# Criteria to Verify Knowledge Sharing Technology

**Asunción Gómez-Pérez**

Knowledge Systems Laboratory
Stanford University
701 Welch Road, Building C
Palo Alto, CA, 94304, USA
Tel: (415) 723-1867, Fax: (415) 725-5850
Email: gomez@hpp.stanford.edu, Asun@fi.upm.es

## Abstract

This paper is focused on providing some criteria 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 software and documentation guarantee that they are correct. Verification of the ontologies refers to building the ontology right, 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.

## 1. Introduction

During recent years, considerable progresses has been made in developing the conceptual bases for building technology that allows the reuse and sharing of knowledge. As libraries of definitions, ontologies avoid the need to build a Knowledge Based System (KBS) from scratch by allowing KBS developers to assemble reusable components [9], and they allow communication among software agents [5, 8] by establishing common vocabularies and semantic interpretations of terms. Evaluation of ontologies as well as evaluation of the documentation and software environments is critical to the integration of this technology in real applications. If incorrect definitions coexist with specific knowledge in the Knowledge Base (KB), the KBS might arrive at mistaken conclusions. If ontology definitions and software environments have not been sufficiently evaluated, communication between software agents may not succeed. A well-evaluated ontology will not guarantee the absence of problems, but it will make its use easier.

Although ontologies differ from KBs [2], both have a common foundational problem. Ontologies and KBs by nature are incomplete, as it is impossible to capture all that it is 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 start, during each stage and between stages of the development process. After leaving the industrial or academic lab and going to the market place, any of the definitions of the ontologies may be modified or deleted, and new definitions could be added. The maintenance of

ontologies would usually require complete evaluation of the whole ontology if any definition is added, modified or removed. If new or modified definitions from the ontology replace old ones in a well-evaluated KBS, we need to evaluate the KBS again because the imported definitions may alter KB consistency and the behavior of the KBS. Therefore, the KST community needs to draw up a set of guidelines (terminology, definitions, criteria, methods and tools) to evaluate ontologies during their life cycle, as well as any definitions reused in a KBS or shared by software agents. Some works have been done on the evaluation of KST.

- Related to *terminology and definitions of terms*, Gómez-Pérez identifies the main terms [2]. In this paper, the differences between "Evaluation" and "Assessment" are 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 [7]. "Verification" 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" guarantees that the ontologies, the software environments, and the documentation correspond to the systems that they are supposed to represent. "Assessment" 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.

- In relation to the *criteria* for evaluating ontologies, Gómez-Pérez [3] provides a few examples of how to detect the absence of some well-defined properties of definitions in ontologies.

- With regard to the *methods*, Gruninger and Fox [7] use a set of competency questions 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.

- Concerning *tools*, Ontolingua [4] provides a parser for legal KIF [1] sentences, analyses of whether definitions are well formed, and generates a report on undefined concepts and intra-ontology dependencies.

In view of the immaturity of the field and the absence of a core of previous ideas, *this paper is focused exclusively on providing some criteria that guide the verification of KST*. Taking the previous definition of "Verification" as a point of departure, we subdivide the paper into three sections. Section Two is the main contribution. There, we provide a set of criteria that guarantee a set of well-defined properties in the structure, the syntax, and the content of the ontologies. Section Three briefly describes how to perform verification of the software environments for building, reusing and sharing definitions. Finally, Section Four provides a set of criteria and ideas for verifying the documentation generated.

## 2. Verification of ontologies

Ontology verification refers to building the ontology right, that is, ensuring that its definitions[1] correctly implements its requirements, its competence questions or the real world, but it is not related to the use of the definitions by KBS and software agents. Ontology verification includes verification of: (1) each individual definition and axiom, (2) the collection of definitions and axioms that are set out explicitly in the ontology, (3) the definitions that are imported from other theories, and (4) the set of axioms that could be inferred using other definitions and axioms. Table 1 shows the levels and criteria of this step.

| LEVELS | CRITERIA |
|---|---|
| Verification of the structure | Soundness |
| Verification of the lexicon and syntax | Correctness |
| Verification of the content | Consistency, Completeness, Conciseness, Expandability and Sensitiveness |

Table 1. Levels and criteria in the verification of the ontologies

### *2.1 Verification of the architecture*

The *soundness* criterion will be used to verify the structure of an ontology and its definitions. The structure of an ontology is sound if it has been developed following the principles of design of the environment in which the ontology is included. So, verification of the ontology structure is performed against these principles, and the environment that provides the principles should provide techniques, methods and examples to help to prove that specific ontologies satisfy those criteria. For example, ontologies built in the Ontolingua environment [4] should satisfy the five design criteria given by Gruber [5]: Clarity, Coherence, Extendibility, Minimal Encoding Bias, and Minimal Ontological Commitment.

