# An Evolutive Process to Convert Glossaries into Ontologies

José R. Hilera, Carmen Pagés, J. Javier Martínez, J. Antonio Gutiérrez, and Luis de-Marcos

*This paper describes a method to generate ontologies from glossaries of terms. The proposed method presupposes an evolutionary life cycle based on successive transformations of the original glossary that lead to products of intermediate knowledge representation (dictionary, taxonomy, and thesaurus). These products are characterized by an increase in semantic expressiveness in comparison to the product obtained in the previous transformation, with the ontology as the end product. Although this method has been applied to produce an ontology from the "IEEE Standard Glossary of Software Engineering Terminology," it could be applied to any glossary of any knowledge domain to generate an ontology that may be used to index or search for information resources and documents stored in libraries or on the Semantic Web.*

From the point of view of their expressiveness or semantic richness, knowledge representation tools can be classified at four levels: at the basic level (level 0), to which *dictionaries* belong, tools include definitions of concepts without formal semantic primitives; at the *taxonomies* level (level 1), tools include a vocabulary, implicit or explicit, as well as descriptions of specialized relationships between concepts; at the *thesauri* level (level 2), tools further include lexical (synonymy, hyperonymy, etc.) and equivalence relationships; and at the *reference models* level (level 3), tools combine the previous relationships with other more complex relationships between concepts to completely represent a certain knowledge domain.[1] *Ontologies* belong at this last level.

According to the hierarchic classification above, knowledge representation tools of a particular level add semantic expressiveness to those in the lowest levels in such a way that a dictionary or glossary of terms might develop into a taxonomy or a thesaurus, and later into an ontology. There are a variety of comparative studies of these tools,[2] as well as varying proposals for systematically generating ontologies from lower-level knowledge representation systems, especially from descriptor thesauri.[3]

This paper proposes a process for generating a *terminological ontology* from a dictionary of a specific knowledge domain.[4] Given the definition offered by Neches et al. ("an ontology is an instrument that defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary")[5] it is evident that the ontology creation process will be easier if there is a vocabulary to be extended than if it is developed from scratch.

If the developed ontology is based exclusively on the dictionary, the outcome will be limited by the richness of the definition of terms included in that dictionary. It would be what is normally called a "lightweight" ontology,[6] which could later be converted into a "heavyweight" ontology by implementing, in the form of axioms, knowledge not contained in the dictionary. This paper describes the process of creating a lightweight ontology of the domain of software engineering, starting from the *IEEE Standard Glossary of Software Engineering Terminology*.[7]

## Ontologies, the Semantic Web, and Libraries

Within the field of librarianship, ontologies are already being used as alternative tools to traditional controlled vocabularies. This may be observed particularly within the realm of digital libraries, although, as Krause asserts, objections to their use have often been raised by the digital library community.[8] One of the core objections is the difficulty of creating ontologies as compared to other vocabularies such as taxonomies or thesauri. Nonetheless, the semantic richness of an ontology offers a wide range of possibilities concerning indexing and searching of library documents.

The term *ontology* (used in philosophy to refer to the "theory about existence") has been adopted by the artificial intelligence research community to define a categorization of a knowledge domain in a shared and agreed form, based on concepts and relationships, which may be formally represented in a computer readable and usable format. The term has been widely employed since 2001, when Berners-Lee et al. envisaged the Semantic Web, which aims to turn the information stored on the Web into knowledge by transforming data stored in every webpage into a common scheme accepted in a specific domain.[9] To accomplish that task, knowledge must be represented in an agreed-upon and reusable computer-readable format. To do this, machines will require access to structured collections of information and to formalisms which are based on mathematical logic that permits higher levels of automatic processing.

Technologies for the Semantic Web have been developed by the World Wide Web Consortium (W3C). The most relevant technologies are RDF (Resource Description

**José R. Hilera** (jose.hilera@uah.es) is Professor, **Carmen Pagés** (carmina.pages@uah.es) is Assistant Professor, **J. Javier Martínez** (josej.martinez@uah.es) is Professor, **J. Antonio Gutiérrez** (jantonio.gutierrez@uah.es) is Assistant Professor, and **Luis de-Marcos** (luis.demarcos@uah.es) is Professor, Department of Computer Science, Faculty of Librarianship and Documentation, University of Alcalá, Madrid, Spain.

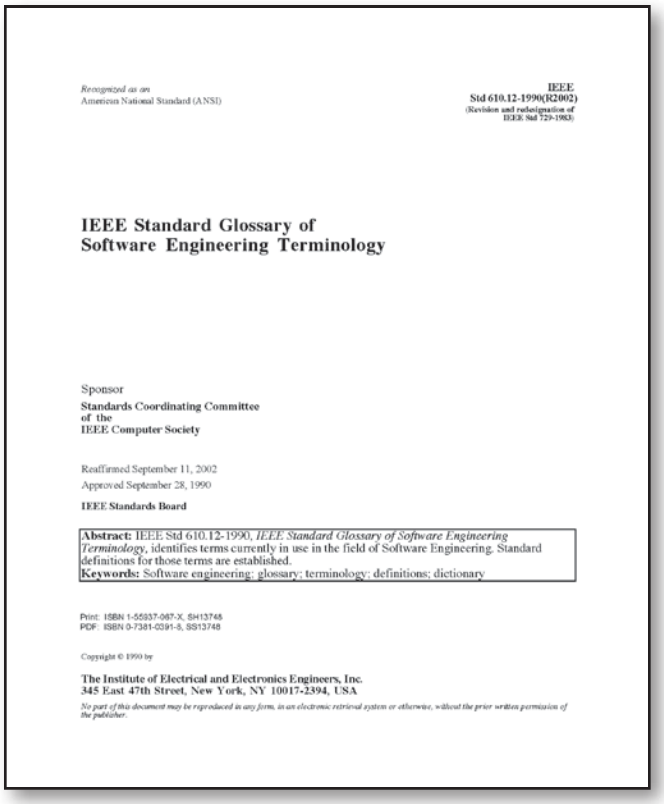
Framework),[10] which defines a common data model to specify metadata, and OWL (Ontology Web Language),[11] which is a new markup language for publishing and sharing data using Web ontologies. More recently, the W3C has presented a proposal for a new RDF-based markup system that will be especially useful in the context of libraries. It is called SKOS (Simple Knowledge Organization System), and it provides a model for expressing the basic structure and content of concept schemes, such as thesauri, classification schemes, subject heading lists, taxonomies, folksonomies, and other similar types of controlled vocabularies.[12]

