And that of Hamza [46], who affirms that ontologies can assist in the reuse of high-level generic solutions in determined problems (that is, *analysis patterns*, in analogy with the known *design patterns*), that avoid facing the analysis phase of a project from scratch. The proposed method has four phases: (1) Knowledge extraction, where a collection of existing patterns of another knowledge source are analyzed. (2) Ontology development, where an ontology that captures the extracted knowledge is developed. (3) Knowledge reuse, where the knowledge included in the ontology is converted into a knowledge asset that can be reused to construct analysis models. (4) Knowledge augmentation, whose objective is to discover new knowledge, upon developing an application, in order to incorporate it into the ontology.

The rest of the proposals of ontologies as software artifacts at development time for the development process included in Table 2.4 refer to the use in activities at a lower level: **design and coding**. For instance, Goldman [36] proposes a development method called "ontology oriented programming" in which the specification of a problem's solution is expressed in the form of an ontology, with its annotations, that is compiled to produce an ontology-specific library, which is linked with other libraries and code to produce an application. Annotations allow for trade-offs between the flexibility of the generated library and its performance. This is a programming paradigm of a higher abstraction level than object-oriented programming ("concepts" versus "objects"), but which finally, through the indicated compiler, makes it possible to generate object-oriented code.

Related to programming, Zimmer and Rauschmayer [101], with the goal of enriching the source code of applications constructed by applying the well-known agile methodology "Extreme Programming" (when "the code is the model"), propose a generic ontology for the source code and a tool with which they write annotations that can be added externally without changing the source code, and that offers the possibility of making queries or navigating through the (semantic) content of the programs created.

All of these described proposals have referred to the use of ontologies in the process of software development; however, works have also been published in relation to their use in the **maintenance** process, included in the taxonomy, along with development, in the engineering processes category. In the work of Ruiz et al. [80], an ontology to assist in the management of software maintenance projects is presented. Also, it includes some elements such as *product*, *activity*, *process*, *agent*, *measure* and some dynamic aspects such as *workflow*. This ontology has been the basis of the development of an "extended software engineering environment" to manage maintenance pro-

jects (called the MANTIS environment), previously presented in this chapter, and also has been used for the construction of a knowledge management system (KM-MANTIS) for improving and supporting the management maintenance projects.

For Non-engineering Processes

There have been some proposals for the use of ontologies in other processes than pure software engineering, development and maintenance, although fewer than those previously described. These include ontologies for the processes of management, quality assurance, verification, validation or documentation.

In the case of **management processes,** Nour et al. [69] developed ontology-based techniques and tools that allow recovery of the acquired experience in previous software projects to be applied to new projects. In order to achieve this, three different ontologies for annotating knowledge stored in an "experience base" were created: (1) Skill Ontology, that describes skills and qualifications required for performing specific task types (ex. Java programming). (2) Process Ontology, that allows the definition of process structures. (3) Project Ontology, allowing the representation of information of a project context. The objective is, for a project manager, to be able to query this experience base in order to obtain the information needed to plan the current project.

On the other hand, Larburu et al. [58] created a prototype of a decision support system to assist in the deployment of software development processes, which permit the modeling and execution of software processes previously defined based on a set of four linked ontologies. This prototype has a descriptive capability sufficient for defining roles, tasks, artifacts and decision problems as class instances (concepts) defined by the mentioned ontologies. The four ontologies used are *SPont* (of the domain of software process), *GLIF* (of the Guidelines Interchange Format), *MCDA* (of the domain of multi-criteria decision analysis) and *PROAFTN* (of a fuzzy classification methodology).

