Towards a Framework to Verify Knowledge Sharing Technology

ASUNCIÓN GÓMEZ-PÉREZ

Laboratorio de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo sn., Boadilla del Monte, 28660 Madrid, Spain

Abstract—*Based on the empirical verification of bibliographic-data and other Ontolingua ontologies, this paper provides an initial framework for verifying Knowledge Sharing Technology (KST). Verification of KST refers to the engineering activity that guarantees the correctness of the definitions in an ontology, its associated software environments and documentation with respect to a frame of reference during each phase and between phases of its life cycle. Verification of the ontologies refers to building the correct ontology, and it verifies that (1) the architecture of the ontology is sound, (2) the lexicon and the syntax of the definitions are correct and (3) the content of the ontologies and their definitions are internally and metaphysically consistent, complete, concise, expandable and sensitive. Copyright © 1996 Elsevier Science Ltd*

1. INTRODUCTION

DURING RECENT YEARS, considerable progress has been made in developing the conceptual bases for building technology that allows the reuse and sharing of knowledge. As libraries of definitions, ontologies are essential for both building intelligent systems and enabling interoperation of agents. Evaluation of ontologies as well as evaluation of the documentation and software environments is critical to the integration of this technology in real applications. In this sense, judging an ontology to be used for knowledge representation is just as necessary as getting a thorough inspection before purchasing a second-hand car. Just as one does not buy a car without first taking it to a mechanic for a checkup to ensure that the car is mechanically sound, it is unwise to publish your ontology, or to implement a software application that relies on ontologies written by others, without first evaluating and assessing their definitions and axioms. A well-evaluated ontology will not guarantee the absence of problems, but it will make its use safer.

The word "ontology" is a fashionable word in the knowledge engineering community. Although there exist many possible interpretations of this term, Guarino and Giaretta (1995) discuss different definitions and propose the following interpretation. An ontology is: (sense 1) a logical theory which gives an explicit, partial account of a conceptualization; (sense 2) a synonym of conceptualization. Although ontologies differ from knowledge bases (KBs) (Gómez-Pérez, 1994), both have a common foundational problem. Ontologies and KBs by nature are incomplete, as it is impossible to capture everything known about the real world in a finite structure. Since ontologies are developed incrementally by adding new definitions and modifying the old ones, one of the most important problems is to guarantee complete, consistent and concise definitions from the beginning, during each stage and between stages of its development process. The maintenance of ontologies would also require complete evaluation of the whole ontology if a definition is added, modified or removed.

Therefore, the knowledge sharing community needs to draw up a complete framework (Gómez-Pérez et al., 1995) with terminology, definitions, criteria, methods and tools for *Knowledge Sharing Technology (KST) evaluation*. This evaluation includes: ontologies, software and documentation. Some works have been done:

(a) Related to *terminology and definitions of terms*, the main terms are (Gómez-Pérez et al., 1995): "evaluation", "verification", "validation" and "assessment". In this paper, the differences between "evaluation" and "assessment" are also emphasized.
- "Evaluation of KST" subsumes "verification" and "validation". "Evaluation" means to judge the ontologies, their associated software environments and documentation technically with respect to a frame of reference during each

phase and between phases of their life cycle. Examples of frames of references are the real world, a set of requirements or a set of competency questions (Gruninger & Fox, 1994).

- "Verification of KST" refers to the technical activity that guarantees the correctness of an ontology, its associated software environments and documentation with respect to a frame of reference during each phase and between phases of its life cycle.
- "Validation of KST" guarantees that the ontologies, the software environments and the documentation correspond to the systems that they are supposed to represent.
- "Assessment of KST" refers to the usability and utility of the ontologies, software environments, and their documentation when they are reused by KBS or shared by software agents.

This paper is focused on verification of KST, that is, verification of ontologies, verification of software and verification of documentation. The next sections cover these issues.

(b) In relation to the *criteria* for evaluating knowledge sharing technology, a few examples of verification of class definitions, hierarchies and relations and functions appear in Gómez-Pérez (1995).

(c) With regard to the *methods*, competency questions (Gruninger & Fox, 1994) are proposed as a methodology for evaluating ontologies in the domain of enterprise engineering. The competency questions are the basis for a rigorous characterization of the problems that the ontology has to cover, and they specify the problem and what constitutes a good solution to the problem.

(d) Concerning *tools*, Ontolingua (Gruber, 1993a) provides a parser for legal KIF (Genesereth & Fikes, 1993) sentences, analyses of whether definitions are well formed and generates a report on undefined concepts and intra-ontology dependencies.

Since the evaluation of knowledge sharing technology is very immature and there is an absence of a deep core of previous ideas, *this paper is focused exclusively on providing some criteria and examples that guide the verification of KST.* The paper gathers the experience on the evaluation of the bibliographic-data (Gruber, 1994) and other Ontolingua (Gruber, 1993a) ontologies. It is organized as follows. Section 2 deals with how to verify ontologies. Section 3 shows how to verify software used to build ontologies. Finally, Section 4 shows verification of documentation.