The emergence of the Semantic Web has created great interest within librarianship because of the new possibilities it offers in the areas of publication of bibliographical data and development of better indexes and better displays than those that we have now in ILS OPACs.[13] For that reason, it is important to strive for semantic interoperability between the different vocabularies that may be used in libraries' indexing and search systems, and to have compatible vocabularies (dictionaries, taxonomies, thesauri, ontologies, etc.) based on a shared standard like RDF.

There are, at the present time, several proposals for using knowledge organization systems as alternatives to controlled vocabularies. For example, folksonomies, though originating within the Web context, have been proposed by different authors for use within libraries "as a powerful, flexible tool for increasing the user-friendliness and interactivity of public library catalogs."[14] Authors argue that the best approach would be to create interoperable controlled vocabularies using shared and agreed-upon glossaries and dictionaries from different domains as a departure point, and then to complete evolutive processes aimed at semantic extension to create ontologies, which could then be combined with other ontologies used in information systems running in both conventional and digital libraries for indexing as well as for supporting document searches. There are examples of glossaries that have been transformed into ontologies, such as the Cambridge Healthtech Institute's "Pharmaceutical Ontologies Glossary and Taxonomy" (http://www.genomicglossaries.com/content/ontologies.asp), which is an "evolving terminology for emerging technologies."

## IEEE Standard Glossary of Software Engineering Terminology

To demonstrate our proposed method, we will use a real glossary belonging to the computer science field, although it is possible to use any other. The glossary, available in electronic format (PDF), defines approximately 1,300 terms in the domain of software engineering (figure 1). Topics include addressing assembling, compiling, linking, loading; computer performance evaluation; configuration management; data types; errors, faults, and failures; evaluation techniques; instruction types; language types; libraries; microprogramming; operating systems; quality attributes; software documentation; software and system testing; software architecture; software development process; software development techniques; and software tools.[15]


*Recognized as an*
American National Standard (ANSI)

IEEE
Std 610.12-1990(R2002)
(Revision and redesignation of
IEEE Std 729-1983)

**IEEE Standard Glossary of Software Engineering Terminology**

Sponsor
**Standards Coordinating Committee of the IEEE Computer Society**

Reaffirmed September 11, 2002
Approved September 28, 1990
**IEEE Standards Board**

**Abstract:** IEEE Std 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology*, identifies terms currently in use in the field of Software Engineering. Standard definitions for those terms are established.
**Keywords:** Software engineering; glossary; terminology; definitions; dictionary

Print: ISBN 1-55937-067-X, SH13748
PDF: ISBN 0-7381-0391-8, SS13748

Copyright © 1990 by

**The Institute of Electrical and Electronics Engineers, Inc.**
**345 East 47th Street, New York, NY 10017-2394, USA**

*No part of this document may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

**Figure 1.** Cover of the Glossary document

In the glossary, entries are arranged alphabetically. An entry may consist of a single word, such as "software," a phrase, such as "test case," or an acronym, such as "CM." If a term has more than one definition, the definitions are numbered. In most cases, noun definitions are given first, followed by verb and adjective definitions as applicable. Examples, notes, and illustrations have been added to clarify selected definitions.

Cross-references are used to show a term's relations with other terms in the dictionary: "contrast with" refers to a term with an opposite or substantially different meaning; "syn" refers to a synonymous term; "see also" refers to a related term; and "see" refers to a preferred term or to a term where the desired definition can be found.

Figure 2 shows an example of one of the definitions of the glossary terms. Note that definitions can also include

**high order language (HOL).** A programming language that requires little knowledge of the computer on which a program will run, can be translated into several difference machine languages, allows symbolic naming of operations and addresses, provides features designed to facilitate expression of data structures and program logic, and usually results in several machine instructions for each program statement. Examples include Ada, COBOL, FORTRAN, ALGOL, PASCAL. *Syn:* **high level language; higher order language; third generation language.** *Contrast with:* **assembly language; fifth generation language; fourth generation language; machine language.** *Note:* Specific languages are defined in P610.13

**Figure 2.** Example of term definition in the IEEE Glossary

examples associated with the described concept. In the resulting ontology, the examples were included as instances of the corresponding class. In figure 2, it can be seen that the definition refers to another glossary on programming languages (Std 610.13), which is a part of the series of dictionaries related to computer science ("IEEE Std 610," figure 3). Other glossaries which are mentioned in relation to some references about term definitions are 610.1, 610.5, 610.7, 610.8, and 610.9.

