Embedding Knowledge in Web Documents

Philippe Martin and Peter Eklund
Griffith University, School of Information Technology, PMB 50 Gold Coast MC, QLD 9726 Australia
Tel: +61 7 5594 8271; Fax: +61 7 5594 8066; Email: {philippe.martin,p.eklund}@gu.edu.au

Abstract
The paper argues for the use of general and intuitive knowledge representation languages (and simpler notational variants, e.g. subsets of natural languages) for indexing the content of Web documents and representing knowledge within them. We believe that these languages have advantages over metadata languages based on the Extensible Mark-up Language (XML). Indeed, the retrieval of precise information is better supported by languages designed to represent semantic content and support logical inference, and the readability of such a language eases its exploitation, presentation and direct insertion within a document (thus also avoiding information duplication). We advocate the use of Conceptual Graphs and simpler notational variants that enhance knowledge readability. To further ease the representation process, we propose techniques allowing users to leave some knowledge terms undeclared. We also show how lexical, structural and knowledge-based techniques may be combined to retrieve or generate knowledge or Web documents. To support and guide the knowledge modeling approach, we present a top-level ontology of 400 concept and relation types. We have implemented these features in a Web-accessible tool named WebKB (http://meganesia.int.gu.edu.au/~phmartin/WebKB/), and show examples to illustrate them.

Keywords. Knowledge Modeling, Precision-oriented Information Retrieval, Knowledge-based Indexation and Annotation, Data and Metadata Management, Ontology.

Table of contents

1 Introduction

Document indexation techniques, such as those used in large-scale Web search engines, support the retrieval of documents that might contain some parts related to the query. Alternative approaches involve natural language parsing techniques to extract a precise semantic network from the content of documents. Such a network enables an inference engine to give a precise answer to a query. However, despite substantial progress, e.g. DR-LINK[1], CYC[2], Web documents cannot in general be "understood" using natural language processing techniques.

Precision-oriented information retrieval is performed by Web robots such as Harvest[3], W3QS[4], WebSQL[5] and WebLog[6]. Such tools perform string-matching searches (e.g. with regular expressions) and structure-matching searches (e.g. on tags, link names and link paths) in documents. These tools may compose the retrieved information to answer queries and generate documents. However, for precise information to be retrieved in this way, the documents (or Web-accessible databases) must be rigorously structured and this structure known to the users making the queries.

[1] http://www.textwise.com/drlink.html
[2] http://www.cyc.com/applications.html#nl
[3] http://harvest.transarc.com/
[4] http://www.cs.technion.ac.il/~konop/w3qs.html
[5] http://www.cs.toronto.edu/~websql/
[6] http://www.cs.concordia.ca/~special/bibdb/weblog.html

One way to improve information retrieval precision and ease of representation is to use a knowledge representation language. Many "metadata" languages are currently being developed to allow people to index Web information resources by knowledge representations (logical statements) and subsequently store them in Web documents. Most of these languages are built above XML[7], e.g. RDF[8] and OML[9].

The choice of XML as an underlying format ensures that standard XML tools will be usable to exchange and parse these metadata languages. However, like XML, metadata languages built above it are also verbose and therefore difficult to use without specialized editors (this point will be illustrated in Figure 3). Such editors do not eliminate the need for people to use a language for representing knowledge (except in application-dependent editors that simply allow predefined "frames" to be filled). Consequently, as noted by the authors of Ontobroker[10] [1], with XML-based languages information has to be written in two versions, one for machines and another for humans. Additionally, standard XML tools are of little help in managing these languages since specialized editors, analyzers and inference engines are required. To reduce information redundancy, Ontobroker provides a notation for embedding attribute-value pairs within an HTML hyperlink tag. These tags may be used by the document's author to delimit an element. In this way, each element may be implicitly referenced in the knowledge statement within the tag enclosing the element. When a final version of RDF is recommended by the Word Wide Web Consortium[11], a wrapper can be added to Ontobroker for automatically generating RDF definitions from Ontobroker metadata, thus making them accessible to a broader community.

We favor the Ontobroker approach. However, we believe the Ontobroker metadata language has the following drawbacks that prevent it being used for precise knowledge modeling or rapid information indexing: (i) it is general but basic and hard to read (it is a notation for embedding attribute-value pairs within HTML hyperlink tags), and (ii) the terms used in the knowledge statements cannot be defined in the same document. Furthermore, the Ontobroker metadata language does not complement HTML with better indexation features.

Our solution for easing the representation of knowledge is first to propose *a set of intuitive, complementary and combinable languages or commands* that allow users to represent and index any Web-accessible information at the levels of precision they desire. More precisely, this implies an expressive formal model (the user should not be restricted by the language) and various notations for it. Any formalism equivalent to full first-order logic allowing the use of contexts, such as KIF[12], would be appropriate. For search or reasoning purposes, the users' knowledge statements may be translated into less expressive but more efficient languages, e.g. Loom[13]. For our knowledge annotation and exploitation tool, WebKB, we have chosen the Conceptual Graphs (CGs)[14] formalism, first because it has a graphical notation and a linear notation, both concise and easily comprehensible, and second because we can reuse CG inference engines that exploit subsumption relations defined between formal terms for calculating specialization relations between graphs — and therefore between a query graph and facts in a knowledge base. Hence, queries may also be made at various levels of granularity. We have added operators to these CG engines, e.g. a maximal join on given CGs, and complemented the CG linear notation by other less expressive but more readable linear notations using a formalized English, HTML structures and indented text.

Even with such languages usable with any text editor, representing knowledge may still be considered too tedious by the user if all the terms used in the knowledge statements must be declared and organized. In WebKB, *the user may choose not to declare all the terms s/he uses*. The use of semi-formal statements is at the expense of knowledge precision and accessibility but allows rapid expression and incremental refinement of knowledge. When forewarned by a special command ("no decl"), WebKB accepts CGs that include undeclared terms. We show below how this imprecision may partially be compensated by exploiting a natural language ontology and constraints in the application ontology.

Top-level ontologies provide constraints and building blocks for knowledge representation. For instance, the Knowledge Sharing Effort public library[15] provides many precise ontologies. WebKB proposes a more terminologically oriented ontology to ease rapid and simple knowledge representation. It includes 400 concept and relation types, and was created by merging other top-level ontologies used in knowledge acquisition, knowledge representation, and cooperation-oriented hypertext tools. For the sake of brevity, we do not detail this ontology in this paper but provide some of its components and uses. It is accessible and browsable at the WebKB site, and more details on its construction may be found in [2] and [3].