2. VERIFICATION OF ONTOLOGIES

Ontology verification refers to correct building of the ontology, that is, ensuring that its definitions[1] correctly implement its requirements, its competence questions or perform correctly in the real world. Ontologies verification is orthogonal to the use of the definitions by any KBS or software agent. Ontology verification includes verification of:

(1) Each individual definition and axiom.
(2) Collection of definitions and axioms that are stated explicitly in the definitions of the ontology.
(3) Definitions that are imported from other ontologies.
(4) Axioms that can be inferred using other definitions and axioms.

To verify an ontology we have to determine the *correctness* of definitions and axioms by figuring out what the ontology explicitly defines, does not define or defines incorrectly. We also have to look at the scope of definitions and axioms by figuring out what can be inferred, cannot be inferred or can be inferred incorrectly. To guarantee that an ontology is well-verified, we have to judge its architecture, its lexicon and syntax and its content using criteria specified in Table 1.

2.1. Verification of the Architecture

At this point, we look to see if the structure of an ontology has been developed following the principles of design of the environment in which the ontology is included. For example, ontologies built in the Ontolingua environment should satisfy the five design criteria given by Gruber (1993b).

2.2. Verification of the Lexis and Syntax

The ontology definitions must be lexically and syntactically correct. The environment should provide a scanner to detect that the lexical structure of the expressions is correct, and a parser to detect that its syntactic structure is also correct. It is particularly important that the lexical and syntax analyzer compo-

TABLE 1
Levels and Criteria in the Verification of Ontologies

Levels	Criteria
Verification of the architecture	soundness
Verification of the lexicon and syntax	correctness
Verification of the content	consistency, completeness, conciseness, expandability and sensitiveness

[1] A definition is written in natural language (informal definition) and in a formal language (formal definition).

nents of the software environment rigorously implement the definitions of the lexis and grammar rules for the portable language. Failure to do this will allow the writing of non-portable definitions. As an example we have the use of wrong keywords in formal definitions.

2.3. Verification of Content

Verification of the content is concerned with the analysis of *completeness*, *consistency*, *conciseness*, *expandability* and *sensitiveness* of the definitions and axioms that are explicitly set out in the ontology, and with the analysis of those that can be inferred using other definitions and axioms.

2.3.1. Consistency. Consistency refers to whether it is possible to obtain contradictory conclusions from valid input data (Gómez-Pérez, 1996). With the goal of providing mechanisms that help to verify semantically the consistency of an ontology and its definitions, we assume that:

- A definition *Def* is composed of an informal definition I_{Def} and a formal definition F_{Def}.
- An informal definition I_{Def} is a free text documentation written in English.
- A formal definition F_{Def} is a collection of sentences written in a formal language.

$$F_{Def}=((Sent_1)...(Sent_n))$$

  Since the semantics of KIF (Knowledge Interchange Format)[2] unambiguously determines the referent of any term and the truth or falsity of any sentence, we assume that formal definitions are written in this language.
- Given a definition *Def*, the function $Interpretation_{FDef}(I_{Def})$ interprets the meaning of an informal definition I_{Def} with respect to its formal definition F_{Def}. This function maps the documentation string I_{Def} into the truth values true or false.

$$Interpretation_{F_{Def}}(I_{Def}):I_{Def}\Rightarrow\{true,false\}$$

- *Defined(Def Ont)* is a function that determines if the definition *Def* is defined in the ontology *Ont*.

$$Defined(Def\ Ont)=\begin{cases} true & if\ Def\ is\ defined\ in\ Ont \\ false & otherwise \end{cases}$$

- $Inferred(F_{Sent}\ Def\ Ont)$ is a function that determines if the formal sentence F_{Sent} is inferred using the definition *Def* and the ontology *Ont*.

$$Inferred(F_{Sent}\ Def\ Ont)=\begin{cases} true & if\ F_{Sent}\ is\ inferred\ using\ Def\ and\ Ont \\ false & otherwise \end{cases}$$

An ontology *Ont* is semantically consistent *S-Consistency(Ont)* if, and only if, each definition *Def* in the ontology is semantically consistent.

$$S\text{-}Consistency(Ont)\Leftrightarrow ((\forall Def)Defined(Def\ Ont)\wedge S\text{-}Consistency_{Ont}(Def))$$

A given definition *Def* in the ontology *Ont* is semantically consistent $S\text{-}Consistency_{Ont}(Def)$ if, and only if: (1) the individual definition is consistent and (2) no contradictory sentences may be inferred using other definitions and axioms.

$$(\forall Def, Ont)\ S\text{-}Consistency_{Ont}(Def)\Leftrightarrow (S\text{-}Individual\text{-}Consistency_{Def}(Def)\wedge S\text{-}Inferred\text{-}Consistency_{Ont}(Def))$$

2.3.1.1. Individual Consistency. A given definition *Def* is individually consistent $S\text{-}Individual\text{-}Consistency_{Def}(Def)$ if, and only if: (1) the definition *Def* is metaphysically consistent, that is, it is consistent with respect to the real world *RW* and (2) it is internally consistent.

$$S\text{-}Individual\text{-}Consistency_{Def}(Def)\Leftrightarrow S\text{-}Consistency_{RW}(Def)\wedge S\text{-}Consistency_{Def}(Def)$$

To guarantee that the definition *Def* is metaphysically consistent $S\text{-}Consistency_{RW}(Def)$, we prove that its formal as well as its informal definitions are metaphysically consistent.