To avoid redundant definitions and possible inconsistencies, links must be implemented between ontologies developed from those glossaries that include common concepts. The ontology generation process presented in this paper is meant to allow for integration with other ontologies that will be developed in the future from the other glossaries.

610—Standard Dictionary of Computer Terminology
- 610.1—Standard Glossary of Mathematics of Computing Terminology
- 610.2—Standard Glossary of Computer Applications Terminology
- 610.3—Standard Glossary of Modeling and Simulation Terminology
- 610.4—Standard Glossary of Image Processing Terminology
- 610.5—Standard Glossary of Data Management Terminology
- 610.6—Standard Glossary of Computer Graphics Terminology
- 610.7—Standard Glossary of Computer Networking Terminology
- 610.8—Standard Glossary of Artificial Intelligence Terminology
- 610.9—Standard Glossary of Computer Security and Privacy Terminology
- 610.10—Standard Glossary of Computer Hardware Terminology
- 610.11—Standard Glossary of Theory of Computation Terminology
- 610.12—Standard Glossary of Software Engineering Terminology
- 610.13—Standard Glossary of Computer Languages Terminology

**Figure 3.** IEEE Computer Science Glossaries

In addition to the explicit references to other terms within the glossary and to terms from other glossaries, the textual definition of a concept also has implicit references to other terms. For example, from the phrase "provides features designed to facilitate expression of data structures" included in the definition of the term *high order language* (figure 2), it is possible to determine that there is an implicit relationship between this term and the term *data structure*, also included in the glossary. These relationships have been considered in establishing the properties of the concepts in the developed ontology.

## Ontology Development Process

Many ontology development methods presuppose a life cycle and suggest technologies to apply during the process of developing an ontology.[16] The method described by Noy and McGuinness is helpful when beginning this process for the first time.[17] They establish a seven-step process:

1. Determine the domain and scope of the ontology
2. Consider reusing existing ontologies
3. Enumerate important terms in the ontology
4. Define the classes and the class hierarchy
5. Define the properties of classes (slots)
6. Define the facets of the slots
7. Create instances

As outlined in the Introduction, the ontology developed using our method is a terminological one. Therefore we can ignore the first two steps in Noy's and McGuinness' process as the concepts of the ontology coincide with the terms of the glossary used.

Any ontology development process must take into account the basic stages of the life cycle, but the way of organizing the stages can be different in different methods. In our case, since the ontology has a terminological character, we have established an incremental development process that supposes the natural evolution of the glossary from its original format (dictionary or vocabulary format) into an ontology. The proposed life cycle establishes a series of steps or phases that will result in intermediate knowledge representation tools, with the final product, the ontology, being the most semantically rich (figure 4).

Therefore this is a product-driven process, in which the aim of every step is to obtain an intermediate product useful on its own. The intermediate products and the final

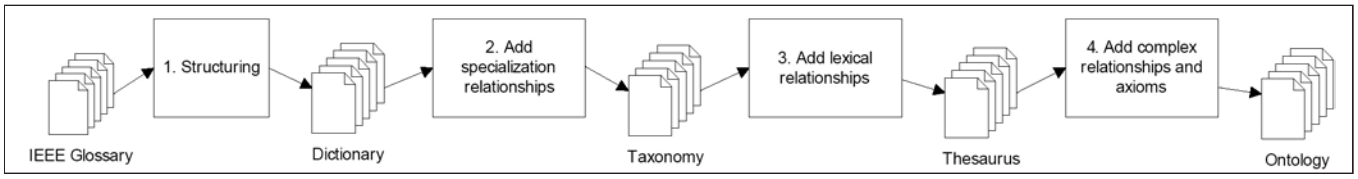

IEEE Glossary
1. Structuring
Dictionary
2. Add specialization relationships
Taxonomy
3. Add lexical relationships
Thesaurus
4. Add complex relationships and axioms
Ontology

**Figure 4.** Ontology development process

product are a dictionary, which has a formal and computer processed structure, with the terms and their definitions in XML format; a taxonomy, which reflects the hierarchic relationships between the terms; a thesaurus, which includes other relationships between the terms (for example, the synonymy relationship); and, finally, the ontology, which will include the hierarchy, the basic relationships of the thesaurus, new and more complex semantic relationships, and restrictions in form of axioms expressed using description logics.[18] The following paragraphs describe the way each of these products is obtained.

## Dictionary

The first step of the proposed development process consists of the creation of a dictionary in XML format with all the terms included in the *IEEE Standard Glossary of Software Engineering Terminology* and their related definitions. This activity is particularly mechanical and does not need human intervention as it is basically a transformation of the glossary from its original format (PDF) into a format better suited to the development process.

All formats considered for the dictionary are based on XML, and specifically on RDF and RDF schema. In the end, we decided to work with the standards DAML+OIL and OWL,[19] though we are not opposed to working with other languages, such as SKOS or XMI,[20] in the future. (In the latter case, it would be possible to model the intermediate products and the ontology in UML graphic models stored in xml files.)[21] In our project, the design and implementation of all products has been made using an ontology editor. We have used OilEd (with OilViz Plugin) as editor, both because of its simplicity and because it allows the exportation to OWL and DAML formats. However, with future maintenance and testing in mind, we decided to use Protégé (with OWL plugin) in the last step of the process, because this is a more flexible environment with extensible modules that integrate more functionality such as ontology annotation, evaluation, middleware service, query and inference, etc.

