Some Ideas and Examples to Evaluate Ontologies

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

The lack of methods for evaluating ontologies in laboratories can be an obstacle to their use in companies. This paper presents a set of emerging ideas in evaluation of ontologies useful for: (1) ontologies developers in the lab, as a foundation from which to perform technical evaluations; (2) end users of ontologies in companies, as a point of departure in the search for the best ontology for their systems; and (3) future research, as a basis upon which to perform progressive and disciplined investigations in this area. After briefly exploring some general questions such as: why, what, when, how and where to evaluate; who evaluates; and, what to evaluate against, we focus on the definition of a set of criteria useful in the evaluation process. Finally, we use some of these criteria in the evaluation of the Bibliographic-Data [5] ontology.

1. Introduction

Several years ago, the idea of a *knowledge factory* emerged when several research groups [3, 7, 9] put their efforts in building ontologies in several application domains. As libraries of definitions that can be used for different purposes in different domains, ontologies allow the reuse and sharing of knowledge and reasoning methods among agents. From the point of view of knowledge *reusability*, ontologies avoid the need to build a Knowledge Base (KB) from scratch by allowing Knowledge-Based Systems (KBS) developers to assemble reusable components [10]. From the point of view of knowledge sharability, ontologies can be used by software agents [3, 6, 8, 10] that are in operation. We can say that ontologies are the platforms that enable the sharing and reuse of knowledge by establishing common vocabularies and semantic interpretations of terms.

While ontologies may provide for reusability, sharability or both, the evaluation of their definitions and software environment is critical to the success of the final applications that reuse and share these definitions. If wrong definitions from the ontology coexist with specific knowledge formalized in the KB, the KBS may make poor or wrong conclusions. If the ontology definitions and software environment have not been sufficiently evaluated, communication between software agents may not succeed because ontologies provide the channel through which queries are asserted.

Until now, the number of ontologies developed has not been large and their practical use in final and real applications in businesses is small. The lack of methods for evaluating ontologies in laboratories can be an obstacle to their use in companies. The purpose of the present work is twofold. The first is to describe a set of initial and general ideas that guide the evaluation of ontologies. The second is to apply empirically some of these ideas in the evaluation of the Bibliographic-Data ontology [5]. The following sections describe the outcome of this research. In Section Two, we briefly describe a set of emerging ideas in the evaluation of Knowledge Sharing Technology (KST). Section Three shows some criteria used in performing a technical evaluation of the ontologies in the academic or industrial laboratory. Section Four discusses a set of ideas related to the assessment that the end user must perform before using this technology in companies. Finally, Section Five discusses three kind of examples in the technical evaluation of the Bibliographic-Data ontology [5] written in KIF [2].

2. The Primary Ideas

This section briefly answers several questions in the evaluation of KST.

What does evaluation and assessment mean? Evaluation means to judge technically the features of KST, and assessment refers to the usability and utility of KST in companies.

What can be evaluated? We can evaluate any of the following items: any intermediate or final definition, a set of definitions, documentation, and software environment.

Why evaluate? To guarantee to end users (both KE and software agent developers) the correctness and completeness of the ontologies' definitions, documentation, and software when they leave the laboratory and are used in organizations or by software agents in operation.

What to evaluate against? Evaluation of the ontologies should be performed against a frame of reference. It can be a set of competency questions [7], requirements, and the real world. Evaluation of the software environment should be performed against its requirements.

When to evaluate? Evaluation is an iterative process that should be performed during each phase and between phases of the KST life cycle. The goal is to detect as soon as possible wrong, incomplete, or missed definitions.

How to evaluate? Avoid "ad hoc" techniques and use standard techniques.

Who evaluates KST? The development team, other development teams, and end users evaluate different features of the ontologies' definitions and software. While the development teams evaluate technical properties of the definitions, end users evaluate their actual utility within a given organization or by other software agents.

Where to evaluate? Evaluation can be performed anywhere, including at the lab (technical evaluation) or at the end user location (assessment).