$$S\text{-}Consistency_{RW}(Def)\Leftrightarrow (S\text{-}Consistency_{RW}(F_{Def})\wedge S\text{-}Consistency_{RW}(I_{Def}))$$

A formal definition F_{Def} is metaphysically consistent $S\text{-}Consistency_{RW}(F_{Def})$ if, and only if, there is no contradiction in the interpretation of the formal definition with respect to the real world. The goal is to prove compliance of the world model (if it exists and is known) with the world modeled formally. So, $S\text{-}Consistency_{RW}(F_{Def})$ maps a formal definition F_{Def} into the truth values true or false.

$$S\text{-}Consistency_{RW}(F_{Def}):F_{Def}\Rightarrow\{true,false\}$$

Since a formal definition is a set of KIF sentences, the

[2] The semantics of KIF is a correlation between the terms and sentences of the language and a conceptualization of the world. The *semantic value* of a term and the *truth value* of a sentence are defined using the notions of *interpretation* of constants and *variable assignment*. An interpretation is a function i that associates the constants of KIF with the elements of a conceptualization. A variable assignment is a function v that maps (1) individual variables V into objects in a universe of discourse O and (2) maps sequence variables W into finite sequences of objects. Given an interpretation and a variable assignment, the semantic value of every term in the language is a function s_{iv} from the set T of terms into the set O of objects in the universe of discourse. The truth value for sentences is defined as a function t_{iv} that maps sentences S into the truth values *true* or *false*.

function $S\text{-}Consistency_{RW}(F_{Def})$ is equivalent to determining the truth value of each KIF sentence $Sent_i$ in the formal definition.

$$S\text{-}Consistency_{RW}((Sent_1)...(Sent_n)) = \begin{cases} true & \Leftrightarrow t_{iv}(Sent_i) = true \text{ for all } i \text{ in } [1..n] \\ false & otherwise \end{cases}$$

An informal definition I_{Def} is metaphysically consistent $S\text{-}Consistency_{RW}(I_{Def})$ if, and only if, there is no contradiction in the interpretation of the informal definition with respect to the real world. The goal is to prove the compliance of the world with the world modeled informally. This function maps the documentation string I_{Def} into the truth values true or false.

$$S\text{-}Consistency_{RW}(I_{Def}): I_{Def} \Rightarrow \{true, false\}$$

We assure that the definition *Def* is internally consistent $S\text{-}Consistency_{Def}(Def)$, by proving that its formal as well as its informal definitions have the same meaning.

$$S\text{-}Consistency_{Def}(Def) \Leftrightarrow (Interpretation_{F_{Def}}(I_{Def}) = S\text{-}Consistency_{RW}(F_{Def}))$$

To prove the individual consistency of the definition MONTH-NAME in Example 1, we verify its internal and metaphysical consistency. As the terms used to name the months are the same in the formal and informal definitions, the definition of MONTH-NAME is internally consistent. However, both, its formal and informal definitions are metaphysically inconsistent because the term "house" is not a month in the real world. If we replace the term "house" by the term "January" in the formal definition of MONTH-NAME, then the whole definition is internally inconsistent, the formal definition is metaphysically consistent, and the informal definition is metaphysically inconsistent. To solve the inconsistencies, we replace the term "house" by "January" in the informal definition. However, if we were to replace the term "house" by the term "Enero" (this means January in Spanish), for those English speakers who are not Spanish speakers there is still a metaphysical inconsistency in the informal definition (something other than January is written in the informal definition). However, for those who are Spanish speakers, the formal definition and the informal definition are metaphysically consistent, but the whole definition is internally inconsistent because the symbols that name the months are different.

(Define-Class **MONTH-NAME** (?Month)
"The months of the year are: House, February, March, April, May, June, July, August, September, October, November, December"
:iff-def (Member ?Month (setof House February March April May June July August September October November December)))

Example 1. Internally consistent definition, but not metaphysically consistent

2.3.1.2. Inferred Consistency. For a definition to be inferentially and semantically consistent, it must be impossible to obtain contradictory conclusions using the meaning of all the definitions and axioms in the current logical theory. We guarantee the inferred consistency of a given definition $Inferred\text{-}Consistency_{Ont}(Def)$ by proving that if Δ is the set of inferred sentences for a given definition *Def*, (1) each inferred formal sentence F_{Sent} is individually consistent with respect to the definition *Def* and that (2) the set Δ of inferred sentences is internally consistent.

$$(\forall F_{Sent} F'_{Sent} \in \Delta)$$
$$(\forall Def, Ont, Ont')$$
$$(Inferred\text{-}Consistency_{Ont}(Def) \Leftrightarrow$$
$$(Defined(Def\ Ont) \wedge$$
$$Inferred(F_{Sent} Def\ Ont') \wedge$$
$$S\text{-}Individual\text{-}F_{Sent}\text{-}Consistency_{Def}(F_{sent}) \wedge$$
$$Inferred(F'_{Sent}\ Def\ Ont') \wedge$$
$$S\text{-}\Delta\text{-}Consistency(F_{Sent} F'_{Sent}))$$