Figure 5 shows the dictionary entry for "high order language," which appears in figure 2. Note that the dictionary includes only owl:class (or daml:class) to mark the term; rdf:label to indicate the term name; and rdf:comment to provide the definition included in the original glossary.

```
<owl:Class rdf:about="#HighOrderLanguage">

 <rdfs:label>HighOrderLanguage</rdfs:label>

 <rdfs:comment><![CDATA[A programming language
                        that requires little knowledge of the
                        computer on which a program will
                        run, can be translated into several
                        different machine languages, allows
                        symbolic naming of operations
                        and addresses, provides features
                        designed to facilitate expression of
                        data structures and program logic,
                        and usually results in several machine
                        instructions for each program
                        statement.]]>

 </rdfs:comment>

</owl:Class>
```

**Figure 5.** Example of dictionary entry

Since there are terms with different meanings (up to five in some cases) in the *IEEE Glossary of Software Engineering Terminology*, during dictionary development we decided to create different concepts (classes) for the same term, associating a number to these concepts to differentiate them. For example, there are five different definitions for the term *test*, which is why there are five concepts (*Test1–Test5),* corresponding to the five meanings of the term: (1) An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component; (2) To conduct an activity as in (1); (3) A set of one or more test cases; (4) A set of one or more test procedures; (5) A set of one or more test cases and procedures.

## Taxonomy

The proposed lifecycle establishes a stage for the conversion of a dictionary into a taxonomy, understanding taxonomy as an instrument of concepts categorization, that is, as a systematical classification in a traditional way. As Gilchrist states, there is no consensus on the meaning of terms like *taxonomy*, *thesaurus*, or *ontology*.[22] In addition, much work in the field of ontologies has been done without taking advantage of similar work performed in the fields of linguistics and library science.[23] This situation is changing because of the increasing publication of works that relate the development of ontologies to the development of "classic" terminological tools (vocabularies, taxonomies, and thesauri).

This paper emphasizes the importance and usefulness of the intermediate products created at each stage of the evolutive process from glossary to ontology. The end product of the initial stage is a dictionary expressed as XML. The next stage in the evolutive process (figure 4) is the transformation of that dictionary into a taxonomy through the addition of hierarchical relationships between concepts.

To do this, it is necessary to undertake a lexical-semantic analysis of the original glossary. This can be done in a semiautomatic way by applying natural language processing (NLP) techniques, such as those recommended by Morales-del-Castillo et al.,[24] for creating thesauri. The basic processing sequence in linguistic engineering comprises the following steps: (1) incorporate the original documents (in our case the dictionary obtained in the previous stage) into the information system; (2) identify the language in which they are written, distinguishing independent words; (3) "understand" the processed material at the appropriate level; (4) use this understanding to transform, search, or traduce data; (5) produce the new media required to present the produced outcomes; and finally, (6) present the final outcome to human users by means of the most appropriate peripheral device—screen, speakers, printer, etc.

An important aspect of this process is natural language comprehension. For that reason, several different kinds of programs are employed, including lemmatizers (which implement stemming algorithms to extract the lexeme or root of a word), morphologic analyzers (which glean sentence information from their constituent elements: morphemes, words, and parts of speech), syntactic analyzers (which group sentence constituents to extract elements larger than words), and semantic models (which represent language semantics in terms of concepts and their relations, using abstraction, logical reasoning, organization and data structuring capabilities).

From the information in the software engineering dictionary and from a lexical analysis of it, it is possible to determine a hierarchical relationship when the name of a term contains the name of another one (for example, the term *language* and the terms *programming language* and *hardware design language*), or when expressions such as "is a" linked to the name of another term included in the glossary appear in the text of the term definition. (For example, when analyzing the definition of the term compiler: "(Is) A computer program that translates programs expressed in a high order language into their machine language equivalent," it is possible to deduce that compiler is a subconcept of computer program, which is also included in the glossary.) In addition to the lexical or syntactic analysis, it is necessary for an expert in the domain to perform a semantic analysis to complete the development of the taxonomy.

The implementation of the hierarchical relationships among the concepts is made using rdfs:subClassOf, regardless of whether the taxonomy is implemented in OWL or DAML format, since both languages specify this type of relationship in the same way. Figure 6 shows an example of a hierarchical relationship included in the definition of the concept pictured in figure 5.

## Thesaurus

According to the International Organization for Standardization (ISO), a thesaurus is "the vocabulary of a controlled indexing language, formally organized in order to make explicit the a priori relations between concepts (for example 'broader' and 'narrower')."[25] This definition establishes the lexical units and the semantic relationships between these units as the elements that constitute a thesaurus. The following is a sample of the lexical units:

- *Descriptors* (also called "preferred terms"): the terms used consistently when indexing to represent a concept that can be in documents or in queries to these documents. The ISO standard introduces the option of adding a definition or an application note to every term to establish explicitly the chosen meaning. This note is identified by the abbreviation SN (Scope Note), as shown in figure 7.
- *Non-descriptors* ("non-preferred terms"): the synonyms or quasi-synonyms of a preferred term. A nonpreferred term is not assigned to documents submitted to an indexing process, but is provided as an entry point in a thesaurus to point to the appropriate descriptor. Usually the descriptors are written in capital letters and the nondescriptors in small letters.
- *Compound descriptors*: the terms used to represent complex concepts and groups of descriptors, which allow for the structuring of large numbers of thesaurus descriptors into subsets called micro-thesauri.