The proposal by Falbo et al. [26] is related to the subcategory of management processes, but also to the quality assurance process, which belongs to the subcategory of support processes. These authors present a process-centered SET, called *ODE* (Ontology-based software Development Environment), whose goal is to facilitate the partial automation of the software process. This environment is made up of several integrated tools, oriented towards the process definition, software projects monitoring and software quality control. A main element of the environment is an ontology, resulting from the combination of others created by the same authors, related to the

knowledge domains of software process, quality and software metrics [27]. The use of ODE to define processes in real-world projects assumes the instantiation of the elements previewed, including, for example: *Activity*, *Artifact*, or *Resource.*

In the scope of **support processes**, Deridder and Wouters [22] propose the use of ontologies to improve the creation, verification and validation of software artifacts created during the software development life cycle, through the integration of ontological engines into CASE tools. These authors classify the ontological engines into two kinds according to how they use the ontological data: (1) "*ontology-driven engines*" that retrieve data from the ontology within a given context, and use them to guide to software engineers in the performance of their tasks (for example, transforming ontological data into UML diagrams); and (2) "*ontology-based engines*" that utilize the ontology as a passive component, only needing to verify and look up data. The authors have created ontological engines of both kinds and have integrated them into the Rational Rose CASE tool.

The final work included in Table 2.4 refers to a proposal of Deridder et al. [23] for utilizing ontologies in the documentation process. It involves applying a structured approach to document a system by linking artifacts from the documentation and the implementation, using an ontology and obtaining what is referred to as "meta-documentation", which provides a coupling between the results of the analysis and design phases to the results of the implementation. The goal is to facilitate the software maintenance activities, avoiding wasted time in searching for "missing links" among artifacts it different levels of abstraction. For this, the ontology is a necessary element to establish the implicit links between related artifacts or between artifacts that represent the same concept in different languages.

2.5.2.2 Ontologies as Software Artifacts at Run Time

As Architectural Artifacts (Ontology-Driven Software)

In the proposals that were included in this taxonomy category (see Table 2.5), the software architecture is characterized by the use of one or more ontologies as central elements of the proposed system. The **knowledge-based system** (KBS) has an architecture that consists mainly of a knowledge repository that is formed by an ontology and an inference engine acting on this repository. There are numerous proposals of this type of system that could be referred to in this section. However, it is not necessary to describe all of them, as they share, in most cases, similar architecture, varying in each case just the application domain of the system. Therefore, we only refer to three proposals.

The first is that of Vieira and Casanova [95], who proposed the development of a **Workflow Management System** to integrate an ontology for representing the semantic relationships among elements such as *Workflow*, *Resource*, or *User*. This ontology indicates which resources and users are required to execute each workflow, and guides the discovery of possible alternatives when the execution of a workflow instance fails to proceed. This ontology is complemented by semantic rules dictating the way that alternatives can be found to allow workflow execution to continue.

Table 2.5. Proposals of ontologies as software artifacts at run time

Category	Proposal	Author(s) and reference
Architectural Artifacts (Ontology-driven software)	Flexible workflow execution through an ontology-based approach	Vieira and Casanova [95]
	An ontology-based context management and reasoning process for UbiComp applications	Chistopoulou et al. [18]
	Developing and managing software components in an ontology-based application server	Oberle et al. [70]
Information Resources (Ontology-aware software)	Swoogle: Semantic Web search	UMBC [88]
	Upgrade and publication of legacy data	Barrasa [6]
	Using ontologies as artifacts to enable databases interoperability	Brauner et al. [11]

Another example is the work of Cristopoulou et al. [18], who present an architecture for **ubiquitous computing applications**. These applications operate within an extremely dynamic and heterogeneous environment, and have to dynamically adapt to changes in their environment as a result of users' or other actors' activities. Therefore, context definition, representation, management and use are important factors affecting their operation. The authors propose the integration in the architecture of these context-aware systems, an ontology and an inference engine. The basic goal of the ontology is to support a context management process based on a set of rules which determine how a decision should be made and how it must be applied on existing knowledge represented by this ontology.

The third proposal of ontology-driven software included in Table 2.5 is that of Oberle et al. [70], who presented an ontology-based **application server**; this server, in addition to the habitual installed software components, includes an inference engine in which an ontology is loaded, with which an explicit and executable conceptual model for the administering the application server is represented. The server is implemented with J2EE technology, and the ontology conceptualizes key elements related to this technological platform, such as *Realm*, *User*, *Group* or *Roles*. It also includes concepts on

security mechanisms such as *Resource*, *Method*, *ResourceGroup*, *AccesRight*, *Invocation* or *RequestContext*; from these, the elements utilized are instantiated in order to control the server security. At run time, the server manages information in the form of semantic metadata (generated from configuration files), which are processed by the inference engine along with the content of the ontology.