The lexical, structural and knowledge-based approaches are *complementary* for information retrieval and exploitation. In WebKB, these approaches are *combined* in the following way: lexical and structural query commands working on Web-

[7] http://www.w3.org/XML/
[8] http://www.w3.org/RDF/
[9] http://wave.eecs.wsu.edu/CKRMI/OML.html
[10] http://www.aifb.uni-karlsruhe.de/WBS/broker/
[11] http://www.w3.org/
[12] http://logic.stanford.edu/kif/kif.html
[13] http://www.isi.edu/isd/LOOM/LOOM-HOME.html
[14] http://concept.cs.uah.edu/CG/Standard.html
[15] http://www-ksl.stanford.edu/knowledge-sharing/

accessible documents are proposed and may be combined with the knowledge query/assertion commands via a simple Unix shell-like script language. An important feature is that all these commands may be *embedded within documents*. This permits command scripts and eases document generation. For instance, a WebKB query command or script may be associated with an HTML hyperlink, thus enabling the generation of context-dependent documents when the link is activated.

Finally, a genuine sharing of knowledge implies a *shared repository* (virtual repository if it is composed of distributed systems) where procedures control the integration of knowledge from the various users. We have not yet implemented this shared repository in WebKB but the expected procedures and work in progress are reported at the WebKB site.

We first present architectural choices for tools like WebKB, then detail its *language features* and introduce the *ontology* it proposes.

2 Architecture

Our survey of Web-based tools for Knowledge Acquisition (KA), Information Retrieval (IR) and Cooperation reveals that these tools face similar design issues that lead to the implementation of a subset of the same basic elements. Figure 1 shows these basic elements.

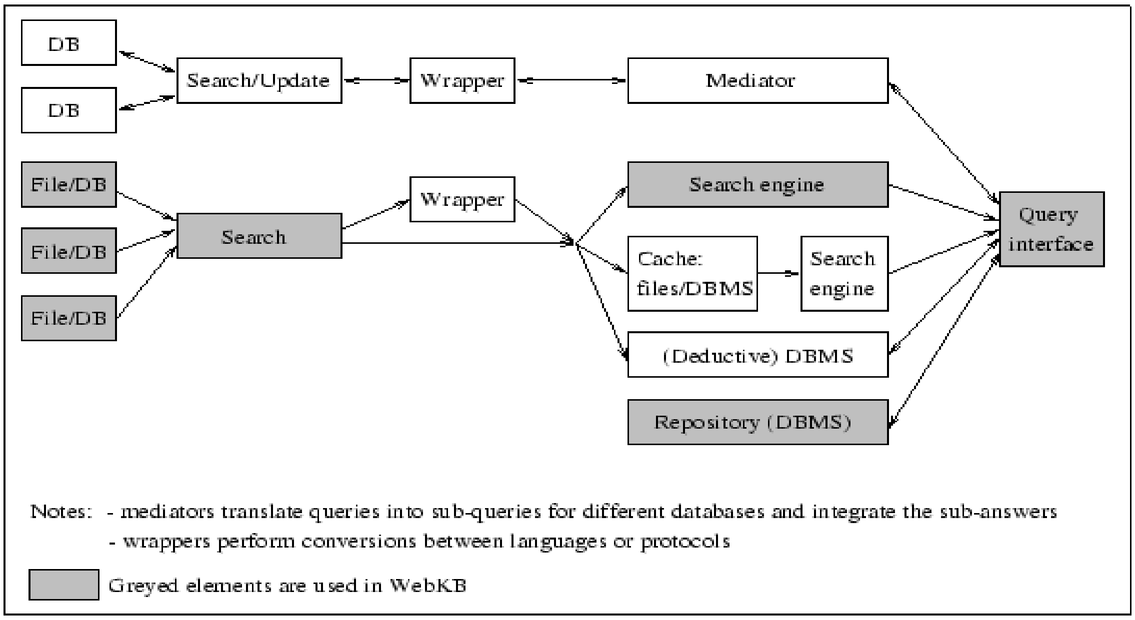

Figure 1: Generalized architecture elements of Web-based IR/KA/Groupware systems.

Some architectural choices have to be made for KA/IR/Cooperation Web-based tools (and thus also for WebKB which is aimed to support these three tasks). More precisely, these tools may (1) integrate distributed systems, (2) search and exploit the content of distributed information sources (plain files or databases), or (3) allow users to store and exploit information in a repository.

In the first case, tools such as AlephWeb[16], Hermes[17], Infomaster[18] and TSIMMIS[19] unify heterogeneous distributed information systems and use a "mediator" that translates user queries into sub-queries for the different systems and then integrates sub-answers. The mediator exploits "wrappers" and content descriptions of information sources to perform the conversion between languages or protocols (cf. Figure 1). The information sources must conform to a predefined structure to allow a wrapper to extract structured information.

In the second case, structured information, metadata or knowledge statements are searched in different Web-accessible files or databases, and possibly translated into the same language. A search may be initiated and directed by a user query (as in WebSQL or WebLog), or done for collecting and caching data in order to efficiently respond to queries later (as in Ontobroker). Thus, the search engine and the storage system can be integrated, as in a database management system.

[16] http://www.pangea.org/alephweb.aleph/paper.html
[17] http://www.cs.umd.edu/projects/hermes/
[18] http://infomaster.stanford.edu/infomaster-info.html
[19] http://www-db.stanford.edu/tsimmis/tsimmis.html

In the third case, a repository stores information. Controls may be initiated when users enter information, and relationships between information from different users automatically created. Integrating distributed systems may be seen as creating a virtual repository, and each distributed system may itself be a repository, as for example in AlephWeb.

According to these distinctions, WebKB has three components:

- a text/knowledge *search engine* that can *generate* new knowledge and documents by assembling operators;
- text/knowledge *query interfaces* written in HTML and Javascript (knowledge editors are also proposed for helping users build knowledge assertions or queries); and
- ontologies stored in Web-accessible documents.

At present, the WebKB processor can search information or knowledge in Web-accessible documents but does not support the construction and access of a knowledge repository by multiple users. This processor is a C/C++ program that is Web-accessible via the Common Gateway Interface (CGI)[20]. It exploits two CG workbenches, CoGITo[21] and Peirce[22] that are both memory bound. To handle a large repository, the WebKB processor needs to be extended to exploit a deductive database. The usable document/knowledge assertion/query/management languages will not change but will operate on the repository in addition to Web documents. The ontologies currently exploited by WebKB, plus the natural language ontology WordNet[23] (90,000 concept types connected to 120,000 words) will initialize the repository.

The WebKB processor will remain Web-accessible by a CGI interface. In this way, it is accessible both from simple form-based interfaces (such as the WebKB user interfaces — easily adaptable by users for their particular needs) or by other programs. More program-oriented interfaces, such as Corba[24] or OKBC[25] may be added in the future. OKBC would enable knowledge exchange with other knowledge representation systems (KRSs), e.g. Loom or Ontolingua, and enable the repository to be graphically browsed and edited by the Generic Knowledge Base Editor[26]. Finally, wrappers for languages such as RDF or KIF might also be added as standards and interfaces to them emerge.