3. Technical Evaluation

The Knowledge Factory idea allows different people (whether related or not to the environment development team) not only to reuse definitions but to build their own. For this reason, the development team must perform a global technical evaluation that ensures well-defined properties in: (1) the definitions of the ontology, (2) the software environment used to build, reuse and share definitions, and (3) the documentation. Since that software evaluation can be performed by using software engineering evaluation techniques, this paper only covers evaluation of the definitions and evaluation of the documentation.

Evaluation of the definitions[1]. Evaluation of the definitions is a technical evaluation that must be performed at the lab during the whole ontology life cycle. The goal is to detect the absence of some well-defined properties in the definitions. The evaluation steps include:

Check the structure or architecture of the ontology. The goal is to figure out if the definitions are built following the design criteria of the environment in which they are included. Ontologies built in the Ontolingua environment [3] should satisfy the five design criteria given by Gruber [4].

Check the syntax of the definitions, to detect as soon as possible syntactically incorrect structure and/or wrong keywords in definitions without looking into their meaning. The environment should provide a syntactic analyzer that automatically checks for the presence/absence of the natural language documentation, wrong keywords in formal definitions, structure of the formal definitions, absence of loops between definitions, and so on.

Check the content in the definitions, to detect what the ontology defines, doesn't define, or defines incorrectly, and what can be inferred, cannot be inferred, or may be inferred incorrectly. The purpose is to identify lack of knowledge and mistakes in the definitions. It deals with the *problem of the three Cs*: Consistency, Completeness and Conciseness. The following definitions are independent of the languages used to write ontologies' definitions.

Consistency refers to the incapability of getting contradictory conclusions simultaneously from valid input data. An ontology is semantically consistent if and only if its definitions are semantically consistent. A given definition is semantically consistent if and only if (1) the individual definition is consistent, that means that the meanings of the formal as well as the informal definitions are consistent with the real world, and consistent with each other, and (2) they are not contradictory sentences that may be inferred by using other definitions and axioms that may or may not belong to the same ontology.

Completeness refers to the extension, degree, amount or coverage to which the information in a user-independent ontology covers the information of the real world. The completeness of a definition depends on the level of granularity agreed to in the whole ontology. We can say that a definition is complete if nothing has been forgotten. That is to say, if all that is supposed to be in the definition is in the definition or can be inferred from the axioms. Completeness of the definitions deals with completeness of their formal and informal definition. To figure out if a *formal definition* is complete, first, we seek to determine whether the definition meets the structural criteria for a complete definition (a predicate defined by necessary and sufficient conditions [4]). Second, we determine whether the domain and range of the relations and functions are exactly and precisely delimited. Third, we detect whether the generalization/specialization of a given class exactly and precisely represents the superclasses/subclasses of a given class in the real world. Finally, we look for a complete set of attributes in each class definition. An *informal definition*

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

written in natural language is complete if it expresses the same knowledge intended in the formal definition.

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 the definitions. We can guarantee that a given definition in an ontology is concise if we avoid redundancies in its formal as well as informal definitions. Second, explicit redundancies don't exist among definitions. Third, redundancies cannot be derived using axioms attached to other definitions. Finally, the set of properties in the definition of a class are precisely and exactly defined. The natural language explanation of the definitions and examples are not considered redundant knowledge of the formal definitions.

Evaluation of the documentation. The role of documentation in KST is a key factor in its dissemination. The goals of this evaluation are to guarantee that certain documents are developed and that they evolve in step with the definitions and software. Documentation includes: the natural language string in each definition, general information about the ontology, its basic ontological commitments, a summary of its definitions, studied cases in its evaluation, 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. In this case, semantic incoherence, context mistakes, and morphological mistakes can appear easily in the indexes of the natural language documentation. A study performed in Ontolingua ontologies reveals that the majority of semantic and context errors can be easily avoided if the ontology writer writes the words to be indexed using given conventions (i.e.., using uppercase for all the words, and/or using hyphenated strings of words).