The specification of the described systems and, in general, of any ontology-driven software, requires modeling techniques that can be used for the specification of the ontology integrated into the system, or for the inference engine, or for the rest of the system components. UML is an adequate notation for this purpose, having UML extension proposals such as that of Baclawski et al. [4], in order to model ontologies that can later be implemented into a language such as OWL. We must emphasize the Object Management Group (OMG) initiative to create a standard "Ontology Development Metamodel" (ODM) using the OMG's Meta Object Facility (MOF), to ease the development of ontologies with an engineering approach, more than adequate in the development of ontology-driven software. This initiative is comprehensively described in Chap. 8 of this book.

As Information Resources (Ontology-Aware Software)

Within this category are those proposals which deal with software systems that use one or more ontologies at run time in order to, for example, use their content in operations of information searching. Such is the case of **Web searchers** for the Semantic Web, such as Swoogle [88], which access over 10,000 ontologies to execute semantic searches.

Other applications framed inside this category are those that use ontologies as database substitutes, for information storage. Proposals exist to convert pre-existing databases into ontologies, assuming that the applications which previously accessed the original database now should access the ontology, constituting what has been named by us, following Guarino [43], ontology-aware software.

Among the proposals for transforming databases into ontologies is that of Barrasa [6], described in detail in Chap. 11 of this book, who has defined a language known as R_2O for mapping relational databases into ontologies, using a mapping processor called *ODEMapster*, both for generating the ontology (also called the "semantic repository of data") as well as for the execution of queries on the ontology. This facilitates the transformation of the applications that use a relational database to allow semantic access to the content available in the database.

Other work similar to that previously discussed is that of Brauner et al. [11], who have gone further, applying a mechanism of transformation to sev-

eral databases, in order to create an ontology-based catalogue which serves as a mediator to federated databases, and which offers centralized access to the data.

Although no proposal of this sort has been presented in Table 2.5, to conclude we will mention the applications that are being developed for the **Semantic Web**, considered in the category of "ontology-aware software", as the use of ontologies is and will be common place in the future development of these application types. And if we have "Service-oriented Architectures" (SOAs), with the use of Semantic Web services in the form of uncoupled, self-contained, self-described and semantically annotated software components, the ontologies will be used to describe not only the domain knowledge of these services, but also the interaction process of applications with these services, in such a way that eases the discovery, composition and execution of these services, thereby offering more complex functionality. For all this, it is fundamental that ontologies are used for Web services modeling, such as WSMO [78], not only by the creators of such services, in order to semantically annotate them, but also by the consumers, for discovery and use of the services.

References

1. Althoff, K.-D., Birk, A., Hartkopf, S., Mülle, W.: Managing Software Engineering Experience for Comprehensive Reuse. Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE), Kaiserslautern, Germany, 1999.
2. Assmann, U., Wagner, G.: Ontologies, metamodels and model-driven paradigm. In Ontologies for Software Engineering and Technology, Springer-Verlag, Berlin, chapter 9 (2006).
3. Association for Computing Machinery: Computing Curricula 2005 – The Overview Report. 30 September 2005. ACM, AIS, IEEE-CS. Available in: http://info.acm.org/education/curricula.html
4. Baclawski, K., Kokar, M.K., Kogut, P.A., Hart, L., Smith, J., Holmes, W.S., Letkowski, J., Aronson, M.L., Emery, P.: Extending the Unified Modeling Language for Ontology Development. International Journal of Software and Systems Modeling (SoSyM), 1(2): 142–156, 2002.
5. Baroni, A., Calero, C., Brito e Abreu. F. and Piattini, M. (2006) Object-Relational Database metrics formalization. Sixth International Conference on Quality Software (QSIC 2006). Beijig (China). To be published.
6. Barrasa, J.: Semantic Upgrade and Publication of Legacy Data. In Ontologies for Software Engineering and Technology, Springer-Verlag, Berlin, chapter 11 (2006).