Figure 2 shows the WebKB menu and the "Knowledge-based Information Retrieval/Handling Tool".

3 Language features

We now give some examples of the language features we propose and have implemented in WebKB. More examples and the grammar of these languages may be found at the WebKB site. The commands of these languages may be combined with commands of simple Unix shell-like scripting language, e.g. if, for, pipe and set.

3.1 Lexical and Structural Query Languages

Because WebKB proposes knowledge representation and query commands, and a script language, we have not felt the need to give it a lexical and structural query language as precise as those of Harvest, WebSQL and WebLog. Instead, we have implemented some Unix-like text processing commands for exploiting Web-accessible documents or databases and generating other documents, e.g. cat, grep, fgrep, diff, head, tail, awk, cd, pwd, wc and echo. We added the hyperlink path exploring command "accessibleDocFrom". This command lists the documents directly and indirectly accessible from given documents within a maximal number of hyperlinks. For example, the following command lists the HTML documents accessible from http://www.foo.bar/foo.html (maximum 2 levels) and that include the string "nowledge" in their HTML source code.

```
accessibleDocFrom -maxlevel 2 -HTMLonly http://www.foo.bar/foo.html | grep nowledge
```

[20] http://www.w3.org/CGI/
[21] http://seine.inapg.inra.fr/~ollivier/Publis.html
[22] http://www.cs.rmit.edu.au/~ged/publications.html
[23] http://www.cogsci.princeton.edu/~wn/
[24] http://www.whatis.com:80/corba.htm
[25] http://www.ai.sri.com/~okbc/
[26] http://www.ai.sri.com/~gkb/

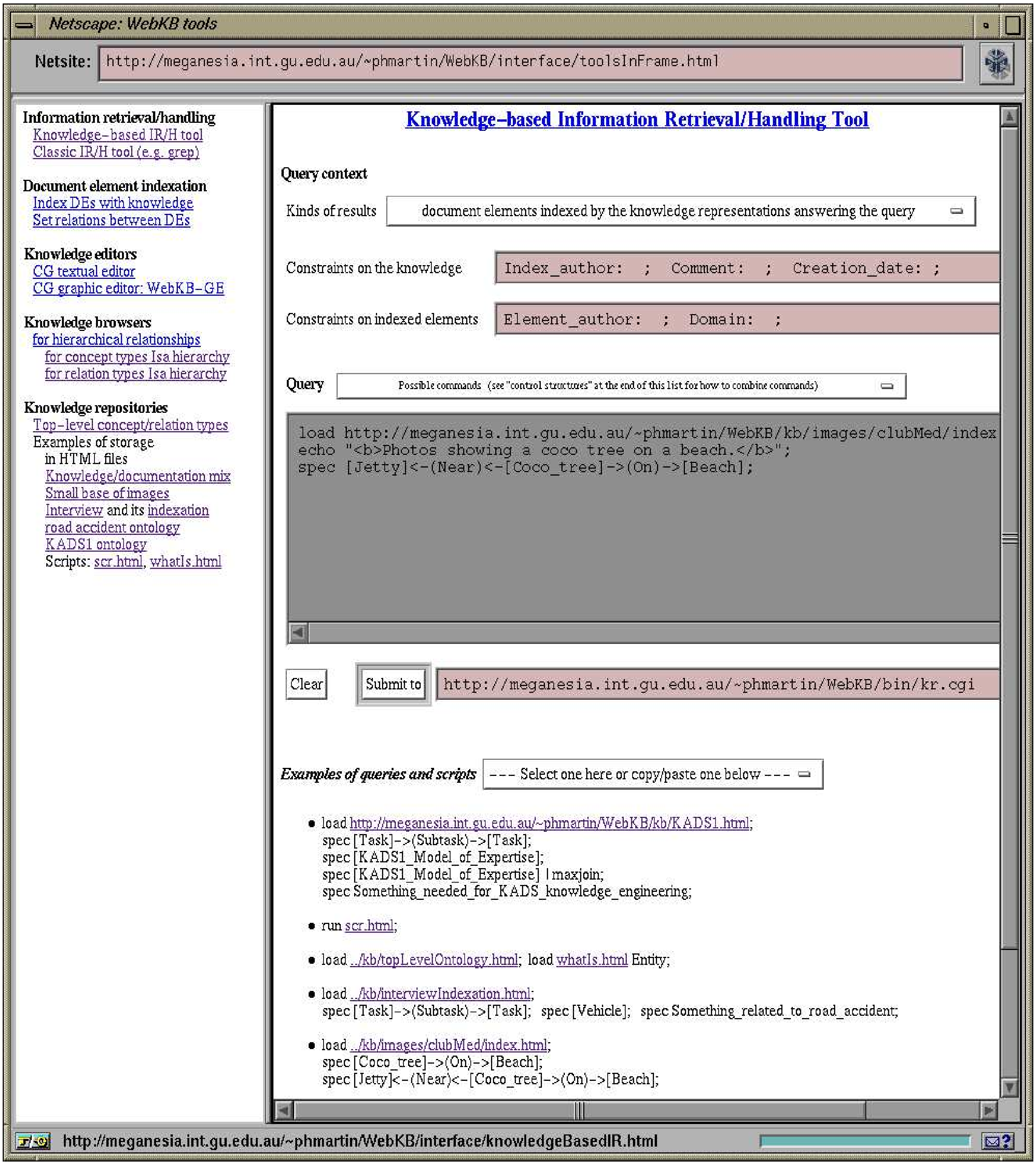

Figure 2: The WebKB tool menu and knowledge-based Information Retrieval/Handling Tool. *The example query shows how a document containing CGs (indexing images) is loaded into the WebKB processor and then how the command "spec" (which looks for specializations of a CG) can be used to retrieve CGs and the images they index. According to the value selected in the "kinds of results" option (cf. top right of the figure), the images, but not the knowledge statements, will be presented. A similar query and its results is shown in the figures 6 and 8.*

3.2 Knowledge Representation

3.2.1 Knowledge Representation Languages vs XML-based Metadata Languages

XML is intended as a machine-readable rather than human-readable language because it is mainly meant to be generated and read by machines not people. XML-based metadata languages inherit this poor readability and most of them (e.g. RDF) do not specify how to represent logical operators or quantifiers. As an alternative, WebKB proposes to use expressive but intuitive knowledge representation languages to represent (or index) information in documents and mix knowledge statements with other textual elements (e.g. sentences, sections or references to images). To allow this, the knowledge (or commands exploiting it) must be enclosed whithin the HTML tags "`<KR>`" and "`</KR>`" or the strings "$(" and ")$". The knowledge representation language used in each chunk must be specified at its beginning, e.g.: "`<KR language="CG">`". (Lexical/structural/procedural commands may be used whichever language is specified). Thus, there is no need to separate knowledge from its documentation nor duplicate it in an external knowledge base.