4. Assessment

A technically well-evaluated ontology will not guarantee the absence of errors in the integration of its definitions with the KBS definitions or in the communication among agents, but it will make the process easier. This section provides a set of ideas that knowledge engineers and software agent developers should be familiar with before using the ontologies definitions.

Ontologies used in KBS. Developing and implementing ontologies for reusing requires integrating their definitions into the entire life cycle of the KBS. For some authors, ontologies are useful in the KBS conceptualization phase [3, 10], or in the knowledge acquisition [3] phase, or in the verification and validation [11] phases. In the Ontolingua environment [3], definitions can be translated into a variety of target representation languages, making ontologies definition useful in the KBS implementation phase. Since ontologies are useful in the entire life cycle of an ES, the knowledge sharing community should create methods and techniques that aid in the integration of the ontologies definitions in the expert systems life cycle by developing technology that: (1) supports, checks and makes effective this integration, and (2) allows the integration of different solutions provided by different families of ontologies.

When the KE decides to use reusable components, the KE is looking for some definitions that simplify the development process for the new system. So, *KE assessment* is focused on judging the understanding, usability, abstraction, quality, well-defined properties and portability of the reusable definitions, the software environment capabilities, and methodologies that make possible this approach. Before using this technology, the KE should evaluate the answer the following questions: Does the ontologies environment provide methods and tools that help the KE to design a KBS using reusable components? Does the environment include a tutorial or case studies that reduce the cost (time and money) required for KBS developers to learn and assimilate this new technology? Has knowledge stored in ontologies reduced the bottleneck in the knowledge acquisition phase? Does the ontology software environment provide translators that transform the ontology definitions into the target language used in knowledge acquisition, conceptualization, formalization, implementation and evaluation tools? Are the definitions complete, consistent, and concise? How much knowledge is lost when a given definition from the ontology is translated into the target language of the application? Is each definition self-explanatory?, and questions related to the evaluation of ontologies (the following questions should be asked if new ontologies definitions or axioms will replace old ones in the KBS, or if new definitions will be included in the KB): How do new definitions modify the KB and the behavior of the system? If the KBS was already evaluated, what kind of evaluation should be performed when new definitions are added?, and are the new axioms consistent with the old axioms?

The KE will at some point report the chosen definitions to the programming team. Because the programming team integrates the definitions from the ontologies with the domain-specific knowledge of the application, programming team tries answer the following questions: How trustworthy is the translation of the definitions in the chosen target language? And is it possible to integrate the definitions in the KB without making significant modifications?

Ontologies used by software agents. From the point of view of knowledge *sharability*, an ontology [3] is defined as the vocabulary with which queries and assertions are exchanged among agents that are in operation. According to Gruber and Olsen [6], the ontology vocabulary defines the *ontological commitments* among agents that are agreements to use the shared vocabulary in a coherent and consistent manner. Guha and Lenat [8] view ontologies as foundational knowledge shared by agents to enable them to communicate their specialized knowledge.

When definitions are used in the dialog among agents, inferences are always performed by the agent making the query and by the agents giving the answers. Since ontologies definitions are the channels through which agents request and answer queries, the knowledge inferred by the agent answering the query should be consistent with the whole set of definitions made in the ontology. It is possible that different agents could provide different answers for a given query. In this case, each answer might or might not be contradictory with the other answers, and each individual answer might or might not be wrong. Software agent answers, by themselves, can be wrong due to an ill-defined or ambiguous query formulation, or because the agent itself is wrongly implemented, is incomplete, inconsistent, or ambiguous. In this last case, the solution would be to enable the communication of the wrong agent with other agents through the ontologies definitions. However, there are some situations in which well-evaluated software agents would provide ambiguous, or contradictory answers because the query was not well-formulated. In this case, the software agent should tolerate and recover from ambiguities, omissions, and errors in human or agent requests [1]. The evaluation of software agents is beyond the scope of this paper. However, software agent developers should consider evaluation of this new technology and its relations with ontologies.