7. Bertrand, T., Bézivin, J.: Ontological Support for Business Process Improvement. In D. Bustard, P. Kawalek, M.Norris (eds.), Systems Modeling for Business Process Improvement. Artech House Publishers, London, pp. 313–331 (2000).
8. Boehm, B., In, H.: Identifying Quality Requirements Conflicts. IEEE Software, March: 25–35, 1996.
9. Bose, P.: Conceptual design model based requirements analysis in the Win-Win framework for concurrrent requirements engineering. IEEE Workshop on Software Specification and Design (IWSSD), 1995.
10. Brandão, A.F., Torres, V., De Lucena, C.: Ontologies as Specifications for the Verification of Multi-Agent Systems Design. In Workshop on Ontologies as Software Engineering Artifacts (OOPSLA), Vancouver, Canada, 24–28 October 2004.
11. Brauner, D.F., Casanova, M.A., De Lucena, C.J.P.: Using ontologies as artifacts to enable databases interoperability. Workshop on Ontologies as Software Engineering Artifacts (OOPSLA), Vancouver, Canada, 24–28 October 2004.
12. Bures, T., Denney, E., Fischer, B., Nistor, E.C.: The role of ontologies in schema-based program synthesis. Workshop on Ontologies as Software Engineering Artifacts (OOPSLA), Vancouver, Canada, 24–28 October 2004.
13. Calero, C., Piattini, M.: An ontological approach to the SQL:2003. In Ontologies for Software Engineering and Technology, Springer-Verlag, Berlín, chapter 7 (2006).
14. Calero, C., Ruiz, F., Baroni, A.L., Brito e Abreu, F., Piattini, M.: An Ontological Approach to Describe the SQL:2003 Object-Relational Features. Computer Standards & Interfaces. Available online December 2, 2005 in: http://www.sciencedirect.com/science/journal/09205489
15. Chandrasekaran, B., Josephson, J.R., Benjamins, V.: Ontology of Tasks and Methods. In Proceedings of KAW'98, Banff, Alberta, Canada, 1998.
16. Chandrasekaran, B., Josephson, J.R., Benjamins, V.: What Are Ontologies, and Why Do We Need Them?. IEEE Intelligent Systems, 14 (1) 20–26, 1999.
17. Chen, H., Perich, F., Finin, T., Joshi, A.: SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. International Conference on Mobile and Ubiquitous Systems: Networking and Services, Boston, USAs, 22–25 August 2004, pp. 258–267.
18. Chistopoulou, E., Goumopoulos, C., Kameas, A.: An ontology-based context management and reasoning process for UbiComp applications. In Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: innovative context-aware services: usages and technologies, Grenoble, France, October 2005, pp. 265–270.
19. Cyc: OpenCyc.org: Formalized Common Knowledge. Cycorp, USA. Available in;
http://www.opencyc.org68
20. Denker, G.: Security Mechanisms Ontology. Computer Science Laboratory, SRI International, 2002. Available in:

http://www.csl.sri.com/~denker/owl-sec/security.owl
21. Deridder, D.: A Concept-Oriented Approach to Support Software Maintenance and Reuse Activities. 5th Joint Conference on Knowledge-Based Software Engineering (JCKBSE), Maribor, Slovenia, September 2002.
22. Deridder, D., Wouters, B.: The Use of Ontologies as a Backbone for Software Engineering Tools. Fourth Australian Knowledge Acquisition Workshop (AKAW), Sydney, Australia, December 1999.
23. Deridder, D., Wouters, B., Lybaert, W.: The use of an ontology to support a coupling between software models and implementation. International Workshop on Model Engineering, 14th European Conference on Object-Oriented Programming (ECOOP), Sophia Antipolis and Cannes, France, 2000.
24. Devedzíc, V.: Understanding Ontological Engineering. Communications of the ACM, 45(4): 136–144, 2002.
25. Dias, M.G., Anquetil, N., De Oliveira, K.M.: Organizing the Knowledge Used in Software Maintenance. Journal of Universal Computer Science, 9(7): 641–658, 2003.
26. Falbo, R.A., Cruz, A.C., Mian, P.G., Bertollo, G., Borges, F.: ODE: Ontology-based software Development Environment. IX Argentine Congress on Computer Science (CACIC), La Plata, Argentina, 6–7 October 2003.
27. Falbo, R.A., Guizzardi, G., Duarte, K.C.: An Ontological Approach to Domain Engineering. In Procedings of 14th International Conference on Software Engineering and Knowledge Engineering (SEKE), Ischia, Italy, July 1992, pp. 351–358.
28. Falbo, R., Menezes, C., Rocha, A.: Using Ontologies to Improve Knowledge Integration in Software Engineering Environments. 4th International Conference on Information Systems Analysis and Synthesis (ISAS), Orlando, USA, 1998.
29. Fernández, M., Gómez-Pérez, A., Juristo, N.: METHONTOLOGY: From Ontological Art Towards Ontological Engineering. AAAI Spring Symposium, University of Stanford, Palo Alto, California (USA), pp. 33–40, 1997.
30. Fensel, D: Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce. Second Edition, Springer-Verlag, Berlin, Heidelberg (2004).
31. Furtado, E., Vasco, J., Bezerra, W., Tavares, D., Da Silva, L., Limbourg, Q., Vander-Donckt, J.: An ontology based method for universal design of user interfaces. Workshop on Multiple User Interfaces over the Internet, British Human Computer Interaction Group Conference (HCI/IHM), 2001.
32. García, E.: A proposal of a knowledge model aimed at the use of questionnaires in the usability evaluation (in Spanish). PhD Thesis, University of Alcalá, Spain, 2004.
33. García, F., Bertoa, M.F., Calero, C., Vallecillo, A., Ruíz, F., Piattini, M., Genero, M.: Towards a consistent terminology for software measurement. Information and Software Technology. Available online August 22, 2005 in: http://www.sciencedirect.com/science/journal/09505849

34. García, F., Ruiz, F., Bertoa, M.F., Calero, C., Genero, M., Olsina, L.A., Martín, M.A., Quer, C., Condori, N., Abrahao, S., Vallecillo, A., Piattini, M.: Una Ontología de la Medición del Software (in Spanish). Technical Report DIAB-04-02-2, Dept. of Computer Science, University of Castilla-La Mancha. Available in: http://www.info-ab.uclm.es/trep.php
35. Girardi, R., Faria, C.: A Generic Ontology for the Specification of Domain Models. In Proceedings of 1st International Workshop on Component Engineering Methodology (WCEM'03) at Second International Conference on Generative Programming and Component Engineering, Erfurt, Germany, 2003.
36. Goldman, N.M.: Ontology-oriented programming: static typing for the inconsistent programmer. In International Semantic Web Conference (ISWC'2003), LNCS Vol. 2870, Sprinter-Verlag, Berlin, pp. 850–865 (2003).
37. Gómez-Pérez, A.: Knowledge sharing and reuse. In Jay Liebowitz (ed.), The Handbook of Applied Expert Systems. CRC Press, Boca Raton, Florida, 1998.
38. Gómez Pérez, A., Fernández López, M. Corcho, O.: Ontological Engineering. Springer-Verlag, London (2004).
39. González-Pérez, C., Henderson-Sellers, B.: An Ontology for Software Development Methodologies and Endeavours. In Ontologies for Software Engineering and Technology, Springer-Verlag, Berlin, chapter 4 (2006).
40. Gruber, T.R.: A translation approach to portable ontologies. Knowledge Acquisition, 5(2): 199–220, 1993.
41. Gruber, T.: Towards Principles for the Design of Ontologies used for Knowledge Sharing. International Journal of Human-Computer Studies, 43(5/6): 907–928, 1995.
42. Gruninger, M., Lee, J.: (2002): Ontology Applications and Design. Communications of the ACM, 45(2): 39–41, 2002.
43. Guarino, N.: Formal Ontology in Information Systems. In Proceedings of FOIS'98, Trento, Italy. IOS Press, Amsterdam (1998).
44. Guarino, N., Schneider, L.: Ontology-Driven Conceptual Modelling: Advanced Concepts. ER 2002. Pre-Conference Tutorials. Available in: http://www.loa-cnr.it/odcm.html
45. Guizzardi, G., Herre, H., Wagner, G.: On the General Ontological Foundations of Conceptual Modeling. 21st International Conference on Conceptual Modeling (ER), Tampere, Finland, October 2002.
46. Hamza, H.S.: Improving Analysis Patterns Reuse: An Ontological Approach. Workshop on Ontologies as Software Engineering Artifacts (OOPSLA), Vancouver, Canada, 24–28 October 2004.
47. Hilera, J.R., Sánchez-Alonso, S., García, E., Del Molino, C.J..: OntoGLOSE: A Light-weight Software Engineering Ontology. 1st Workshop on Ontology, Conceptualizations and Epistemology for Software and Systems Engineering (ONTOSE), Alcalá de Henares, Spain, 9–10 June 2005.