To assure that an inferred formal sentence is individually consistent with respect to the definition $S\text{-}Individual\text{-}F_{Sent}\text{-}Consistency_{Def}(F_{Sent})$, we prove that: (1) there are no contradictions between the interpretation of the formal definition F_{Def} and the interpretation of the inferred formal sentence F_{Sent} with respect to the real world and (2) there are no contradictions between the interpretation of the informal definition I_{Def} regarding the formal definition F_{Def} and the interpretation of the inferred formal sentence F_{Sent} regarding the real world.

$$(\forall Def, F_{Sent}) S\text{-}Individual\text{-}F_{Sent}\text{-}Consistency_{Def}(F_{Sent}) \Leftrightarrow$$
$$(((S\text{-}Consistency_{RW}(F_{Def}) = S\text{-}Consistency_{RW}(F_{Sent})) \wedge$$
$$(Interpretation_{F_{Def}}(I_{Def}) = S\text{-}Consistency_{RW}(F_{Sent})))$$

We guarantee that a set Δ of inferred formal sentences is internally consistent $S\text{-}\Delta\text{-}Consistency(F_{Sent}\ F'_{Sent})$, by proving that there are no contradictions between the interpretation of any inferred formal sentence F_{Sent} and the interpretation of any other inferred formal sentence F'_{Sent}.

$$(\forall F_{Sent} F'_{Sent} \in \Delta) S\text{-}\Delta\text{-}Consistency(F_{Sent} F'_{Sent}) \Leftrightarrow (S\text{-}Consistency_{RW}(F_{Sent}) = S\text{-}Consistency_{RW}(F'_{Sent}))$$

Taking definitions in Example 2, the definition of KEYWORD would seem to be individually consistent. Since KEYWORD is a subclass of BIBLIO-TEXT, we can infer the formal sentence (string ?keyword), which means that ?keyword is a string. So, there is a semantically inferred inconsistency between the meanings of the inferred formal sentence and the informal definition of KEYWORD.

2.3.2. Completeness. Incompleteness is a fundamental problem in ontologies. In fact, we cannot prove either the

(Define-Class	(Define-Class	(Define-Class
BIBLIO-TEXT (?String)	**BIBLIO-NAME** (?String)	**KEYWORD** (?Keyword)
"The general class of	"A name of something in the	"A keyword is a number used as an
text objects"	bibliographic-data ontology"	index"
:def (String?String))	:def (Biblio-Text?String))	:def (Biblio-Name?Keyword))

Example 2. Inferred inconsistency.

completeness of an ontology or the completeness of its definitions, but we can prove both the incompleteness of an individual definition to derive the incompleteness of the ontology and the incompleteness of an ontology if at least a definition is missed. So, an ontology is semantically complete if, and only if:

(1) All that is supposed to be in the ontology is explicitly set out in it, or can be inferred using other definitions and axioms.
(2) Each definition is complete. Semantic completeness of a definition refers to the degree to which the definitions in a user-independent ontology cover the equivalent concepts in the real world. We determine the completeness of a definition by figuring out: (a) what information the definition defines or does not explicitly define about the world; and (b) for all the information that is not explicitly defined, but required, we check if it can be inferred using other axioms and definitions. If it can be inferred, the definition is complete. Otherwise, it is incomplete.

Completeness of the definitions concerns completeness of their formal and informal definitions. An *informal definition* written in natural language is complete if it expresses the same knowledge that the formal definition provides. To determine whether a formal definition or collection of formal definitions is complete we need a frame of reference, certain criteria to measure the degree of completeness and some guidelines to perform it. If there are no requirements or competency questions to be used as a frame of reference, other sources of information such as: the real world, relevant experts in developing ontologies, relevant users, books, examples, other ontologies could be used. In this case, the incompleteness of an ontology can be established by failure of test of any of the following three properties. *Scope*, which specifies the variety of different types of applications that might reuse or share the definitions; *exhaustiveness*, which refers to the level of precision of the definitions; *granularity*, which denotes the level of detail reached in each individual definition, as well as in the ontology.

In order to provide a mechanism to verify the completeness of an ontology, we assume that the world is conceptualized in terms of KIF objects, relations and functions. The following ordered set of activities might help you to find incomplete definitions in an ontology.

Step 1: Check completeness of the class hierarchy in which the current definition is included. The goal is to determine whether the superclasses of a given class exactly and precisely delimit the subclasses/superclasses of the appropriate class in the real world. Errors appear when: (a) the superclasses or subclasses of a given class are imprecise, over-specified or when they include classes that are not appropriate in the real world and (b) information about subclasses that are mutually disjoint or exhaustive subclass partitions are missing in their superclasses.

Assume the following ontolingua classes definitions[3]

Def. 1. (Define-Class **DOCUMENT** (?X)
"A document is something created by author(s) that may be viewed, listened to, etc., by some audience..."
:def (And (Individual-Thing?X)
(Has-One?X Title-Of)
(Has-One?X Number-Of-Pages-Of))
:axiom-def
(Subclass-Partition Document
(Setof Book Thesis Miscellaneous-Publication)))

Def. 2. (Define-Class **BOOK** (?X)
"Pages in a bound cover. You can't judge it by its cover."
:def (And (Document?X)
(Has-Some?X Has-Author)
(Has-One?X Title-Of)))

Def. 3. (Define-Class **THESIS** (?X)
"An official report on a bout of graduate work for which one receives a degree, published by the university. Never mind that some fields make a big deal about the difference between dissertations and theses. From the bibliographic perspective, they are both of the same family."
:def (And (Document?X)
(Has-One-Of-Type?X Organization-Of University)))

Def. 4. (Define-Class **MASTERS-THESIS** (?X)
"M.S. thesis document."
:def (Thesis?X))

[3] These definitions might not correspond with definitions in the Bibliographic-Data ontology.