### *2.2 Verification of the lexicon and the 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 correct too. The scanner-parser pair has to detect the following incidences:

a) Use of wrong keywords in formal definitions.

b) Detection of undefined concepts in formal definitions.

c) Absence of informal and formal definitions for a given definition.

d) Loops between definitions. A loop between definitions can be detected when definition $D_1$ is defined in terms of definition $D_2$ and vice versa. Loops between definitions sometimes enable one to determine the truth

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

value of any definition in the loop. Given definitions in example One, the semantics of KIF [1] tell us that to prove the truth value of AUTHOR.NAME we must prove that every conjunct is *true* in the conjunction, and that at least one of the disjuncts is *true* in the disjunction. Having proved that the truth value of the two first sentences of AUTHOR.NAME is *true*, the loop between PENNAME and AUTHOR.NAME occurs when the truth value of the sentence (agent.name ?author ?name) is *false*. Then, we must prove that PENNAME is *true* for AUTHOR.NAME to be *true*. Since PENNAME is defined in terms of AUTHOR.NAME, the loop between the definitions prevents us from reaching any conclusion. Note that the loop does not appear if the sentence (agent.name ?author ?name) is *true*.

```
((define-relation   AUTHOR.NAME (?author ?name)
"An author name is the name of an agent used
to identify it as an author.  It is not necessarily
unique; authors may go by pseudonyms."
  :def (and  (author ?author)
             (biblio-name ?name)
             (or (agent.name ?author ?name)
                 (penname ?author ?name))))
```

```
((define-relation PENNAME    (?author ?name)
  "An author's pseudonym [Webster].
An author may use several pseudonyms.
Which name is a function of the document."
  :def (author.name ?author ?name))
```

Example 1. A loop between definitions

### *2.3 Verification of content*

Verification of the content is concerned with the analysis of the *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 get contradictory conclusions from valid input data. With the goal of providing mechanisms that help to verify the consistency of an ontology and its definitions formally, 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 string written in English.
- A formal definition $F_{Def}$ is a collection of sentences written in a formal language.

$$F_{Def} = ((Sent_1) \ldots (Sent_n))$$

Since the semantics of KIF[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.

[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

- Given a definition *Def*, the function *Interpretation* $F_{Def}(I_{Def})$ 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*.

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

- *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*.

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

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

$$S\text{-}Consistency\ (Ont) <=> ((\forall\ Def)\ Defined\ (Def\ Ont)\ \Lambda\ S\text{-}Consistency_{Ont}\ (Def))$$

A given definition *Def* in the ontology *Ont* is semantically consistent *S-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.

$$(\forall\ Def, Ont)\quad S\text{-}Consistency_{Ont}\ (Def) <=> (Defined\ (Def\ Ont)\ \Lambda\ S\text{-}Individual\text{-}Consistency_{Def}\ (Def)\ \Lambda\ S\text{-}Inferred\text{-}Consistency_{Ont}\ (Def))$$

**Individual Consistency.**

We can say that a definition *Def* is individually consistent *S-Individual-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.

$$S\text{-}Individual\text{-}Consistency_{Def}\ (Def) <=> S\text{-}Consistency_{RW}\ (Def)\ \Lambda\ S\text{-}Consistency_{Def}\ (Def)$$

---

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*.

We guarantee that the definition *Def* is metaphysically consistent *S-Consistency* $_{RW}$ (*Def*), by proving that its formal as well as its informal definitions are metaphysically consistent.