In addition to lexical units, other fundamental elements of a thesaurus are semantic relationships between these units. The more common relationships between lexical units are the following:

- *Equivalence*: the relationship between the descriptors and the nondescriptors (synonymous and

```
<owl:Class rdf:about="#HighOrderLanguage">
 ...
 <rdfs:subClassOf>
 <owl:Class rdf:about="#ProgrammingLanguage"/>
 </rdfs:subClassOf>
</owl:Class>
```

**Figure 6.** Example of taxonomy entry

quasi-synonymous). ISO establishes that the abbreviation UF (Used For) precedes the nondescriptors linked to a descriptor; and the abbreviation USE is used in the opposite case. For example, a thesaurus developed from the IEEE glossary might include a descriptor "high order language" and an equivalence relationship with a nondescriptor "high level language" (figure 7).

- *Hierarchical*: a relationship between two descriptors. In the thesaurus one of these descriptors has been defined as superior to the other one. There are no hierarchical relationships between nondescriptors, nor between nondescriptors and descriptors. A descriptor can have no lower descriptors or several of them, and no higher descriptors or several of them. According to the ISO standard, hierarchy is expressed by means of the abbreviations BT (Broader Term), to indicate the generic or higher descriptors, and NT (Narrower Term), to indicate the specific or lower descriptors. The term at the head of the hierarchy to which a term belongs can be included, using the abbreviation TT (Top Term). Figure 7 presents these hierarchical relationships.
- *Associative*: a reciprocal relationship that is established between terms that are neither equivalent nor hierarchical, but are semantically or conceptually associated to such an extent that the link between them should be made explicit in the controlled vocabulary on the grounds that it may suggest additional terms for use in indexing or retrieval. It is generally indicated by the abbreviation RT (Related Term). There are no associative relationships between nondescriptors and descriptors, or between descriptors already linked by a hierarchical relation. It is possible to establish associative relationships between descriptors belonging to the same or different category. The associative relationships can be of very different types. For example, they can represent causality, instrumentation, location, similarity, origin, action, etc. Figure 7 shows two associative relations, indicating that high order language relates to both assembly and machine languages.

```
..
HIGH ORDER LANGUAGE (descriptor)
 SN A programming language that...
 UF High level language (no-descriptor)
 UF Third generation language (no-descriptor)
 TT LANGUAGE
 BT PROGRAMMING LANGUAGE
 NT OBJECT ORIENTED LANGUAGE
 NT DECLARATIVE LANGUAGE
 RT ASSEMBLY LANGUAGE (contrast with)
 RT MACHINE LANGUAGE (contrast with)
..
High level language
 USE HIGH ORDER LANGUAGE
..
Third generation language
 USE HIGH ORDER LANGUAGE
..
```

**Figure 7.** Fragment of a thesaurus entry

The life cycle proposed in this paper (figure 4) includes a third step or phase that transforms the taxonomy obtained in the previous phase into a thesaurus through the incorporation of relationships between the concepts that complement the hierarchical relations included in the taxonomy. Basically, we have to add two types of relationships—equivalence and associative, represented in the standard thesauri with UF (and USE) and RT respectively.

We will continue using XML to implement this new product. There are different ways of implementing a thesaurus using a language based on XML. For example, Matthews et al. proposed a standard RDF format,[26] where as Hall created an ontology in DAML.[27] In both cases, the authors modeled the general structure of a thesaurus from classes (rdf:Class or daml:class) and properties (rdf:Property or daml:ObjectProperty). In the first case they proposed five classes: ThesaurusObject, Concept, TopConcept, Term, ScopeNote; and several properties to implement the relations, like hasScopeNote (SN), IsIndicatedBy, PreferredTerm, UsedFor (UF), ConceptRelation, BroaderConcept (BT), NarrowerConcept (NT), TopOfHierarchy (TT) and isRelatedTo (RT).

Recently the W3C has developed the SKOS specification, created to define knowledge organization schemes. In the case of thesauri, SKOS includes specific tags, such as skos:Concept, skos:scopeNote (SN), skos:broader (BT), skos:narrower (NT), skos:related (RT), etc., that are equivalent to those listed in the previous paragraph. Our specification does not make any statement about the formal relationship between the class of SKOS concept schemes and the class of OWL ontologies, which will allow different design patterns to be explored for using SKOS in combination with OWL.

Although any of the above-mentioned formats could be used to implement the thesaurus, given that the end-product of our process is to be an ontology, our proposal is that the product to be generated during this phase should have a format compatible with the final ontology and with the previous taxonomy. Therefore a minimal number of changes will be carried out on the product created in the previous step, resulting in a knowledge representation tool similar to a thesaurus. That tool does not need to be modified during the following (final) phase of transformation into an ontology. Nevertheless, if for some reason it is necessary to have the thesaurus in one of the other formats (such as SKOS), it is possible to apply a simple XSLT transformation to the product. Another option would be to integrate a thesaurus ontology, such as the one proposed by Hall,[28] with the ontology representing the IEEE glossary.