Def. 5. (Define-Class **DOCTORAL-THESIS** (?X)
"Ph.D. thesis document."
:def (Thesis?X))

Def. 6. (Define-Class **MISCELLANEOUS-PUBLICATION** (?X)
"A miscellaneous category of documents that are infrequently found in bibliographic references"
:def ((Document?X))

Def. 7. (Define-Class **COMPUTER-PROGRAM** (?X)
"The Has-Author is the programmer."
:def (Miscellaneous-Publication?X))

Def. 8. (Define-Class **PICTURE** (?X)
""
:def (Miscellaneous-Publication?X))

Their attached hierarchy is given in Fig. 1—which does not exactly correspond to the hierarchy of the bibliographic-data ontology (Gruber, 1994)—in which, bold words represent classes, italic words mean properties attached to the class, plain lines between classes represent subclass-of relations between classes and dashed lines mean that the subclasses of a class are mutually disjoint. Notice that:

- The classes Doctoral-Thesis and Master-Thesis are an exhaustive subclass partition of the class THESIS. The following Ontolingua sentence should be included in the definition of Thesis.
  :axiom-def
  (Exhaustive-Subclass-Partition Thesis
  (Setof Masters-Thesis Doctoral-Thesis)))
- The classes Computer-Program and Picture should be mutually disjoint with respect to the class Miscellaneous-Publication. The following should be included in the definition of Miscellaneous-Publication.
  :axiom-def
  (Subclass-Partition Miscellaneous-Publication
  (Setof Computer-Program Picture)))
- The specialization of the class Book into different subfields, that is, Computer-Book, Chemistry-Book and so on is possible. Ontologies builders should decide before building ontologies the level of granularity and what to cut out.

Step 2: Check the completeness of the domains and ranges of the functions and relations and that the domains of these functions and relations are defined in the class hierarchy of the ontology being verified. The aim is to figure out whether the domain (or range) of each argument of each function or relation in the ontology exactly and precisely delimits the classes that are appropriate for that argument. Errors appear when the domains and ranges are imprecise, overspecified or completely wrong.

For a given subgraph of the class hierarchy of the current ontology, we find errors in the domain and range of its functions and relations when we fill in their *tables of domains and ranges*. These tables allow us to compare the old and new domains and ranges of the functions and relations in a hierarchy. In them, column 1 gathers the names of all the functions and relations whose domains are in the hierarchy. Columns 2 and 4 represent their original domains and ranges (as they are defined in the ontology you are verifying). Finally, columns 3 and 5 are the new domains and ranges of the functions and relations if they have to be modified (as you think they

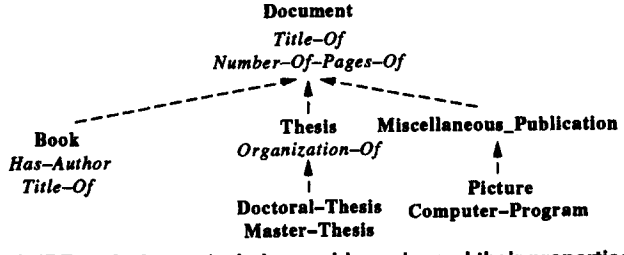

FIGURE 1. A classes/subclasses hierarchy and their properties.

TABLE 2
Domains and Ranges for the Functions of the Hierarchy in Fig. 1

Definition	Original domain	New domain	Original range	New range
Title-Of	Document	—	Title	—
Number-Of-Pages-Of	Document	—	Natural	—
Organization-Of	Book	Document	Organization	—
Conference-Of	Proceeding	—	—	Conference
University-Of	Document	Thesis	University	—

TABLE 3
Domains and Ranges for the Relations of the Hierarchy in Fig. 1

Definition	Original domain	New domain	Original range	New range
Has-Author	Document	—	Author	—

should be defined).

Assume the following Ontolingua functions and relations definitions. Tables 2 and 3 summarize the domains and ranges of some functions and relations that have as a domain some classes in the hierarchy of the Fig. 1.

Def. 9. (Define-Relation **Has-Author** (?Doc? Author)
"The creator(s) of a document. Not necessarily the author of a work published in the document, but often so. The author is a real agent, not a name of an agent."
:def (And (Document?Doc)
(Author?Author)))

Def. 10. (Define-Function **Title-Of** (?Doc):→?Title
"The title of a document. Not necessarily the title of a work published in the document."
:def (And (Document?Doc)
(Title?Title)))

Def. 11. (Define-Function **Conference-Of** (?Proc):→ ?Conference
"The conference associated with a proceedings."
:def (And (Proceedings ?Proc)))

Def. 12. (Define-Function **Number-Of-Pages-Of** (?Doc):→?N
"Number of pages contained in a document. Not the page numbers of an article."
:def (And (Document?Doc)
(Natural?N)))

Def. 13. (Define-Function **Organization-Of** (?Doc):→?Organization
"The institution that publishes a document, like a University or trade association."
:def (And (Book?Doc)
(Organization?Organization)))

Def. 14. (Define-Function **University-Of** (?Doc):→?University
"The University that publishes a thesis."
:def (And (Document ?Doc)
(Organization ?University)))

Taking these tables and hierarchy, we can say that:

(a) The domain and range of the functions Title-Of (the title of a document) and Number-Of-Pages-Of (number of pages of a document) are well-defined.
(b) The domain and range of the relation Has-Author (the author of a document) is well-defined.
(c) The domain of the function University-Of (the university of a thesis) is over-specified.
(d) The domain of the function Organization-Of (the organization that publishes a document) is imprecise—any document has an institution that publishes it.