*S-Consistency* $_{RW}$ (*Def*) <=> (*S-Consistency* $_{RW}$ ($F_{Def}$) $\Lambda$ *S-Consistency* $_{RW}$ ($I_{Def}$))

A formal definition $F_{Def}$ is metaphysically consistent *S-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-Consistency* $_{RW}$ ($F_{Def}$) maps a formal definition $F_{Def}$ into the truth values *true* or *false*.

$$\textit{S-Consistency}_{RW}\ (F_{Def}) : F_{Def} \Rightarrow \{true, false\}$$

Since a formal definition is a set of KIF sentences, the function *S-Consistency* $_{RW}$ ($F_{Def}$) is equivalent to determining the truth value of each individual KIF sentence $Sent_i$ in the formal definition.

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

An informal definition $I_{Def}$ is metaphysically consistent *S-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 model with the world modeled informally. This function maps the documentation string $I_{Def}$ into the truth values *true* or *false*.

$$\textit{S-Consistency}_{RW}\ (I_{Def}) : I_{Def} \Rightarrow \{true, false\}$$

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

*S-Consistency* $_{Def}$ (*Def*) <=> (*Interpretation* $_{F_{Def}}$ ($I_{Def}$) = *S-Consistency* $_{RW}$ ($F_{Def}$))

For example, to prove the individual consistency of the definition MONTH-NAME in example 2, we have to prove that:

a) The whole definition is internally consistent by proving that the formal as well as the informal definitions have the same meaning.

b) The formal as well as the informal definitions are metaphysically consistent.

```
(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 2. Internally consistent, but not metaphysically consistent.

As the terms used to name the months are the same in the formal and informal definitions, we can say that 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, we can say that:

- The whole definition is internally inconsistent.
- The formal definition is metaphysically consistent.
- 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 (that 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 of the symbols that name the months are different.

**Inferred consistency**

It refers to the impossibility of getting contradictory conclusions using the meaning of definitions and axioms that belong to the same ontology and ontologies included by the current ontology. We guarantee the inferred consistency of a given definition *Inferred-Consistency $_{Ont}$ (Def)* by proving that if $\Delta$ 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 $\Delta$ of inferred sentences is internally consistent.

$$(\forall\ F_{Sent}\ F'_{Sent} \in \Delta)\ (\forall\ Def, Ont, Ont')$$

$$\begin{aligned}(Inferred\text{-}Consistency_{\ Ont}\ (Def) \iff\ & (Defined\ (Def\ Ont)\ \wedge\ Inferred\ (F_{Sent}\ Def\ Ont')\ \wedge \\ & S\text{-}Individual\text{-}F_{Sent}\text{-}Consistency_{\ Def}\ (F_{sent})\ \wedge \\ & Inferred\ (F'_{Sent}\ Def\ Ont')\ \wedge \\ & S\text{-}\Delta\text{-}Consistency\ (F_{Sent}\ \ F'_{Sent}))\end{aligned}$$

To assure that an inferred formal sentence is individually consistent with respect to the definition *S-Individual-F$_{Sent}$-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.

$$\begin{aligned}(\forall\ Def, F_{Sent})\quad & S\text{-}Individual\text{-}F_{Sent}\text{-}Consistency_{\ Def}\ (F_{Sent}) \iff \\ & (((S\text{-}Consistency_{\ RW}\ (F_{Def}) = S\text{-}Consistency_{\ RW}\ (F_{Sent}))\ \wedge \\ & (Interpretation_{\ F_{Def}}\ (I_{Def}) = S\text{-}Consistency_{\ RW}\ (F_{Sent}))\end{aligned}$$

We guarantee that a set $\Delta$ of inferred formal sentences is internally consistent *S-Δ-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 3, the definition of KEYWORD would seem to be individually consistent. Knowing that a KEYWORD is a subclass of BIBLIO-TEXT, formally, we can derive the formal sentence (string ?keyword), which means that ?keyword is a string. So, there is a semantically inferred inconsistency between the inferred formal sentence (string ?keyword) and the informal definition of KEYWORD.