At present,WebKB only exploits the CG formalism. However, the exploitation of wrappers (e.g. KIF to CGs) or other inference engines would allow WebKB to accept other knowledge representation languages. To compare the alternatives, Figure 3 shows how a simple sentence may currently be represented in WebKB, how it could be represented in KIF, and what its RDF representation is. The sentence is: "John believes that Mary has a cousin who has the same age as her".

```
<KR language="CG">
load "http://www.bar.com/topLevelOntology";   //Import this ontology
Age < Property;                //Declare Age as a subtype of Property
Cousin(Person,Person) {Relation type Cousin};

[Person:"John"]<-(Believer)<-[Descr: [Person:"Mary"]- { (Chrc)->[Age: *a];
                                                         (Cousin)->[Person]->(Chrc)->[*a];
                                                       } ];
</KR>
```

```
<KR language="KIF">
load "http://www.bar.com/topLevelOntology";    //Import this ontology
(Define-Ontology Example (Slot-Constraint-Sugar topLevelOntology))
(Define-Class Age (?X)  :Def (Property ?X))
(Define-Relation Cousin(?s ?p) :Def (And (Person ?s) (Person ?p)))

(Exists ((?j Person))
   (And (Name ?j John) (Believer ?j '(Exists ((?m Person) (?p Person) (?a Age))
                                        (And (Name ?m Mary) (Chrc ?m ?a)
                                             (Cousin ?m ?p) (Chrc ?p ?a)
                                        ))
                       ))) </KR>
```

```
         <!-- RDF notation (with allowed abbreviations); this file is named "example" -->
<RDF xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
     xmlns:t="http://www.bar.com/topLevelOntology">
  <Class ID="Age"><subClassOf resource="t#Property"/></Class>
  <PropertyType ID="Cousin"><comment>Relation type Chrc (Characteristic)</comment>
                            <range  resource="t#Person"/>
                            <domain resource="t#Person"/></PropertyType> </RDF>

<RDF xmlns="http://www.w3.org/TR/WD-rdf-syntax#"   xmlns:x="http://www.bar.com/example"
     xmlns:t="http://www.bar.com/topLevelOntology">
  <Description aboutEach="#Statement_01"> <t#Believer>John</t#Believer> </Description>

  <t#Person bagID="Statement_01"><t#Name>Mary</t#Name>
                                 <x#Chrc><x#Age ID="age"></x#Age></x#Chrc>
                                 <x#Cousin><t#Person><x#Chrc resource="#age"/></t#Cousin>
  </t#Person> </RDF>
```

Figure 3: Comparing knowledge representation with KIF, CGs and RDF.

The CG representation (top) seems simpler than the others. The semantic network structure of CGs (i.e. concepts connected by relations) has three advantages: (i) it restricts the formulation of knowledge without compromising expressivity and this tends to ease knowledge comparison from a computational viewpoint; (ii) it encourages the users to express relations between concepts (as opposed, for instances, to languages where "slots" of frames or objects can be used); (iii) it permits a better visualization of relations between concepts.

3.2.2 Less Expressive but More Intuitive Notations

Even if CGs seem relatively intuitive, they are not readable by everyone. In restricted cases, simpler notations may be preferable. For instance, Figure 4 shows notations that are accepted by WebKB as equivalent to the following CG:

```
TC for KADS1_conceptual_model(x) are   //note: TC means "Typical Conditions"
[KADS_conceptual_model:*x]-
{ (Part)->[Model_of_problem_solving_expertise];
  (Part)->[Model_of_communication_expertise];
  (Part)->[Model_of_cooperation_expertise];
  (Input)<-[Knowledge_design]->(Output)->[Knowledge_base_system];
}
```

```
    /* Structured text (":" ends the name of a "typical" relation,
                       "=>" of a "necessary" relation, "<=" of a sufficient relation) */
KADS1 conceptual model.
    Part: Model of problem solving expertise,
          Model of communication expertise,
          Model of cooperation expertise.
    Input of: Knowledge design (Output: Knowledge base system).
```

```
    /* Text structured with HTML tags (and same conventions for relations) */
<dl><dt>KADS1 conceptual model
    <dd>Part: <ul><li>Model of problem solving expertise
                  <li>Model of communication expertise
                  <li>Model of cooperation expertise
              </ul>
    <dd>Input of: Knowledge design (Output: Knowledge base system)
</dl>
```

```
    /* Formalized english */
Typically, a KADS1 conceptual model has for part a model of problem solving expertise,
a model of communication expertise and a model of cooperation expertise.
Typically, knowledge design has for input a KADS1 conceptual model and for output a
knowledge base system.
```

Figure 4: Complementary notations for simple knowledge statements.

3.2.3 Allowing Undeclared Terms in Knowledge Statements

The user may not want to take the time to declare and order many of the terms s/he uses when representing knowledge. This may for example be the case when a user indexes sentences from various documents for private knowledge organisation purposes.

To permit this, and still allow the system to perform some minimal semantic checks and knowledge organisation, we propose the casual user represent knowledge with basic declared relation types and leave undeclared the terms used as concept types. This method has the following rationales:

- If knowledge statements are made from concepts linked by basic relations, i.e. if the complexity is manifest within concept types rather than in relation types, only a limited set of relation types are necessary for an application. WebKB already proposes a top-level ontology of 200 basic relation types[27] [2] [3] collecting common thematic, mathematical, spatial, temporal, rhetorical and argumentative relations types.
- WebKB can use relation signatures to give suitable types to the undeclared terms used as concept types. For instance, in the top-level ontology proposed by WebKB, the relation types *Input, Output, Agent, Method, SubProcess* and *Purpose* are all defined to have a concept of type *Process* as the first argument. Hence, in the previous example, WebKB can infer that *Knowledge_design* must be a subtype of *Process*.
- We have merged the natural language ontology WordNet[28] (120,000 words linked to 90,000 concept types) into our top-level ontology (cf. [2] [3]). When the WebKB shared repository is implemented and initialized with these ontologies, it will be possible for WebKB to semi-automatically relate the undeclared terms used as concept types to precise concept types in the repository, thanks to links between words and concept types and to constraints imposed by the

[27] http://meganesia.int.gu.edu.au/~phmartin/WebKB/kb/topLevelOntology.html
[28] http://www.cogsci.princeton.edu/~wn/

relation signatures. Consider for example, the following CG where the terms *Cat* and *Table* have not been declared: `[Cat]->(On)->[Table]`. In WordNet, the word *cat* has 5 meanings (feline, gossiper, X-ray, beat and vomit) and the word *table*, 5 meanings (array, furniture, tableland, food and postpone). In the WebKB ontology, the relation type *On* connects a concept of type *Spatial_entity* to another concept of the same type. Thus, WebKB can infer that "beat" and "vomit" are not the intended meanings for *Cat*, and "array" and "postpone" are not the intended meanings for *Table*. To further identify the intended meanings, WebKB could prompt the following questions to the user: "does *Cat* refer to feline, gossiper, X-ray or something else?" and "does *Table* refer to furniture, tableland, food or something else?".