So, definitions 11, 13 and 14 are modified as follow:

Def. 11. (Define-Function **Conference-Of** (?Proc):→ ?Conference
"The conference associated with a proceedings."
:def (And (Proceedings ?Proc)
(*Organization ?Conference*)))

Def. 13. (Define-Function **Organization-Of** (?Doc):→ ?Organization
"The institution that publishes a document, like a university or trade association."
:def (And (*Document ?Doc*)
(Organization ?Organization)))

Def. 14. (Define-Function **University-Of** (?Doc):→?University
"The university that publishes a thesis."
:def (And (*Thesis ?Doc*)
(Organization ?University)))

Step 3: Check the completeness of the classes. The goal is to know if the class gathers as much information as possible. So, the class should be defined by a predicate defined by necessary and sufficient conditions, and the set of properties attached to a given class represent the set of properties that the class owns in the real world. In this case, errors appear when:

(a) *There are missed properties in the definition of a class.* We discover missed properties by checking that all the functions and relations that have the class as a domain are included as properties in the definition of the class. For example, going through columns 2 and 3 in Tables 1 and 2, we detect some potential properties (Title-Of, Number-Of-Pages-Of, Has-Author, Organization-Of) of the class DOCUMENT by selecting those functions and relations whose domain is DOCUMENT. Since the class DOCUMENT (see definition 1) only

owns the properties Title-Of and Number-Of-Pages-Of, we can say that the definition is incomplete. To make the definition complete, we introduce the missed properties (Organization-Of and Has-Author) in the class DOCUMENT to guarantee that the class as well as its subclasses can have these properties defined. The following Ontolingua sentences should be included in the formal definition of DOCUMENT:

(Has-Some ?X Has-Author)
(Has-One ?X Institution-Of)

(b) *There are errors in the cardinality of any property.* We detect errors in the cardinality by comparing that the cardinality of the properties in the world modeled formally is that which it is supposed to have in the real world. Check also that the minimum cardinality of a property is inferior to the maximum cardinality. For example, in the class THESIS we constrain the values of the inherited properties Has-Author (a thesis only has *one* author) when we use the following Ontolingua sentence:

(Has-One ?X Doc.Author)

(c) *Different classes have the same formal definition.* We find equal formal definitions by checking classes that: (1) are classified under the same superclasses, (2) own the same set of properties and (3) the properties have the same cardinality. Checking definitions 4 and 5, we find that there are no semantic differences between the classes MASTER-THESIS and DOCTORAL-THESIS because they do not have any property that differentiates them. We solve the problem by defining a new function Degree-Of in the domain of THESIS and in the range of DEGREE. We differentiate between the two classes by including the Ontolingua sentence (=Degree-Of Ph.D.) in the formal definition of DOCTORAL-THESIS and (=Degree-Of M.S.) in the formal definition of MASTER-THESIS. The definitions of the function Degree-Of and the definition of the class DEGREE are given in Example 3.

(Define-Function **DEGREE-OF** (?Thesis)
:→?Degree
"The degree of a thesis work."
:def (and (Thesis?Thesis)
(Degree?Degree)))

(Define-Class **DEGREE** (?Degree)
"The degree of a study."
:axiom-def
(Subclass-Partition Degree
(Setof B.S. M.S. Ph.D.)))

Example 3. Definitions of a new function and class

(d) *The class does not include properties that it cannot have in the real world.* The goal is to find out which properties the class cannot have in the real world and we include this information in the definition of the class. In particular they should be included if they may be inherited from its superclasses. Looking at Fig. 1, we know that a PICTURE is a subclass of the class DOCUMENT, and that all documents can have pages. The Ontolingua sentence (Cannot-Have ?X Number-Of-Pages-Of) — or the equivalent in your language — would forbid the definition of the property Number-Of-Pages-Of in the instances of PICTURE.

2.3.3. Conciseness. Conciseness refers to whether all the information gathered in the ontology is useful and precise. Conciseness does not imply absence of redundancies. Sometimes, some degree of controlled redundancy can be useful in definitions. *A priori*, it is difficult to prognosticate the conciseness of an ontology or set of ontologies because they provide as many abstract definitions as possible for a given domain. An ontology is concise if:

(a) It does not store any unnecessary or useless definition.
(b) Explicit redundancies do not exist between definitions. For example, if a class is extensionally-defined by enumerating a set of objects, and these objects are defined as instances in the ontology, the ontology is redundant. Example 4 shows a case of explicit redundancy.