5. Cases Studied in the Evaluation

In this section, we describe part of the technical evaluation of the Bibliographic-Data ontology [5]. Three kinds of examples are analyzed. Case study number 1 evaluates a definition. Case study number 2 evaluates a hierarchy of definitions. Finally, case number 3 evaluates the domain and range of the relations and functions.

5.1. Case 1: Evaluating Definitions

In this example we analyze the completeness, conciseness, consistency and coherence of the original definition MONTH-NAME in the Bibliographic-Data ontology [5].

Def.1: (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)))

Def.2: (define-instance JANUARY (month-name))
• • •

The analysis of the *completeness* in definition 1 reveals that: (1) although the formal definition meets the structural criteria for a complete definition and all the months are enumerated, the formal definition is incomplete because there are no references to the months' properties, and (2) the informal definition is incomplete because the months are not enumerated.

Although the formal and informal definitions of MONTH-NAME have different semantics, the formal as well as the informal definitions are individually consistent and consistent with each other. In this example, the natural language documentation complements the formal definition, providing extra knowledge about *how* the definition deals with its instances. Examples of real inconsistencies would be the definition of an unknown month in the formal or informal definition, or in both.

Definitions 1 and 2 also provide a clear case of explicit redundancy. The class MONTH-NAME is extensionally-defined by enumerating the set of months. The months are also defined in definition 2 as instances in the ontology. To avoid redundancies, definition 2 should be deleted.

Finally, note that the definition of some concepts about time inside the Bibliographic-Data ontology alters its coherence. Definitions about time should be set out in an ontology about Time, and the Bibliographic-Data should include this ontology.

5.2. Case 2: Evaluating a Hierarchy

In this example, we evaluate: (a) the individual consistency, completeness and conciseness of the definitions in a hierarchy and (b) the consistency, completeness and conciseness among them. The aim is to evaluate the completeness and conciseness of a given definition using other definitions and axioms.

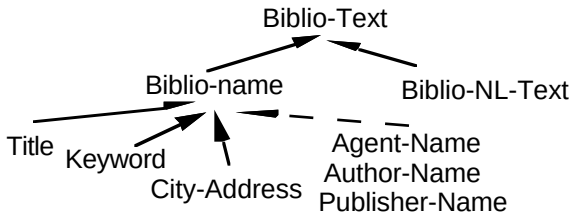

Figure 1. BIBLIO-TEXT Hierarchy.

BIBLIO-TEXT is the most general class of strings in the Bibliographic-Data ontology. The original BIBLIO-TEXT hierarchy explicitly sets out the subclasses linked by a plain line in Figure 1. Dashed lines mean that the subclass definition does not explicitly include the link with its superclass, but it can be inferred using other definitions. This hierarchy and the following definitions taken from the

Bibliographic-Data ontology will be the point of departure in this evaluation.

Def.3 (define-class **BIBLIO-TEXT** (?string)
"The most general class of undifferentiated text objects."
:def (and (biblio-thing ?string)
(string ?string))

Def.4: (define-class **BIBLIO-NAME** (?string)
"A name of something in the bibliographic-data ontology."
:def (biblio-text ?string))

Def.5: (define-class **TITLE** (?x)
"A title is a string naming a publication, a document, or something analogous."
:def (biblio-name?string))

Def.6: (define-class **KEYWORD** (?keyword)
"A keyword is a name used as an index."
:def (biblio-name ?keyword))

Def.7: (define-class **CITY-ADDRESS** (?name)
"A city-address is a string that identifies a city somewhere in the world."
:def (biblio-name ?name))

Def.8 (**define-class AUTHOR-NAME** (?name)
"A string that is used as the author.name of some author. Often databases of author names are kept separately from databases of people or documents."
:axiom-def (exact-range author.name author-name))