- Knowledge statements are more readily comparable if they follow the same conventions. The convention of using basic relations is thus important. (The alternative convention — using primitive concepts and complex relations — would be much harder to follow). Consider for example the sentence "Mary is 20 years old". Following our conventions it is better to use the concept type *Age*, e.g. `[Person:"Mary"]->(Chrc)->[Age:@20]`,
rather than the relation type *Age*, e.g. `[Person:"Mary"]->(Age)->[Integer: 20]`,
unless this relation type has been predefined by a user: [29]

```
relation Age (x,y) is  [Age]- { (Chrc)->[Living_entity:*x];
                                (Measure)->[Integer:*y];
                              }
```

By default, WebKB enforces the use of declared terms in the CG linear notation but permits undeclared terms (for types and instances) in the other (simpler) notations (cf. Figure 4). The commands "decl" and "no decl" overide this default mode and an exclamation mark before a term explicitly tells the system that the term was deliberately left undeclared. Quoted sentences may also be used: they are understood by WebKB as individual concepts of type "Description".

Another facility of the WebKB parser is that, like HTML browsers, it ignores HTML tags (except definition list tags) in knowledge statements. However, when these statements are displayed in response to a query, they are displayed using the exact form given by the user, including HTML tags. Thus, the user may combine HTML or XML features with knowledge statements, e.g. s/he may put some types in italics or make them the source of hypertext links.

3.3 Indexing any Document Element using Knowledge

3.3.1 General Cases

We call a Document Element (DE) any textual/HTML data, e.g. a sentence, a section, a reference to an image or to an entire document. This definition excludes binary data but includes textual knowledge statements. WebKB allows users to index any DE of a Web-accessible document (or later of our repository) by knowledge statements, or connect DEs by relations. Figure 5 shows an example of each case.

```
$(Indexation
   (Context: Language: CG;
             Ontology: http://www.bar.com/topLevelOntology.html;
             Repr_author: phmartin; Creation_date: Mon Sep 14 02:32:21 1998;
              Indexed_doc: http://www.bar.com/example.html; )
   (DE: {2nd occurence}  the red damaged vehicle )
   (Repr: [Color: red]<-(Color)<-[Vehicle]->(Attr)->[Damaged] )
)$

$(DEconnection
   (Context: Language: CG;
             Ontology: http://www.bar.com/topLevelOntology.html;
             Repr_author: phmartin; Creation_date: Mon Sep 14 02:53:36 1998;)
   (DE: {Document: http://www.bar.com/example.html} )
   (Relation: Summary)
   (DE: {Document: http//www.bar.com/example.html} {section title: Abstract})
)$
```

Figure 5: A language for knowledge indexing or connecting any Web-accessible document element.

[29] This solution implies that the inference engine expands the relation type definition when comparing graphs. Few CG engines can perform type expansion.

XML provides more ways to isolate and reference DEs than HTML. Since WebKB exploits the capacities of Web-browsers, XML mechanisms may be used by the WebKB users. However, XML does not help users to annotate others' documents since DEs cannot be referenced if they have not been explicitly delimited by the documents' authors. Therefore, the WebKB facility of referring to a DE by specifying its content and its occurrence number will still be useful.

3.3.2 A Simple Example

The above indexation notations allow the statements and the indexed DEs to be in different documents. Thus, any user may index any element of a document on the Web. Figure 2 presents a general interface for knowledge-based queries and shows how a document containing knowledge must be loaded in the WebKB processor before being queried.

WebKB also allows *the author of a document* to index an image by a knowledge statement directly stored in the "alt" field of the HTML "img" tag used to specify the image. We use this special case of indexation to present a simple illustration of WebKB's features. This example, shown in Figure 6, is a good synthesis but in no way representative of the general use of WebKB — it is not representative because it mixes the indexed source data (in this case, a collection of images), their indexation, and a customized interface to query them, in a single document. Figure 7 shows a part of this document that illustrates the indexation. The result of the query shown in Figure 6 is displayed in Figure 8.