48. IEEE: IEEE Std 610.12-1990(R2002): IEEE Standard Glossary of Software Engineering Terminology (Reaffirmed 2002), IEEE, New York, USA.
49. IEEE: SWEBOK - Guide to the Software Engineering Body of Knowledge 2004 version. IEEE Computer Society. Available in: http://www.swebok.org
50. IEEE: Top-Level Categories for the ACM Taxonomy (extended version of the ACM Computing Classification System 2002). Available in: www.computer.org/mc/keywords/keywords.htm
51. ISO/IEC 9075-11:2003 Information technology – Database languages – SQL – Part 11: Information and Definition Schemas (SQL/Schemata). International Organization for Standardization, Genova.
52. ISO/IEC 12207:1995. Information Technology – Software Life Cycle Processes. ISO/IEC 1995.
53. ISO/IEC 15504-2:1998. Information Technology – Software Process Assessment – Part 2: A Reference Model for Processes and Process Capability. ISO/IEC 1998.
54. Jenz, D.E.: Simplifying the software development value chain through ontology-driven software artifact generation. Jenz and Partner GmbH Strategic White Paper, 2003. Available in: http://www.bpiresearch.com/WP_BPMOntology.pdf
55. Jurisica, I., Mylopoulos, J., Yu, E.: Using ontologies for knowledge management: an information systems perspective. In Proceedings of 62nd Annual Meeting of the American Society for Information Science (ASIS99), 1999, pp. 482–496.
56. Kitchenham, B.A., Travassos, G.H., Mayrhauser, A., Niessink, F., Schneidewind, N.F., Singer, J., Takada, S., Vehvilainen, R., Yang, H.: Towards an Ontology of Software Maintenance. Journal of Software Maintenance: Research and Practice, 11(6): 365–389, 1999.
57. Knublauch, H.: Ontology-driven software development in the context of the semantic web: an example scenario with protegé/OWL. First International Workshop on the Model-Driven Semantic Web (MDSW), 2004.
58. Larburu, I.U., Pikatza, J.M., Sobrado, F.J., García, J.J., López, D.: Hacia la implementación de una herramienta de soporte al proceso de desarrollo de software. Workshop in Artifificial Intelligence Applications to Engineering (AIAI), San Sebastián, Spain, 2003.
59. Lassila, O., McGuinness, D.: The Role of Frame-Based Representation on the Semantic Web. KSL Techical Report No. KSL-01-02, Jan-2001. Available in: http://www.ksl.stanford.edu/people/dlm/etai/lassila-mcguinness-fbr-sw. html
60. Lin, S., Liu, F., Loe, S.: Building A Knowledge Base of IEEE/EAI 12207 and CMMI with Ontology. Sixth International Protégé Workshop, Manchester, England, 7–9 July 2003.
61. Liu, F., Lo, S.: The Study on Ontology Integrating and Applying the Ontologies of IEEE/EIA 12207, CMMI, Workflow and J2EE to Web Service Development Environment. Sixth International Protégé Workshop, Manchester, England, 7-9 July 2003.