(Define-Class **MONTH-NAME** (?Month)
"The months of the year, specified as an extensionally-defined (i.e. enumerated) set of objects, in English. Instances of this class of months are not symbols, they are months that may be denoted by object constants."
:iff-def (Member?Month
(setof january february march april may june july august september october november december)))

(Define-Instance JANUARY (Month-Name))

Example 4. Explicit redundancy

(c) Redundancies cannot be inferred using axioms attached to other definitions. Examples of inferred redundancies are:

- A property that can be inherited from a superclass is defined explicitly in any of its subclasses. For example, the sentence (Has-One ?X Title-Of) in the class THESIS, would make it redundant because we can get this property

from DOCUMENT by using inheritance. So, it must be removed in BOOK.

- A subclass-of relation could be inferred using other definitions. Given the definitions in Example 5 we could infer from the definition of EXACT-RANGE—the EXACT-RANGE is the class whose instances are exactly those that appear in the last item of any tuple in the relation—that the class AGENT-NAME is a subclass of BIBLIO-NAME. Since AGENT-NAME is the EXACT-RANGE of the AGENT.NAME function, and a range of AGENT.NAME is BIBLIO-NAME, and since the EXACT-RANGE of a binary relation is a subclass-of any of ranges, then it follows that AGENT-NAME is a subclass of BIBLIO-NAME. Consequently, the definition of AGENT.NAME is concise and the inclusion of the constraint in the definition makes it redundant.

(Define-Class **AGENT-NAME** (?Name)
"A string that is the name of some agent."
:def (Biblio-Name?Name)
:axiom-def (Exact-Range Agent.Name Agent-Name))

(Define-Function **AGENT.NAME** (?Agent)
:→?Name
"Function from an agent to the name by which it goes."
:def (and (Agent?Agent)
(Biblio-Name?Name)))

(Define-Class **BIBLIO-NAME** (?String)
"A name of something in the bibliographic-data ontology."
:def (Biblio-Text?String))

Example 5. An implicit redundancy is inferred

(d) A definition is itself redundant. Given the definitions in Examples 5 and 6, we can say that the definition of ORGANIZATION.NAME is redundant. It is explicitly said in Example 6 that the function ORGANIZATION.NAME is a specialization of the function AGENT.NAME. If the domain and range of the ORGANIZATION-.NAME function are specializations of the domain and range of the AGENT.NAME function, then all the tuples in the ORGANIZATION.NAME function are specializations of those in AGENT.NAME. We have to delete the sentence (Agent.Name ?Organization ?Name) in ORGANIZATION.NAME to make it non-redundant.

(Define-Function **ORGANIZATION.NAME**
(?Organization):→?Name
"The name by which organizations go by. One name per place."
:def (and (Organization?Organization)
(Biblio-Name?Name)
(Agent.Name?Organization?Name)))

(Define-Class **ORGANIZATION** (?X)
"An organization is a corporate orsimilar institution, distinguished from persons and other agents."
:def (Agent?X))

Example 6. The function ORGANIZATION.NAME is itself redundant

2.3.4. Expandability and Sensitiveness. Expandability refers to the effort required in adding new definitions to an ontology, as well as the effort needed to add new information to a definition, without altering the set of well-defined properties that are already guaranteed after the ontologies verification process.

2.3.5. Sensitiveness. Sensitiveness relates to how small changes in a given definition alter the set of well-defined properties that are already guaranteed. After including or modifying a definition, this criterion must guarantee that:

(1) The architecture of the ontology and the architecture of its definitions are still sound.
(2) The definitions are lexically and syntactically correct.
(3) The ontology and its definitions of conciseness, consistency and completeness are tightly connected.

3. VERIFICATION OF SOFTWARE

Software verification refers to building the software right, which means that the software that builds, reuses and shares definitions and axioms correctly and completely implements its requirements. Software engineering methodologies, techniques and tools provide the appropriate framework to verify KST software in each stage and between stages of its life cycle.

4. VERIFICATION OF DOCUMENTATION

Documentation verification refers to building the documents correctly. It seeks to guarantee that all the required documents have been written, that nothing has been overlooked in any document and that the documents evolve in step with definitions and software environments in each phase and between phases of the life cycle. Verification of the documentation includes: the natural

language string in each definition, general information about the ontology, basic ontological commitments, a summary of definitions, cases studies, definitions taken from other ontologies and also documentation about the software that the environment provides, installation manual, reference manual, release notes, frequently asked questions and tutorials.

Special attention is required if WWW documents are indexed automatically using a program. In this case, mistakes in the indexes of the natural language documentation appear easily due to the creative and flexible use of the language. From the information retrieval point of view, four categories of words can be found in the indexed text.

(a) *Correctly indexed words* represent words in the free text documentation that are properly indexed with a word in the ontology vocabulary.
(b) *Correctly non-indexed words* represent words in the free text documentation that are not indexed with a word in the ontology.
(c) *Incorrectly indexed words* include words that have been wrongly indexed with the ontology vocabulary. Errors in the indexes are classified in the following categories:
    - An index *semantic* blunder arises in natural language documentation when the meaning of the word in the documentation string is not the same as the meaning of the term pointed in the ontology vocabulary.
    - A *context* error appears when there are no semantic errors in the pointer, but the word in the documentation string is not used in the ontology theory context.
    - *Miscellaneous* mistakes cover loops in indexes and problems in polymorphical definitions. While the former deal with indexes from words in the natural language documentation of a definition to the definition itself, polymorphical errors deal with several and different definitions of the same word in different ontologies. Multiple definitions create ambiguity in the selection of the indexes.
(d) *Incorrectly non-indexed words* concern words used in the free text documentation in the ontology theory context that are not indexed with the ontology vocabulary because it is spelt differently.