Before evaluating relationships among definitions, we can say about each individual definition the following. First, definition 3 is inconsistent. BIBLIO-TEXT is a subclass of STRING, but not of BIBLIO-THING. A new definition for BIBLIO-TEXT would delete the sentence (biblio-thing ?string). Second, definition 6 is not precise. To increase the precision, the informal definition should say explicitly that "A keyword is a *string* used as an index". If we had written "A keyword is a *number* used as an index.", an inconsistency would appear because ?keyword is a string in the formal definition of BIBLIO-TEXT (Def. 3). Third, in Figure 1, the classes AGENT-NAME, AUTHOR-NAME and PUBLISHER-NAME are not connected explicitly with the BIBLIO-TEXT hierarchy. However, this knowledge is implicit in the ontology and could be inferred from other definitions. The inclusion of the KIF sentence :constraint (biblio-name ?name) that connects explicitly each one of the three subclasses with the class BIBLIO-NAME makes the three definitions redundant. For example, in the AUTHOR-NAME definition (Def. 8), it can be inferred from the definition of EXACT-RANGE[2] and AUTHOR.NAME (see Def. 14 in section 5.3) that the class AUTHOR-NAME is a SUBCLASS-OF[3] BIBLIO-NAME. Since AUTHOR-NAME is the EXACT-RANGE of the AUTHOR.NAME function, and a RANGE of AUTHOR.NAME is BIBLIO-NAME, and since the EXACT-RANGE of a BINARY-RELATION is a SUBCLASS-OF any of ranges, then it follows that AUTHOR-NAME is a SUBCLASS-OF BIBLIO-NAME. Consequently, the definition of AUTHOR-NAME is concise and the inclusion of the constraint in the definition makes it redundant Finally, if for agents, authors and publishers the Bibliographic-Data ontology defines their names, why doesn't the class UNIVERSITY? In order to maintain the orthogonality of the definitions, the class UNIVERSITY-NAME should be included and defined as a subclass of BIBLIO-NAME in the Bibliographic-Data ontology.

Def.9 (define-class **UNIVERSITY-NAME** (?name)
"A name of some university"
:axiom-def (exact-range university.name
university-name))

The following evaluates the relationships between definitions. First, if we delete the informal definitions of the classes TITLE, KEYWORD and CITY-ADDRESS, there is no semantic difference between the three formal definitions. The three classes contain the same information: they are subclasses of the class BIBLIO-NAME. So, these three formal definitions are incomplete, because they miss the specific properties that define and differentiate each class from the other. In this example, the semantic differences between the three classes are given by the natural language definition and by its name, and not by the meaning of the KIF sentences. If we do not introduce the properties into the previous formal definitions, we wonder if a formal and incomplete definition that is equal to another should be removed from the ontology. The answer is no! For a KBS that would reuse the definitions, they are not useful because they do not provide specific knowledge of the classes. However, for an agent that uses the ontology to communicate with other agents, these definitions allow them to use the terms TITLE, KEYWORD and CITY-ADDRESS to exchange titles, keywords, and city addresses that are strings.

Second, incomplete definitions also appear if any general design criteria are missing in any definitions. The analysis of the class/subclass definitions reveals an interesting design criteria in Bibliographic-Data: in order to assure that the subclasses of a class are mutually disjoint, one should use the relation SUBCLASS-PARTITION[4] to define the subclasses inside the class. However, this important criterion that increases the completeness of the entire ontology is sometimes not followed because its use can make the ontology inconsistent. For example, if we added the following sentence in definition 4:

[2] A relation maps elements of a domain into 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.The EXACT-RANGE is the class whose instances are exactly those that appear in the last item of any tuple in the relation.

[3] Class C is a subclass of class P iff every instance of C is an instance of P.

[4] A subclass partition of a class C is a set of subclasses of C that are mutually disjoint.

:axiom-def (subclass-partition BIBLIO-NAME
(setof title keyword city-address))

we would introduce an inconsistency because BIBLIO-NAME deals with strings that may be equal or may not. Consequently, if we don't introduce the previous :axiom-def sentence, the original BIBLIO-NAME definition is consistent.