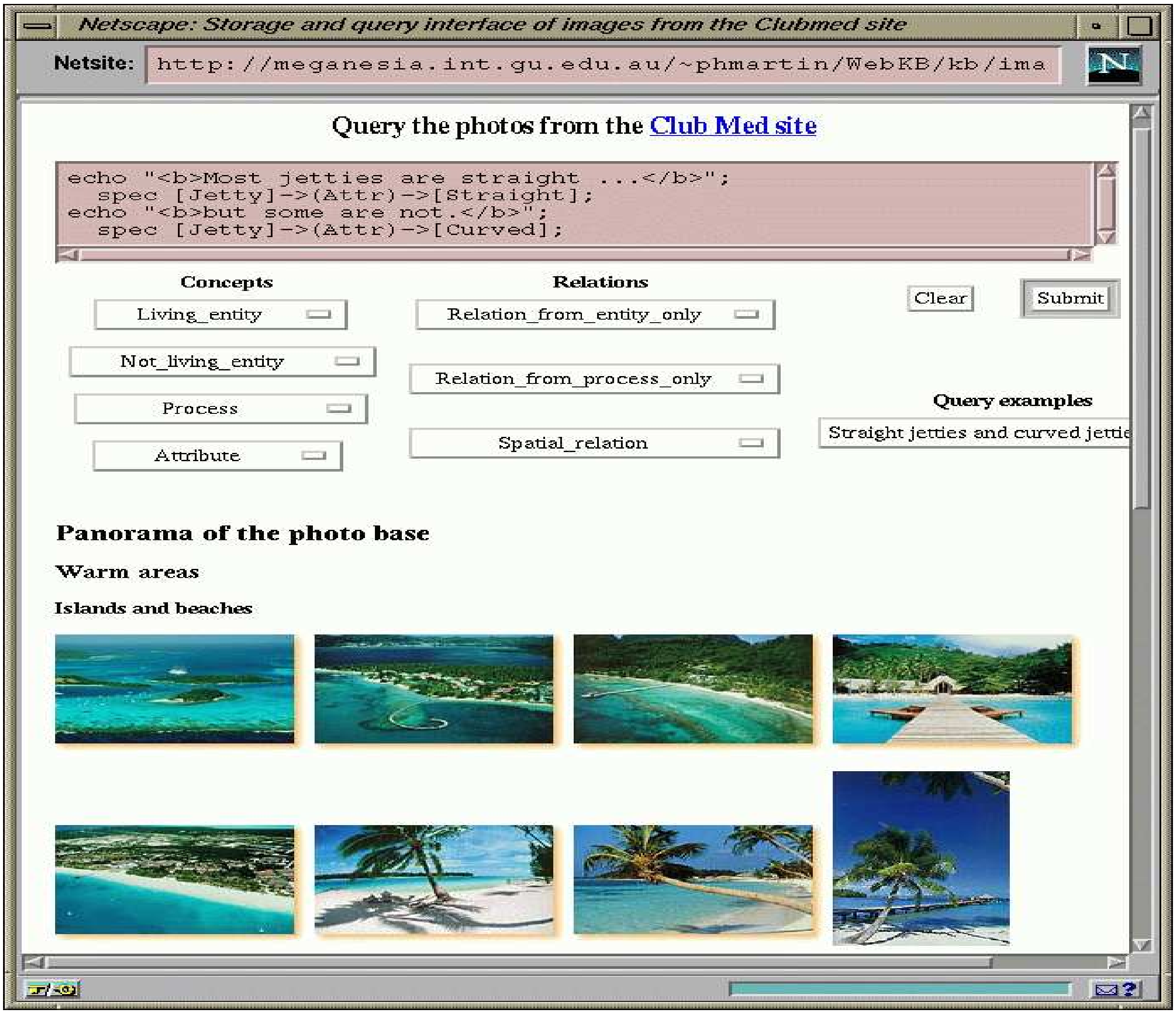

Figure 6: Images, indexations and a customized query interface within a same document *(the example query shows how the command "spec" can be used to retrieve images indexed by CGs. See the results in Figure 8).*

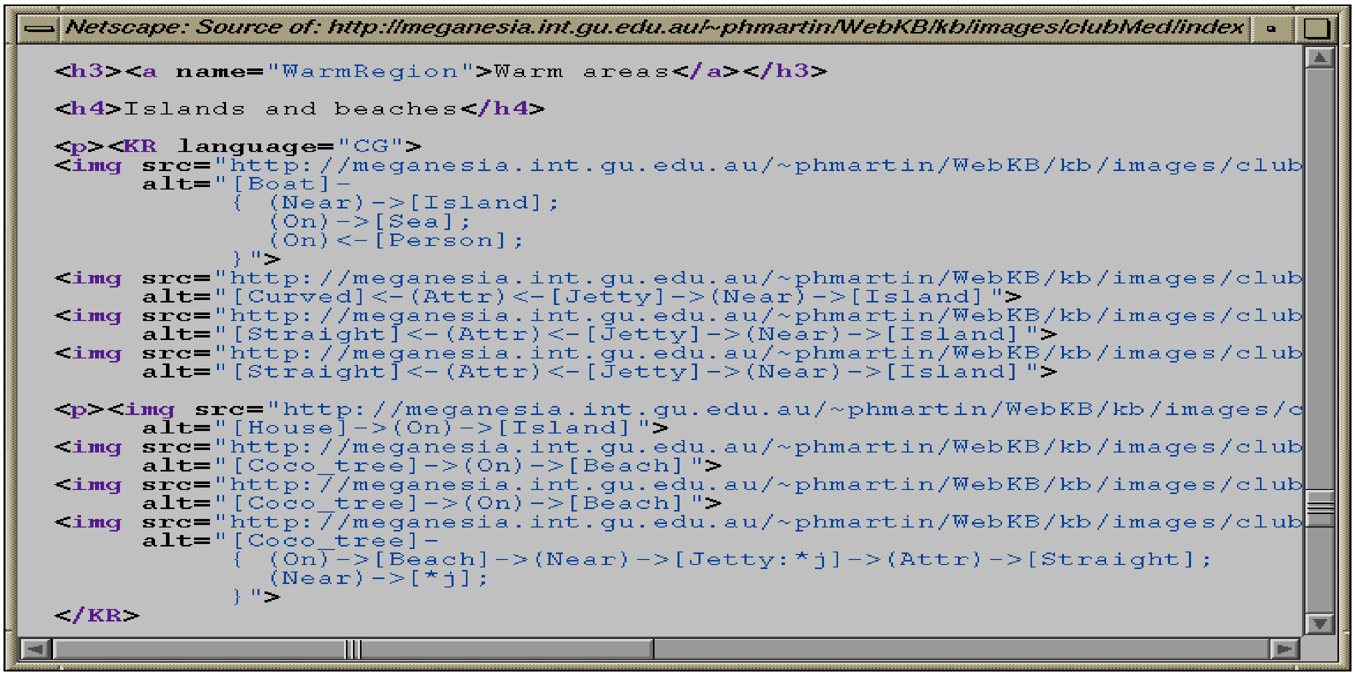

Figure 7: The HTML source of the indexation of the images shown in Figure 6.