```
(define-class
   BIBLIO-TEXT   (?string)
"The general class of
text objects."
 :def (string ?string))
```

```
(define-class
   BIBLIO-NAME (?string)
"A name of something in the
bibliographic-data ontology."
 :def (biblio-text ?string))
```

```
(define-class
   KEYWORD   (?keyword)
"A keyword is a number
used as an index."
:def  (biblio-name  ?keyword))
```

Example 3. Inferred inconsistency

### 2.3.2. Completeness

Completeness refers to the extension, degree, amount or coverage to which the definitions in a user-independent ontology cover the information of the real world. In order to provide a mechanism that helps to verify the completeness of an ontology, we assume that the world is conceptualized in terms of KIF objects[3], relations[4] and functions[5].

A given ontology is semantically complete if and only if: all that is supposed to be in the ontology is explicitly set out in it, or can be inferred using other definitions and axioms. We determine the completeness of an ontology by checking that:

a) There is some explicit or inferred definition or axiom for each requirement or competency question.

b) If there are no requirements or competency questions to be used for verification purposes, the completeness of the ontology could be proved by analyzing the following properties:

   - *Scope*, which specifies the variety of different types of applications that might reuse or share the definitions.

[3]The basic objects in KIF are: words, all the complex numbers, list of objects, sets of objects and bottom [1].

[4]*A relation is an arbitrary set of finite lists of objects (of possibly varying length) [1].* A relation maps elements of a domain onto element of a range. For each tuple in the relation, the last item is in the range, and the tuple formed by the preceeding items is in the domain. In Ontolingua, a class is a unary relation, that is, a set of tuples of length one.

[5]A function is a case of a relation. *For every finite sequence of objects (called arguments), a function associates a unique object (called the value) [1]*

- *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 this case, other sources of information such as: the real world, relevant experts in developing ontologies, relevant users, books, examples, other ontologies, etc. could be used as a frame of reference.

We prove the completeness of an ontology by proving the completeness of each definition. The completeness of a definition depends on the level of granularity agreed to in the whole ontology. We determine the completeness of a definition by figuring out:

1. What information of the world explicitly the definition defines or does not define.

2. For the information that is not explicitly defined, 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 figure out if a **function or relation's formal definition is complete**, we check that: (1) the domain of the functions or relations exactly and precisely delimits the classes that can have defined these relations and functions as a property, and (2) the range of the functions or relations exactly and precisely delimits the class of values to which the values of the functions and relations belong to. Errors appear when the domains and ranges are imprecise, over-specified or completely wrong. For a given hierarchy, we find the errors in the domain and range of its functions and relations when we fill 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 One gathers the name of all the functions and relations whose domains are in the hierarchy. Column Two and Four represent their original domains and ranges such as they are defined in the ontology. Finally, column Three and Five are the new domains and ranges.