5.3. Case 3: Evaluating Functions and Relations

In this example we evaluate the precision in the domain and range of the relations and functions. Figure 2 summarizes the domain (the hierarchy of AGENTs) and the range (the class BIBLIO-NAME) of the relations PENNAME and AUTHOR.NAME, and the domain and range of the functions AGENT.NAME, ORGANIZATION.NAME and PUBLISHER.NAME. Given the definitions 11 and 12, we can increase the precision in the range of definition 12 by replacing the term BIBLIO-NAME by AGENT-NAME. Since we cannot increase precision without creating a loop, the definition of AGENT.NAME is not precise, even though it is syntactically well-defined.

Def.10 : (define-class **AGENT** (?x)
:def (and (biblio-thing ?x)
(has-one ?x agent.name))
:axiom-def (subclass-partition AGENT
(setof person organization)))

Def.11: (define-class **AGENT-NAME** (?name)
"A string that is the name of some agent."
:axiom-def (exact-range Agent.Name Agent-Name))

Def.12:(define-function **AGENT.NAME** (?agent):-> ?name
"Function from an agent to the name by which it goes. If an agent has more than one complete name (not parts of the name, such as first and last name), then the agent.name is the name used to identify that agent in the shared world ..."
:def (and (agent ?agent)
(*biblio-name* ?name)))

The main problems appear in the definitions attached to authors: AUTHOR (Def. 13), AUTHOR.NAME (Def. 14), PENNAME (Def. 15) and AUTHOR-NAME (Def. 8).

Def.13: (define-class **AUTHOR** (?x)
"An author is an agent who writes things. An author must have a name, which is its real name as an agent. The name as author may or may not be the agent's name, but usually is."
:def (and (agent ?x)
(has-one ?x agent.name))
:default-constraints (same-values ?x
author.name agent.name))

Def.14: ((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))))

Def.15 ((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))

Definition 13 says that an AUTHOR is an AGENT (Def. 10) that has a name that is the AGENT.NAME (Def. 12) of the AGENT. Since the name of the AUTHOR may or may not be the agent name (the range of AGENT.NAME is BIBLIO-NAME), and the author is always an AGENT, and since each AGENT must always have a name and only one name in the range of BIBLIO-NAME, then it follows that the AUTHOR always has a unique name in the range of BIBLIO-NAME. So, the AUTHOR definition is consistent but redundant because we prove twice that the author has one agent name. To make definition 13 non redundant, the sentence (has-one ?x agent.name) has to be deleted.

Two relations called AUTHOR.NAME and PENNAME are defined in the domain of AUTHORs. Three kinds of problems are found between these definitions. First, we cannot increase the precision of the range in definition 14 by replacing BIBLIO-NAME with AUTHOR-NAME because it would create a loop between AUTHOR.NAME (Def. 14) and AUTHOR-NAME (Def. 8) definitions. Second, since the name of the author may or may not be unique, the following discussion analyzes the relationships between AUTHOR.NAME and PENNAME. Suppose that the author Asun Gómez is named by the string "AG" and we wonder whether this string is her author name or her penname. When we try to prove that "AG" is the author name of the agent Asun Gómez (AUTHOR.NAME Asun Gómez "AG"), the first two KIF statements in definition 14 tell us that we have to prove that Asun Gómez is an AUTHOR and that "AG" is a string of BIBLIO-NAME. Having proved these two sentences, we must prove that "AG" is her agent name or her penname. If (AGENT.NAME Asun Gómez "AG") is true, it should not matter whether "AG" is used to refer to her penname because "AG" is a valid author name for her, and (AUTHOR.NAME Asun Gómez "AG") is true. In the other case, if we cannot prove that (AGENT Asun Gómez "AG") is true, we must prove that (PENNAME Asun Gómez "AG") is true for (AUTHOR.NAME Asun Gómez "AG") to be true. However, since PENNAME is defined as a specialization of AUTHOR.NAME, we must prove again the truth value of the AUTHOR.NAME. This loop between definitions 14 and 15 prevents us from reaching any conclusion.