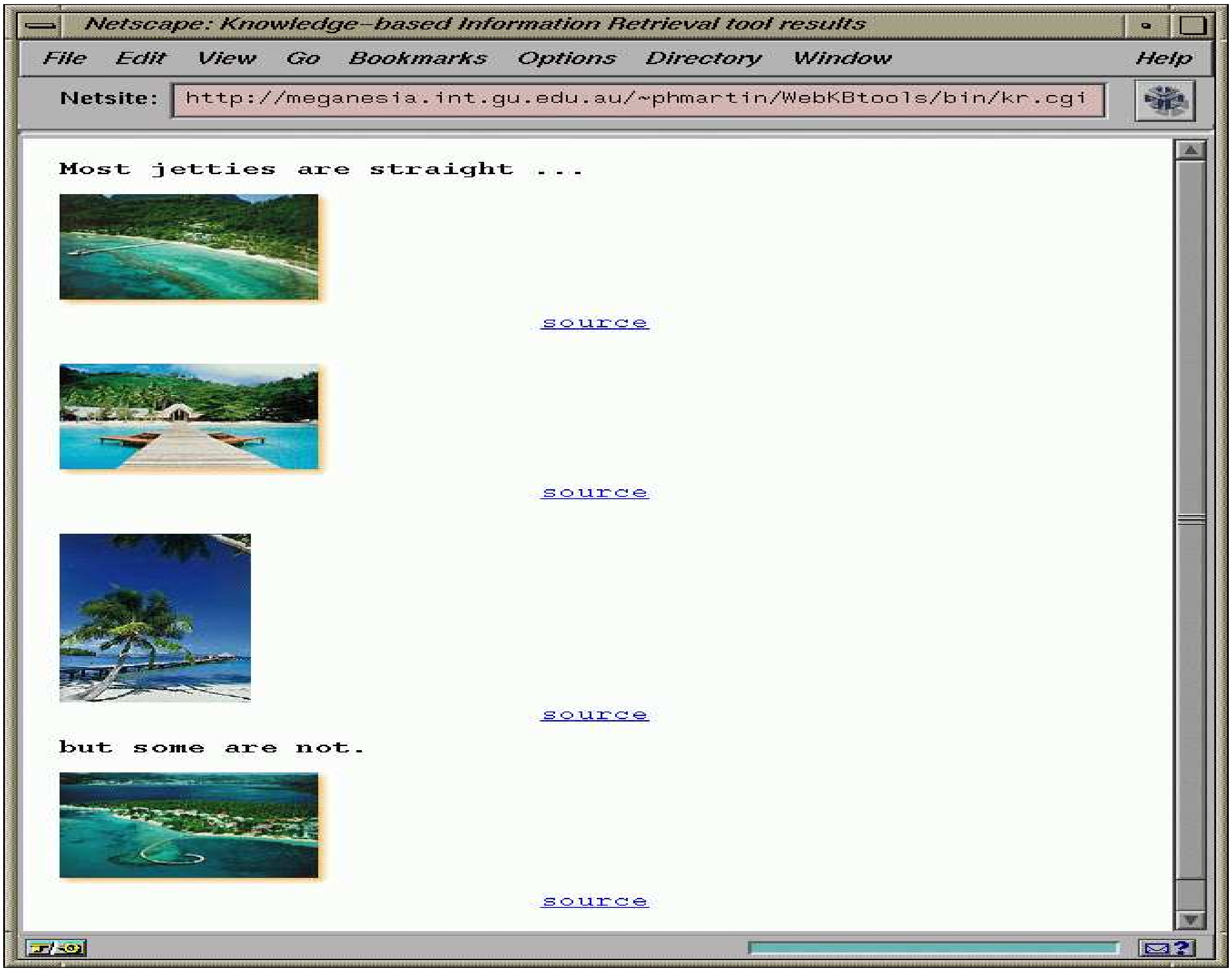

Figure 8: The document generated in response to the query in Figure 6.

3.4 Knowledge Query Commands

WebKB has commands for displaying specializations or generalizations of a concept or relation type or an entire CG in a knowledge base. At present, queries for CG specializations only retrieve connected CGs: the processor cannot retrieve paths between concepts specified in a query. If a retrieved CG indexes a document element, it may be presented instead of the CG (Figure 8 gives an example). In both cases, hypertext links are generated to reach the source of each answer presented in its original document — a copy of this original document will be presented by WebKB in order to instruct the Web browser to display and highlight the selected answer in its source document. What follows is an example of such an interaction, assuming that http://www.bar.com/example.html is the file where the indexation in Figure 5 has been stored, and *Something* is the most general concept type.

```
> load http://www.bar.com/example.html
> spec [Something]->(Color)->[Color: red]
  [Color: red]<-(Color)<-[Vehicle]->(Attr)->[Damaged];
```

Source

```
> use Repr   //display represented DEs
> spec [Something]->(Color)->[Color: red]
  the red damaged vehicle
```

Source

Queries for specializations give the user some freedom in the way s/he expresses queries: searches may be done at a general level and subsequently refined according to the results. However, the exact names of types must be known. To improve this situation, WebKB allows the user to give only a substring of a type in a query CG if s/he prefixed this substring by the character %. WebKB generates the actual request(s) by replacing the substring by the manually/automatically declared types that include that substring. Replacements that violate the constraints imposed by relation signatures or individual types are discarded. Then, each remaining request is displayed and executed. For example, `spec [%thing]` will trigger the generation and execution of `spec [Something]`.

Knowledge query commands may be combined with the script language to generate complex documents, perform consistency tests on the knowledge base, or solve problems procedurally. The WebKB site provides many examples of queries and scripts. For example, one script solves the Sisyphus-I room allocation problem[30]. The reader is invited to test these examples[31].

Here is an example of a script showing that the procedural language frees us to add some special operators to our query language, such as the modal operators "few" and "most", since they are easily definable by the user.

```
spec [Something] | nbArguments | set nbCGs;
spec [Cat] | nbArguments | set nbCGsAboutCat;
set nbCGsdiv2 'expr $nbCGs / 2';
if ($nbCGsAboutCat > $nbCGsdiv2)
{ echo "Most CGs of the base are about cats"; }
```

3.5 Knowledge Generation Commands

The only type of knowledge generation commands in WebKB are commands that join CGs. Various kinds of joins may be defined but WebKB only proposes joins which, given a set of CGs, create a new CG specializing each of the source CGs. Though the result is inserted in the CG base, it may not represent anything true for the user, but provides a device for accelerating knowledge representation. For instance, in WebKB, CGs related to a type may be collected and automatically merged via a command such as this one: `spec [TypeX] | maxjoin`. The result may then serve as a basis for the user to create a type definition for TypeX.

The following is a concrete example for the maximal join command.

```
> maxjoin  [Cat]->(On)->[Mat]  [Cat:Tom]->(Near)->[Table]
  [Cat:Tom]- { (On)->[Mat];
               (Near)->[Table];
             }
```

[30] http://meganesia.int.gu.edu.au/~phmartin/WebKB/kb/sisyphus1.html

[31] http://meganesia.int.gu.edu.au/~phmartin/WebKB/, or if this server is down, http://www.int.gu.edu.au/~phmartin/WebKB/

3.6 Embedding Commands in Documents

We have seen how knowledge statements may be embedded in documents and how adequate notations such as structured text or formalized English may ease the process of merging knowledge and its documentation.

It is also interesting to embed knowledge-based and string-based commands inside documents so that parts of these documents are automatically generated by collecting information or knowledge stored elsewhere. Alternatively, within HTML documents, Javascript may be used for associating a query to an hypertext link in such a way that the query is sent to the WebKB query processor when the link is activated (then, as for any other query, the WebKB processor generates an HTML document that includes the results; if the query has been sent from a Web-browser, this document is automatically displayed). In the hypertext literature this technique is known as *dynamic linking* and the generated document is called a *dynamic document* or a *virtual document* [4]. This idea has many applications, e.g. adapting the content of a document to an individual user. Metadata languages do not currently include knowledge queries and therefore do not support dynamic linking.

Scripts may also be used for generating entire documents, e.g. for reporting results of tests on knowledge. In this case, constant strings may be generated using "print" commands.

4 A Top-Level Ontology

The top-level ontology proposed by WebKB was designed to guide and accelerate the creation of application ontologies and the building of knowledge statements. This ontology gathers about 200 common basic relation types (e.g. thematic, mathematical, spatial, temporal, rhetoric and argumentative relation types) and classifies them in a subsumption hierarchy according to their meaning and the kinds of concepts they connect. Figure 9 shows the upper levels of this hierarchy displayed with the WebKB hierarchy browser. As an example, rhetorical relation types come from the Generalized Upper Model[32] and the argumentation relation types come from the cooperation oriented hypertext system AAA[5]. These relations specialize the type *BinaryRel_from_a_description* since they connect descriptions. A synthesis of the most useful of relations between descriptions is proposed in the menu of the WebKB interface for connecting document elements by conceptual relations.