In the thesaurus implementation carried out in our project, the following limitations have been considered:

- Only the hierarchical relationships implemented in the taxonomy have been considered. These include relationsips of type "is-a," that is, generalization relationships or type–subset relationships. Relationships that can be included in the thesaurus marked with TT, BT, and NT, like relations of type "part of" (that is, partative relationships) have not been considered. Instead of considering them as hierarchical relationships, the final ontology includes the possibility of describing classes as a union of classes.
- The relationships of synonymy (UF and USE) used to model the cross-references in the IEEE glossary ("Syn" and "See," respectively) were implemented as equivalent terms, that is, as equivalent axioms between classes (owl:equivalentClass or daml:sameClassAs), with inverse properties to reflect the preference of the terms. For example:

```
<owl:ObjectProperty rdf:about="#UF">
<owl:inverseOf rdf:resource="#USE"/>.
```

  Or using the glossary notation:

```
<owl:ObjectProperty rdf:about="#Syn">
<owl:inverseOf rdf:resource="#See"/>.
```

- The rest of the associative relationships (RT) that were included in the thesaurus correspond to the cross-references of the type "Contrast with" and "See also" that appear explicitly in the IEEE glossary.
- Neither compound descriptors nor groups of descriptors have been implemented because there is no such structure in the glossary.

## Ontology

Ding and Foo state that "ontology promotes standardization and reusability of information representation through identifying common and shared knowledge. Ontology adds values to traditional thesauri through deeper semantics in digital objects, both conceptually, relationally and machine understandably."[29] This semantic richness may imply deeper hierarchical levels, richer relationships between concepts, the definition of axioms or inference rules, etc.

The final stage of the evolutive process is the transformation of the thesaurus created in the previous stage into an ontology. This is achieved through the addition of one or more of the basic elements of semantic complexity that differentiates ontologies from other knowledge representation standards (such as dictionaries, taxonomies, and thesauri). For example:

- Semantic relationships between the concepts (classes) of the thesaurus have been added as properties or ontology slots.
- Axioms of classes and axioms of properties. These are restriction rules that are declared to be satisfied by elements of ontology. For example, to establish disjunctive classes (<owl:Class rdf:about ="*HigOrderLanguage"> <owl:disjointWith> <owl:Class rdf:about="* MachineLanguage"/>), have been defined, and quantification restrictions (existential or universal) and cardinality restrictions in the relationships have been implemented as properties.

Software based on techniques of linguistic analysis has been developed to facilitate the establishment of the properties and restrictions. This software analyzes the definition text for each of the more than 1,500 glossary terms (in thesaurus format), isolating those words that match the name of other glossary terms (or a word in the definition text of other glossary terms). The isolated words will then be candidates for a relationship between both of them. (Figure 8 shows the candidate properties obtained from the Software Engineering glossary.) The user then has the option of creating relationships with the identified candidate words. The user must indicate, for every relationship to be created, the restriction type that it represents as well as existential or universal quantification or cardinality (minimum or maximum). After confirming this information, the program updates the file containing the ontology (OWL or DAML), adding the property to the class that represents the processed term.

Figure 9 shows an example of the definition of two properties and its application to the class HighOrderLanguage: a property *Express* with existential quantification over the class *DataStructure* to indicate that a language must represent at least one data structure; and a property *TranslateTo* of universal type to indicate that any high-level language is translated into machine language (*MachineLanguage*).

## Results, Conclusions, and Future Work

The existence of ontologies of specific knowledge domains (software engineering in this case) facilitates the process of finding resources about this discipline on the Semantic Web and in digital libraries, as well as the reuse of learning objects of the same domain stored in repositories available on the Web.[30] When a new resource is indexed in a library catalog, a new record that conforms to the ontology conceptual data model may be included. It will be necessary to assign its properties according to the concept definition included in the ontology. The user may later execute semantic queries that will be run by the search system that will traverse the ontology to identify the concept in which the user was interested to launch a wider query including the resources indexed under the concept. Ontologies, like the one that has been "evolved," may also be used in an open way to index and search for resources on the Web. In that case, however, semantic search engines such as Swoogle (http://swoogle.umbc.edu/), are required in place of traditional syntactic search engines, such as Google.

The creation of a complete ontology of a knowledge domain is a complex task. In the case of the domain presented in this paper, that of software engineering, although there have been initiatives toward ontology creation that have yielded publications by renowned authors in the field,[31] a complete ontology has yet to be created and published.

This paper has described a process for developing a modest but complete ontology from a glossary of terminology, both in OWL format and DAML+OIL format,