Finally, the definition of PENNAME is not well-defined because this relation is not a specialization of the relation AUTHOR.NAME. There are strings in BIBLIO-NAME that are not AUTHOR-NAMEs and they may be used as a PENNAME.

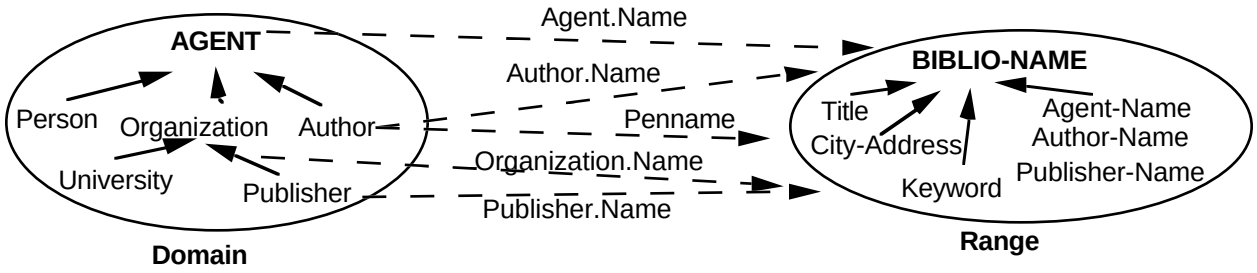

Figure 2. Domain and Range of functions and relations.

To solve the three problems, we don't change the formal definition of the relation AUTHOR.NAME. We only add in the informal definition of definition 14 the following: "*... A particular author.name of an author must be either the author's real name (i.e., her agent.name), or any other string*". We also perform a new formal definition of PENNAME that says that the penname of an author is a string that is not its name as an agent:

```
:def (and (author ?author)
          (biblio-name ?name)
          (not (agent.name ?author ?name))))
```

Conclusions

As ontologies are used for several different purposes, diverse kinds of evaluations are required. Evaluation of their definitions and evaluation of their development environment are critical in the process of transferring knowledge sharing technology from laboratories to companies. This paper described how different people have different concerns in the evaluation process, and how to carry out this evaluation through the example evaluation of the Bibliographic-Data ontology.

We found that ontologies evaluation includes technical *evaluation* by the development team, and *assessment* by the end user. Although the core of the technical evaluation is the evaluation of the definitions, the ontology development team must not forget evaluation of the software environment and documentation. The most important features to consider in the evaluation of the definitions are their structure, content, syntax, and a set of semantic properties that guarantee the coherence, completeness, consistency and conciseness of the definitions.

Acknowledgment

This paper is supported by the *Ministerio de Educación y Ciencia* in Spain. Special thanks to Tom Gruber for some rewarding conversations, and to Bob Engelmore for their insights and advice.

References

[1] Etzioni, O.; Weld, D. *A Softbot-Based Interface to the Internet.* **Communications of the ACM**. July, Vol. 37. No. 7. 1994. PP: 72-79.

[2] 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.

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

[4] 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.

[5] Gruber, T. 1994. This ontology is available through anonymous ftp at *hpp.stanfod.edu*, as **/a/htw/Ontolingua/all-ontologies/biblio/bibliographic-data.lisp~6~**.

[6] Gruber, T; Olsen, G. *An Ontology for Engineering Mathematics*. **Technical Report KSL 94-18.** Knowledge Systems Laboratory. Stanford University. CA. 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] Lenat, D.B., Guha, R.V.; Pittman, K.; Pratt, K.; Shepherd, M. *Cyc: Toward Programs with Common Sense*. **Communications of the ACM**. Vol. 33, 1990. PP: 30-49.

[10] 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.

[11] Plaza, E.; *KBS Validation: From tools to Methodology*. **IEEE Expert**. Vol. 8, No. 3, June. 1993, PP: 45-47.