62. Martin, D. (ed.): OWL-based Web Service Ontology. OWL-S Coalition, 2004. Available in: http://www.daml.org/services/owl-s
63. Meersman, R.A.: The Use of Lexicons and Other Computer-Linguistic Tools in Semantics Design and Cooperation of Database Systems. In Y. Zhang (ed.), CODAS Conference Proceedings, Springer-Verlag, Berlin (2000).
64. Mendes, O., Abran, A.: Issues in the development of an ontology for an emerging engineering discipline. First Workshop on Ontology, Conceptualizations and Epistemology for Software and Systems Engineering (ONTOSE), Alcalá de Henares, Spain, 9–10 June 2005.
65. Mian, P.G., Falbo, R.A.: Building ontologies in a domain oriented software engineering environment. IX Argentine Congress on Computer Science (CACIC), La Plata, Argentina 6–7 October 2003.
66. MIT: RDF Ontology Collection. Simile Project, Massachusetts Institute of Technology, USA, 2004. Available in: http://simile.mit.edu/repository/ontologies/java
67. Mylopoulos, J.: Ontologies. Visited on January 4, 2006 in: http://www.cs.toronto.edu/~jm/2507S/Notes04/Ontologies.pdf
68. Newell, A.: The Knowledge Level. Artificial Intelligence, 18: 87–127, 1982.
69. Nour, P., Holz, H., Maurer, F.: Ontology-based retrieval of software process experiences. ICSE Workshop on Software Engineering over the Internet, 2000.
70. Oberle, D., Eberhart, A., Staab, S., Volz, R.: Developing and Managing Software Components in an Ontology-based Application Server. 5th International Middleware Conference, Toronto, Canada, 18–22 October 2004.
71. Object Management Group: Meta Object Facility (MOF) Specification; version 1.4, April 2002.
72. Oliveira, K.M., Villela, K., Rocha, A.R., Horta, G.: Use of Ontologies in Software Development Environments. In Ontologies for Software Engineering and Technology, Springer-Verlag, Berlin, chapter 10 (2006).
73. Ostertag, E., Hendler, J., Prieto-Díaz, R., Braun, C.: Computing similarity in a reuse library system: an AI-based approach. ACM Transactions on Software Engineering and Methodology, 1(3): 205–228, 1992.
74. Pahl, C.: Ontology-based description and reasoning for component-based development on the Web. In Procedings of ESEC/FSE Workshop on Specification and Verification of Component-based Systems (SAVCBS'03), Helsinki, Finland, September 2003, pp. 84–87.
75. Pisanelli, D.M., Gangemi, A., Steve, G.: Ontologies and Information Systems: the Marriage of the Century?. In Proceedings of LYEE Workshop, Paris, 2002.
76. Polo, M., Piattini, M., Ruiz, F., Calero, C.: MANTEMA: A Software Maintenance Methodology based on the ISO/IEC 12207 Standard. 4th IEEE International Software Engineering Standards Symposium (ISESS), Curitiba, Brazil. IEEE Computer Society, pp. 76–81, 1999.