| | | | |
|---|---|---|---|
| accept | direct | is composed | present for |
| access | disable | is contained | prevent |
| accomplish | disassembles | is contained in | preventaccessto |
| account | display | is establish | process |
| achieve | distribute | is established | produce |
| adapt | divide | is executed after | produce no |
| add | document | is executed by | propose |
| adjust | employ | is incorrect | provide |
| advance | enable | is independent of | rank |
| affect | encapsulate | is manifest | reads |
| aggregate | encounter | is measured in | realize |
| aid | ensure | is not | receive |
| allocate | enter | is not subdivided in | reconstruct |
| allow | establish | is part | records |
| allow symbolic | estimate | is part of | recovery |
| naming | establish | is performed by | refine |
| alter | evaluate | is performed on | reflect |
| analyze | examine | is portion | reformat |
| apply | exchange | is process by | relate |
| approach | execute after | is produce by | relation |
| approve | execute in | is produce in | release |
| arrangement | executes | is ratio | relocates |
| arrive | expand | is represented by | remove |
| assign | express | is the output | repair |
| assigned by | express as | is the result of | replace |
| assume | extract | is translated by | represent |
| avoid | facilitate | is type | request |
| await | fetch | is used | require |
| begin | fill | is used in | reserve |
| break | follow | isolate | reside |
| bring | fulfil | know | restore |
| broke down | generate | link | restructure |
| builds | give | list | result |
| call | give partial | load | resume |
| called by | given constrain | locate | retain |
| can be | govern | maintain | retest |
| can be input | have | make | returncontrolto |
| can be used as | have associated | make up | reviews |
| can operate in | have met | may be | satisfy |
| cannot be usedas | have no | measure | schedule |
| carry out | hold | meet | send |
| cause | identify | mix | server |
| change | identify request | modify | set |
| characterize | ignore | monitors | share |
| combine | implement | move | show |
| communicate | imply | no contain | shutdown |
| compare | improve | no execute | specify |
| comply | incapacitate | no relate | store |
| comprise | include | no use | store in |
| conduct | incorporate | not be | structure |
| conform | increase | connected | submission of |
| consist | indicate | not erase | supervise |
| constrain | inform | not fill | supports |
| construct | initiate | not have | suppress |
| contain | insert | not involve | suspend |
| contains no | install | not involving | swap |
| contribute | intend | not translate | synchronize |
| control | interact with | not use | take |
| convert | interprets | occur | terminate |
| copy | interrelate | occur in | test |
| correct | investigate | occur in a | there are no |
| correspond | invokes | operate | through |
| count | is | operatewith | throughout |
| create | is a defect in | optimize | transfer |
| debugs | is a form of | order | transform |
| decompiles | is a method of | output | translate |
| decomposedinto | is a mode of | parses | transmit |
| decrease | is a part | pas | treat |
| define | is a part of | pass test | through |
| degree | is a sequence | perform | understand |
| delineate | is a sequenceof | permit | update |
| denote | is a technique | permitexecute | use |
| depend | is a techniqueof | permit the | use in |
| depict | is a type | execution | use to |
| describe | is a type of | pertaining | utilize |
| design | is ability | place | value |
| designate | is activated by | preclude | verify |
| detect | is adjusted by | predict | work in |
| determine | is applied to | prepare | writes |
| develop | is based | prescribe | |
| development | is called by | present | |