A study performed on Ontolingua ontologies reveals that the majority of the errors can be easily avoided if the ontology writer writes the words to be indexed using certain conventions (i.e. using uppercase for all the words, and/or using hyphenated strings of words). Assuming that the natural language documentation has been written following these conventions, the following heuristics will provide new semantic, context and morphological capabilities in the program that automatically generates the indexes:

(1) Pluralization of hyphenated and non-hyphenated words in the lexicon.
(2) Detection of situations in which a word is followed by unusual punctuation marks.
(3) Automatic generation of hyphenated words.
(4) Prevention of pointers to words that are out of the scope of the current ontology and ontologies that are included in the current ontologies.
(5) If a polymorphic word and a name of an ontology appear together in a sentence, the polymorphical word should point to the definition in that ontology.
(6) If a polymorphic definition is made in an ontology, any index of the word in the ontology should point to its definition, unless the name of any other ontology appears in the sentence.
(7) Given a word, any index to that word from its natural language documentation must be prevented.

5. CONCLUSIONS

Based on the empirical verification of ontolingua ontologies, a novel approach to verify KST has been illustrated. The main contributions are:

(a) We create a framework to verify KST. This framework includes terminology, definitions, criteria and examples to carry out the verification.
(b) We split the verification process in three processes: verification of ontologies, verification of software for building, reusing and sharing definitions and verification of documentation. The most important is verification of the ontologies. Software engineering provides the framework to verify KST software and documentation.
(c) Verification of the ontologies includes verifying that: the architecture of the ontologies and definitions are *sound*, the lexicon and syntax are *correct* and the content of the definitions is *consistent*, *complete*, *concise*, *expandable and sensitive*. Regarding the content, we provide:
    - A formal definition of internal, metaphysical and inferred consistency, and examples that show how to deal with these new concepts.
    - An informal definition of completeness, and stereotype of errors that make relations, functions and classes incomplete.
    - An informal definition of conciseness and kinds of errors that make ontologies redundant.
    - We define expandability and sensitiveness of an ontology, and we identify which kind of verification has to be performed when definitions are added or modified in an ontology.

Finally, we remark that conciseness, consistency and

completeness are tightly connected. An ontology can be complete and not be concise if the formal sentence written in a formal definition can be inferred using other definitions. However, if the sentences are not explicitly written and they cannot be inferred, the ontology could be concise or not, but it is not complete.

Actually, we are developing a tool called ONE-T (ONtologies Evaluation Tool) for Ontolingua ontologies. The tool detects mistakes and omisions in ontologies and corrects them automatically whenever it is possible. So, it increases the performance and quality of the evaluation process.

Acknowledgements—The research work has been performed at the Knowledge Systems Laboratory in Stanford University, Stanford, CA, U.S.A. It has been sponsored by grant number PF94-8821929 of the Ministerio de Educación y Ciencia in Spain. Thanks to Richard Fikes for his comments on the paper and advice during the work and to Mike Uschold for his comments.

Genesereth, M. R. & Fikes, R. E. (1993). Knowledge interchange format, Version 3.0. Reference manual. Report Logic-92-1, Computer Science Department, Stanford University.

Gómez-Pérez, A. (1994) From knowledge based systems to knowledge sharing technology: Evaluation and assessment. Technical Report KSL 94-73, Knowledge Systems Laboratory, Stanford University.

Gómez-Pérez, A. (1995). Some ideas and examples to evaluate ontologies. In *Proceedings of the Eleventh IEEE Conference on Artificial Intelligence for Applications*, pp. 299–305. New York: IEEE Computer Society Press.

Gómez-Pérez, A. (1996). Guidelines to verify completeness and consistency in ontologies. *The Third World Congress on Expert Systems*.

Gómez-Pérez, A., Juristo, N. & Pazos, J. (1995). Evaluation and assessment of knowledge sharing technology. In *Towards very large knowledge bases: Knowledge building and knowledge sharing*, pp. 289–296. IOS Press.

Gruber, T. (1993a). A translation approach to portable ontology specifications. *Knowledge Acquisition*, **5**, 199–220.

Gruber, T. (1993b). Toward principles for the design of ontologies used for knowledge sharing. Technical Report KSL 93-04, Knowledge Systems Laboratory, Stanford University.

Gruber, T. (1994). Bibliographic-Data ontology. Available at hpp. stanford.edu, as http://www-ksl.stanford.edu/knowledge-sharing /ontologies/html/bibliographic-data/index.html

Gruninger, M. & Fox, M. S. (1994). The role of competency questions in enterprise engineering. IFIP WG5.7 workshop on benchmarking, theory and practice. Trondheim, Norway.

Guarino, N. & Giaretta, P. (1995). Ontologies and knowledge bases: Towards a terminological clarification. In *Towards very large knowledge bases: Knowledge building and knowledge sharing*, pp. 25–32. IOS Press.