77. Pressman, R.S.: Software Engineering: A Practitioner's Approach. Sixth Edition. McGraw-Hill, New York, 2004.
78. Roman, D., Lausen, H., Keller, U. (eds.): Web Service Modeling Ontology (WSMO), SDK WSMO Working Group, 2005. Available in: http://www.wsmo.org
79. Ruiz, F., García, F., Piattini, M., Polo, M.: Environment for Managing Software Maintenance Projects. In "Advances in Software Maintenance Management: Technologies and Solutions". Idea Group Publication (USA), chapter X, pp. 255–290, 2002.
80. Ruiz, F., Vizcaíno, A., Piattini, M., García, F.: An Ontology for the Management of Software Maintenance Projects. International Journal of Software Engineering and Knowledge Engineering, 14(3): 323–349, 2004.
81. Sánchez, D.M., Cavero, J.M., Marcos, E.: An ontology about ontologies and models: a conceptual discussion. First Workshop on Ontology, Conceptualizations and Epistemology for Software and Systems Engineering (ONTOSE), Alcalá de Henares, Spain, 9–10 June 2005.
82. Schleicher, A., Westfechtel, B.: Beyond Stereotyping: Metamodeling Approaches for the UML. 34th Hawaii International Conference on System Sciences (HICSS), Maui, Hawaii (USA), January 2001.
83. Schmidt, D.C.: Model-Driven Engineering. IEEE Computer, special issue on model-driven software development, 39(2): 25–31, February 2006.
84. SEI: Capability Maturity Model Integration (CMMI), Software Engineering Institute, 2002. Available in: http://www.sei.cmu.edu/cmmi/
85. Sicilia, M.A.., Cuadrado, J.J., García, E., Rodríguez, D., Hilera, J.R..: The evaluation of ontological representation of the SWEBOK as a revision tool. In 29th Annual International Computer Software and Application Conference (COMPSAC), Edinburgh, UK, 26–28 July 2005.
86. Spyns, P., Meersman, R., Jarrar, M.: Data Modelling versus Ontology Engineering. SIGMOD Record 31(4): 12–7, 2002.
87. Stumme, G., Maedche, A.: FCA-MERGE: Bottom-Up Merging of Ontologies. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI), Seattle, Washington, 2001.
88. Swoogle: Semantic Web Search. University of Maryland Baltimore County (UMBC), 2006. Available in: http://swoogle.umbc.edu
89. Tansalarak, N., Claypool, K.T.: XCM: A Component Ontology. Workshop on Ontologies as Software Engineering Artifacts (OOPSLA), Vancouver, Canada, 24–28 October 2004.
90. Tautz, C., Von Wangenheim, C.: REFSENO: A Representation Formalism for Software Engineering Ontologies. Fraunhofer IESE-Report No. 015.98/E, version 1.1, October 20, 1998.
91. Uschold, M., Gruninger, M.: Ontologies: Principles, Methods and Applications. Knowledge Engineering Review, 11(2): 93–15, 1996.

92. Uschold, M., Jasper, R.: A Framework for Understanding and Classifying Ontology Applications. In Proceedings of IJCAI Workshop on Ontologies and Problem-Solving Methods, August 1999. Visited on January 8, 2006 in: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-18/
93. Uschold, M., King, M.: Towards a Methodology for Building Ontologies. In Proceedings of the Workshop on Basic Issues in Knowledge Sharing (hosts within IJCAI), 1995.
94. Van Heijst, G., Schereiber, A.T., Wielinga, B.J.: Using Explicit Ontologies in KBS Development. International Journal of Human and Computer Studies, 46(2/3), 1997, pp. 293–310.
95. Vieira, T.A., Casanova, M.A.: Flexible Workflow Execution through an Ontology-based Approach. Workshop on Ontologies as Software Engineering Artifacts (OOPSLA), Vancouver, Canada, 24–28 October 2004.
96. Vizcaíno, A., Anquetil, N., Oliveira, K., Ruiz, F., Piattini, M.: Merging Software Maintenance Ontologies: Our Experience. First Workshop on Ontology, Conceptualizations and Epistemology for Software and Systems Engineering (ONTOSE), Alcala de Henares, Spain, 9–10 June 2005.
97. Wang, X., Chan, C.W.: Ontology Modeling Using UML. 7th International Conference on Object Oriented Information Systems (OOIS), Calgary, Canada, pp. 59–68, 2001.
98. Wang, X., Chan, C., Hamilton, H.: Design of knowledge-based systems with the ontology-domain-system approach. In Proceedings of SEKE 2002, pp. 233–236.
99. World Wide Web Consortium (W3C): Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering. Draft 2006/02/11. Available in: http://www.w3.org/2001/sw/BestPractices/SE/ODA/
100. Wouters, B., Deridder, D., Van Paesschen, E.: The use of ontologies as a backbone for use case management. European Conference on Object-Oriented Programming (ECOOP), 2000.
101. Zimmer, C., Rauschmayer, A.: Tuna: Ontology-Based Source Code Navigation and Annotation. Workshop on Ontologies as Software Engineering Artifacts (OOPSLA), Vancouver, Canada, 24–28 October 2004.