**Figure 8.** Candidate properties obtained from the linguistic analysis of the Software Engineering glossary

```
<owl:Class rdf:about="#HighOrderLanguage">
 ...
 <rdfs:subClassOf>
 <owl:Restriction>
 <owl:onProperty rdf:resource="#Express"/>
 <owl:someValuesFrom>
 <owl:Class rdf:about="#DataStructure"/>
 </owl:someValuesFrom>
 </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
 <owl:Restriction>
 <owl:onProperty rdf:resource="#TranslateTo"/>
 <owl:allValuesFrom>
 <owl:Class rdf:about="#MachineLanguage"/>
 </owl:allValuesFrom>
 </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

**Figure 9.** Example of ontology entry

which is ready to use in the Semantic Web. As described at the opening of this article, our aim has been to create a lightweight ontology as a first version, which will later be improved by including more axioms and relationships that increase its semantic expressiveness. We have tried to make this first version as tailored as possible to the initial glossary, knowing that later versions will be improved by others who might take on the work. Such improvements will increase the ontology's utility, but will make it a less-faithful representation of the IEEE glossary from which it was derived.

The ontology we have developed includes 1,521 classes that correspond to the same number of concepts represented in the IEEE glossary. (Included in this number are the different meanings that the glossary assigns to each term.) We defined 324 properties or relationships between these classes. These are based on a semiautomated linguistic analysis of the glossary content (for example, *Allow*, *Convert*, *Execute*, *OperateWith*, *Produces*, *Translate*, *Transform*, *Utilize*, *WorkIn*, etc.), which will be refined in future versions.

The authors' aim is to use this ontology, which we have called OntoGLOSE (Ontology GLossary Software Engineering), to unify the vocabulary. OntoGLOSE will be used in a more ambitious project, whose purpose is the development of a complete ontology in software engineering from the SWEBOK Guide.[32]

Although this paper has focused on this ontology, the method that has been described may be used to generate an ontology from any dictionary. The flexibility that OWL permits for ontology description, along with its compatibility with other RDF-based metadata languages, makes possible interoperability between ontologies and between ontologies and other controlled vocabularies and allows for the building of merged representations of multiple knowledge domains. These representations may eventually be used in libraries and repositories to index and search for any kind of resource, not only those related to the original field.

## Acknowledgments

This research is co-funded by the Spanish Ministry of Industry, Tourism and Commerce PROFIT program (grant TSI-020100-2008-23). The authors also want to acknowledge support from the TIFyC research group at the University of Alcala.

## References and Notes

**1.** M. Dörr et al., *State of the Art in Content Standards* (Amsterdam: OntoWeb Consortium, 2001).

**2.** D. Soergel, "The Rise of Ontologies or the Reinvention of Classification," *Journal of the American Society for Information Science* 50, no. 12 (1999): 1119–20; A. Gilchrist, "Thesauri, Taxonomies and Ontologies—An Etymological Note," *Journal of Documentation* 59, no. 1 (2003): 7–18.

**3.** B. J. Wielinga et al., "From Thesaurus to Ontology," *Proceedings of the 1st International Conference on Knowledge Capture* (New York: ACM, 2001): 194–201: J. Qin and S. Paling, "Converting a Controlled Vocabulary into an Ontology: The Case of GEM," *Information Research* 6 (2001): 2.

**4.** According to Van Heijst, Schereiber, and Wielinga, ontologies can be classified as terminological ontologies, information ontologies, and knowledge modeling ontologies; terminological ontologies specify the terms that are used to represent knowledge in the domain of discourse, and they are in use principally to unify vocabulary in a certain domain. G. Van Heijst, A. T.

Schereiber, and B. J. Wielinga, "Using Explicit Ontologies in KBS Development," *International Journal of Human & Computer Studies* 46, no. 2/3 (1996): 183–292.

**5.** R. Neches et al., "Enabling Technology for Knowledge Sharing," *AI Magazine* 12, no. 3 (1991): 36–56.

**6.** O. Corcho, F. Fernández-López, and A. Gómez-Pérez, "Methodologies, Tools and Languages for Buildings Ontologies. Where Is Their Meeting Point?" *Data & Knowledge Engineering* 46, no. 1 (2003): 41–64.

**7.** Intitute of Electrical and Electronics Engineers (IEEE), *IEEE Std 610.12-1990(R2002): IEEE Standard Glossary of Software Engineering Terminology* (Reaffirmed 2002) (New York: IEEE, 2002).

**8.** J. Krause, "Semantic Heterogeneity: Comparing New Semantic Web Approaches with those of Digital Libraries," *Library Review* 57, no. 3 (2008): 235–48.

**9.** T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American* 284, no. 5 (2001): 34–43.

**10.** World Wide Web Consortium (W3C), *Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation 10 February 2004*, http://www.w3.org/TR/rdf-concepts/ (accessed Oct. 5, 2009).

**11.** World Wide Web Consortium (W3C), Web Ontology Language (OWL), 2004, http://www.w3.org/2004/OWL (accessed Oct. 5, 2009).

**12.** World Wide Web Consortium (W3C), SKOS Simple Knowledge Organization System, 2009, http://www.w3.org/TR/2009/REC-skos-reference-20090818/ (accessed Oct. 5, 2009).

**13.** M. M. Yee, "Can Bibliographic Data be Put Directly onto the Semantic Web?" *Information Technology & Libraries* 28, no. 2 (2009): 55-80.

**14.** L. F. Spiteri, "The Structure and Form of Folksonomy Tags: The Road to the Public Library Catalog," *Information Technology & Libraries* 26, no. 3 (2007): 13–25.

**15.** Corcho, Fernández-López, and Gómez-Pérez, "Methodologies, Tools and Languages for Buildings Ontologies."

**16.** IEEE, *IEEE Std 610.12-1990(R2002).*

**17.** N. F. Noy and D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," 2001, Stanford University, http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf (accessed Sept 10, 2010).

**18.** D. Baader et al., *The Description Logic Handbook* (Cambridge: Cambridge Univ. Pr., 2003).

**19.** World Wide Web Consortium, DAML+OIL Reference Description, 2001, http://www.w3.org/TR/daml+oil-reference (accessed Oct. 5, 2009); W3C, OWL.

**20.** W3C, SKOS; Object Management Group, *XML Metadata Interchange (XMI)*, 2003, http://www.omg.org/technology/documents/formal/xmi.htm (accessed Oct. 5, 2009).

**21.** UML (Unified Modeling Language) is a standardized general-purpose modeling language (http://www.uml.org). Nowadays, different UML plugins for ontologies' editors exist. These plugins allow working with UML graphic models. Also, it is possible to realize the UML models with a CASE tool, to export them to XML format, and to transform them to the ontology format (for example, OWL) using a XSLT sheet, as the one published in D. Gasevic, "UMLtoOWL: Converter from UML to OWL," http://www.sfu.ca/~dgasevic/projects/UMLtoOWL/ (accessed Oct. 5, 2009).

**22.** Gilchrist, "Thesauri, Taxonomies and Ontologies."

**23.** Soergel, "The Rise of Ontologies or the Reinvention of Classification."

**24.** J. M. Morales-del-Castillo et al., "A Semantic Model of Selective Dissemination of Information for Digital Libraries," Information Technology & Libraries 28, no. 1 (2009): 22–31.

**25.** International Standards Organization, ISO 2788:1986 Documentation—Guidelines for the Establishment and Development of Monolingual Thesauri (Geneve: International Standards Organization, 1986).

**26.** B. M. Matthews, K. Miller, and M. D. Wilson, "A Thesaurus Interchange Format in RDF," 2002, http://www.w3c.rl.ac.uk/SWAD/thes_links.htm (accessed Feb. 10, 2009).

**27.** M. Hall, "CALL Thesaurus Ontology in DAML," Dynamics Research Corporation, 2001, http://orlando.drc.com/daml/ontology/CALL-Thesaurus (accessed Oct. 5, 2009).

**28.** Ibid.

**29.** Y. Ding and S. Foo, "Ontology Research and Development. Part 1—A Review of Ontology Generation," *Journal of Information Science* 28, no. 2 (2002): 123–36. See also B. H. Kwasnik, "The Role of Classification in Knowledge Representation and Discover," *Library Trends* 48 (1999): 22–47.

**30.** S. Otón et al., "Service Oriented Architecture for the Implementation of Distributed Repositories of Learning Objects," *International Journal of Innovative Computing, Information & Control* (2010), forthcoming.

**31.** O. Mendes and A. Abran, "Software Engineering Ontology: A Development Methodology," *Metrics News* 9 (2004): 68–76; C. Calero, F. Ruiz, and M. Piattini, *Ontologies for Software Engineering and Software Technology* (Berlin: Springer, 2006).

**32.** IEEE, *Guide to the Software Engineering Body of Knowledge (SWEBOK)* (Los Alamitos, Calif.: IEEE Computer Society, 2004), http:// www.swebok.org (accessed Oct. 5, 2009).