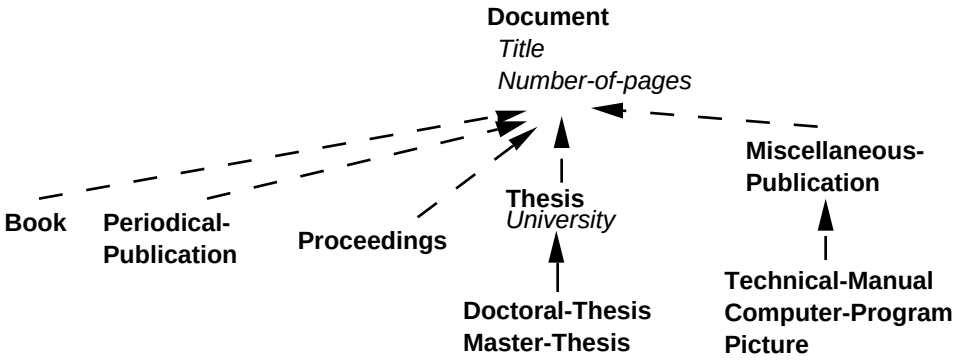


Figure 1. A Classes/Subclasses hierarchy and their properties.

| DEFINITION | ORIGINAL DOMAIN | NEW DOMAIN | ORIGINAL RANGE | NEW RANGE |
|---|---|---|---|---|
| Doc.Title | Document | – | Title | – |
| Doc.Number-of-pages | Document | – | Natural | – |
| Doc.Institution | Book Proceeding | Document | Organization | – |
| Doc.Conference | Proceeding | – | Conference | – |
| Thesis.University | Document | Thesis | University | – |

Table 2. Domains and ranges for the functions of the hierarchy in figure 1.

| DEFINITION | ORIGINAL DOMAIN | NEW DOMAIN | ORIGINAL RANGE | NEW RANGE |
|---|---|---|---|---|
| Doc.Author | Document | – | Author | – |

Table 3. Domains and ranges for the relations of the hierarchy in figure 1.

For example, in the domain of the bibliographic data[6], Table Two and Three summarize the domains and ranges of some functions and relations that have as a domain some classes in the hierarchy of the figure 1. In figure 1, 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. Taking these tables and hierarchy, we can say that:

1. The domain and range of the function Doc.Title (the title of a document) is well-defined.

2. The domain and range of the relation Doc.Author (the author of a document) is well-defined.

3. The domain of the function Thesis.University (the University of a Thesis) is over-specified.

4. The domain of the function Doc.Institution (the Institution that publish a document) is imprecise --any document has an institution that publishes it--.

To figure out if **the formal definition of a class is complete**, we check that:

a) The class is defined by a predicate defined by necessary and sufficient conditions. See example Two.

b) The generalizations/specializations of a given class exactly and precisely delimit the superclasses/subclasses of a given class in the real world. Errors appear when:

   b.1) The generalizations/specializations of a given class are imprecise, over-specified, or they include classes that are not applicable in the real world.

[6]Definitions in this paper don't correspond with the definitions of the Bibliographic-Data ontology [6].

b.2) Information about subclasses that are mutually disjoint is missing in their superclasses. For example, the classes DOCTORAL-THESIS and MASTER-THESIS should be mutually disjoint.

c) The set of properties attached to a given class represent the set of properties that the class owns in the real world. We recommend to perform verification of the classes in a hierarchy after performing verification of its functions and relations. Errors appear when:

c.1) Some properties are missed in the definition of a class. We discover missed properties by verifying 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, looking at table Two and Three, we detect some potential properties (Doc.Title, Doc.Number-of-pages, Doc.Author, Doc.Institution) of the class DOCUMENT by selecting those functions and relations whose domain is DOCUMENT. Since the class DOCUMENT owns the properties Doc.Title and Doc.Number-of-pages, we can say that the definition is incomplete. To make the definition complete, we introduce the missed function (Doc.Institution) and relation (Doc.Author) in the class DOCUMENT to guarantee that the class as well as its subclasses can have these properties defined. The following KIF sentences should be included in the formal definition of DOCUMENT:

```
:axiom-def (and (domain-of document Doc.Author)
                (domain-of document Doc.Institution))
```

c.2) 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 those that it is supposed to have in the real world. For example, in the class THESIS we constraint the values of the inherited properties doc.author and doc.publication-date when we use the following KIF sentences:

```
(has-one ?x doc.author)
(has-one ?x doc.publication-date)
```

c.3) Different classes have the same formal definition. We find equal formal definitions by looking classes that: (1) are classified under the same superclasses, (2) own the same set of properties, and (3) the properties have the same cardinality. Looking at figure One, we find that there are no semantics differences between the classes DOCTORAL-THESIS and MASTER-THESIS because they don't have any property that differentiates them. We solve the problem by defining a new function Thesis.Degree in the domain of THESIS and in the range of DEGREE. We differentiate the classes DOCTORAL-THESIS and MASTER-THESIS by including the KIF sentence (= thesis.degree Ph.D.) in the formal definition of DOCTORAL-THESIS, and (= thesis.degree M.S.) in the formal definition of MASTER-THESIS. The definitions of the function Thesis.Degree and the definition of the class DEGREE are given in example 4.

```
(define-function THESIS.DEGREE (?Thesis) :-> ?Degree
"The degree of a thesis work."
  :def (and (Thesis ?Thesis)
            (Degree ?Degree)))
```

```
(define-class DEGREE (?degree)
"The degree of a study"
:iff-def (member ?degree
        (setof B.S. M.S. Ph.D.)))
```

Example 4. Definitions of a new function and class

c.4) The class does not include properties that it cannot have in the real world, that is, we find out which properties the class cannot have in the real world and we include this information in the definition of the class. Specially they should be included if they may be inherited from its superclasses. Looking at figure One, we know that a PICTURE is a subclass of the class DOCUMENT, and that all documents can have pages. The KIF sentence (cannot-have ?x doc.number-of-pages) would forbid the definition of the property doc.number-of-pages in the instances of PICTURE.

### 2.3.3. Conciseness

Conciseness refers to if all the information gathered in the ontology is useful and precise. Conciseness doesn't 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 doesn't store any unnecessary or useless definition.

b) Explicit redundancies don't 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. Taking example Two, the definition of the months as instances in the ontology would make it redundant.

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

c.1) A property that can be inherited from a superclass is defined explicitly in any of its subclasses. For example, the inclusion of the KIF sentence (has-one ?x doc.title) in the class THESIS, would make it redundant because we can get this property form DOCUMENT by using inheritance.

c.2) A subclass-of relation could be inferred using other definitions. Given definitions in example Five, 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. A implicit redundancy is inferred

d) A definition is itself redundant. Given the definitions in example Five and Six, we can say that the definition of ORGANIZATION.NAME is redundant. It is explicitly said in example Six that the relation ORGANIZATION.NAME is a specialization of the relation AGENT.NAME. If the domain and range of the ORGANIZATION.NAME relation are specializations of the domain and range of the AGENT.NAME relation, then all the tuples in the ORGANIZATION.NAME relation 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. Sensitiveness relates 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, and (3) the ontology and its definitions are 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.

## 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 each 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 word 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 into 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.

## Conclusions

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 verify 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:

c.1) A formal definition of internal, metaphysical and inferred consistency, and examples that show how to deal with these new concepts.

c.2) An informal definition of completeness, and stereotype of errors that make relations, functions and classes incomplete.

c.3) An informal definition of conciseness, and kind of errors that make ontologies redundant.

c4) 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.

As main near future works, we identify the needed to build a tool to verify ontologies that have already been built, and to include extra checking in the tools for building new ontologies. After this, the creation of a methodology to verify ontologies to be integrated with the ontologies life cycle will be useful in order to detect mistakes as soon as possible.

**Acknowledgment**

This paper is supported by the *Ministerio de Educación y Ciencia* in Spain. Thanks to Richard Fikes for their comments and advices.

**REFERENCES**

[1] Genesereth, M.R.; Fikes, R.E. **Knowledge Interchange Format**. Version 3.0. Reference Manual. Report Logic-92-1. Computer Science Department. Stanford University. Stanford, CA, 94305. 1993.

[2] Gómez-Pérez, A. *From Knowledge Based Systems to Knowledge Sharing Technology: Evaluation and Assessment*. **Technical Report KSL 94-73**. Knowledge Systems Laboratory. Stanford University. CA. December 1994.

[3] Gómez-Pérez, *Some Ideas and Examples to Evaluate Ontologies*. **The Eleventh IEEE Conference on Artificial Intelligence for Applications**. 1995. Accepted paper.

[4] Gruber, T. *A Translation Approach to Portable Ontology Specifications*. **Knowledge Acquisition**. Vol 5. 1993. pp:199-220.

[5] Gruber, T. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. **Technical Report KSL 93-04**. Knowledge Systems Laboratory. Stanford University. CA. 94305. 1993.

[6] Gruber, T. *Bibliographic-Data Ontology*. This ontology is available at *hpp.stanfod.edu*, as **http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/bibliographic-data/index.html** 1994.

[7] Gruninger, M.; Fox, M.S. *The role of Competency Questions in Enterprise Engineering*. **IFIP WG5.7 Workshop on Benchmarking. Theory and Practice**. Trondheim, Norway, 1994.

[8] Guha, R.V.; Lenat, D. *Enabling Agents to work Together*. **Communications of the ACM.** July 1994. Vol. 37. N0 7. PP: 127-142.

[9] Neches, R.; Fikes, R.; Finin, T.; Gruber, T.; Patil, R.; Senator, T.; Swartout, W.R. *Enabling Technology for Knowledge Sharing*. **AI Magazine**. Winter 1991. PP: 36-56.