The ontology also structures about 200 general concept types needed for the signatures of the basic relation types, for setting minimal constraints on terminological knowledge, and for representing some useful knowledge acquisition notions such as KADS[33] elements and generic task models. Figure 10 shows the upper levels of the concept types hierarchy. These levels provide a synthesis of classic elementary distinctions that allow one to organise the top-level ontology (and the ontologies that specialize it) into partitions (i.e. in exclusive[34] sets of types): *Situation* (an aspect of a real or imaginary world) / *Entity* (things involved in a situation), *Process* (situation considered as changing by the user who represents it) / *State* (situation considered as static), *Temporal entity* (a point or extent in space) / *Spatial entity* (a point or extent in time) / *Information entity* (partition of the distinctions *Description* / *Description container* / *Property* / *Property measure*). We have not included the distinctions *Abstract thing* / *Concrete thing* and *Collection* / *Elemental thing* in the upper levels in order to keep them easy to visualize (it is also difficult to classify natural language concepts according to these distinctions). However, the type *Collection* and sub-partitions for these types have been included, and it seems that the usual sub-distinctions of abstract things have been represented via other distinctions (e.g. what we call temporal entities and information entities are often considered as abstract entities).

These ontological distinctions may appear obvious but we have often noted that even when these distinctions are clearly stated and used, users make semantic errors when they represent knowledge. Consider for example, two concept types named *Representation* and *Observation*. They could refer to a state, a process, the result of this process (which could either be a description or the thing(s) described) or a document used for storing this result. The creator of such types would probably not make the exact category explicit if s/he was not induced into that by an ontology such as ours. The relation signatures and the exclusive links between our top-level types allow a system like WebKB to do some semantic checks when types are used or specialized by their creators or other users. For example, if *Observation* refers to a state, the user will not be allowed to use the relation type *Agent* (e.g. instead of the relation types *Consequence* or *Successor*) to connect a concept of type *Observation* to another concept. For the same reasons, our top-level distinctions make the ontology they structure more reusable.

Using our top-level ontology, we have structured and complemented the upper levels of the WordNet lexical database. The 90,000 WordNet concept types are subsumed by our top-level ontology and may be accessed from this top-level ontology via a browser (cf. [2] and [3] for such a browser). The constraints in the top-level ontology are convenient to check the use of the WordNet types or even sometimes to understand what they refer to. Other ontologies could be structured and complemented in the same way.

[32] http://www.darmstadt.gmd.de/publish/komet/gen-um/newUM.html

[33] http://www.swi.psy.uva.nl/projects/CommonKADS/home.html

[34] Exclusive types may not have common subtypes.

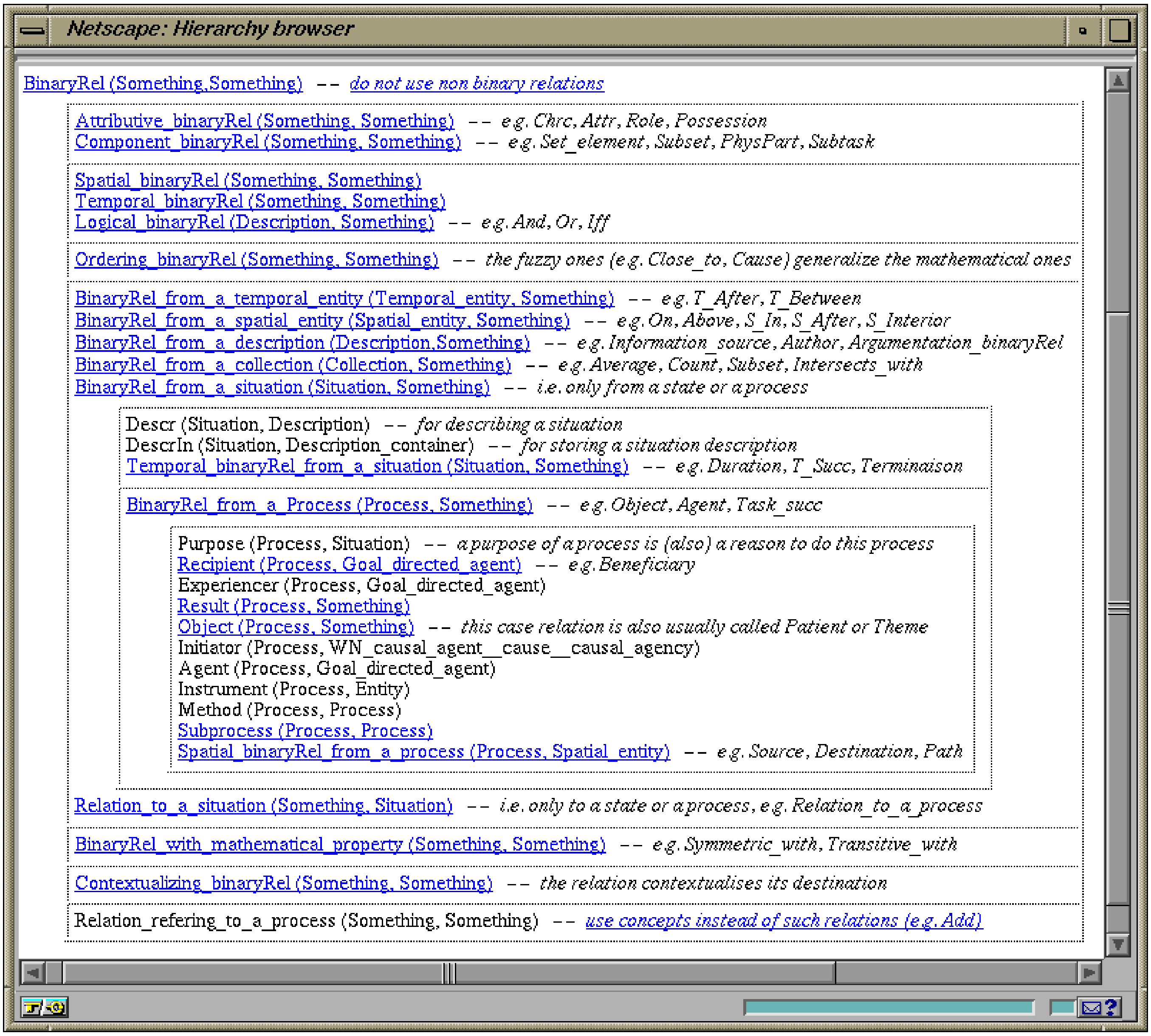

Figure 9: Uppermost relation types of the WebKB top-level ontology as shown by the WebKB hierarchy browser *(types at the same level in a same box are exclusive but are not exclusive with types in other boxes).*

5 Conclusion

Current information retrieval techniques are not knowledge-enabled and hence cannot give precise answers to precise questions (e.g. about the semantic content of documents). This is due to the difficulties involved with automated extraction of knowledge from general documents. As an intermediate step to overcome this problem, a current trend on the Web is to allow users to annotate documents using metadata languages. On the basis of ease and representational completeness, we have argued for the use of a knowledge representation language such as Conceptual Graphs rather than the direct use of XML-based languages. To allow users to represent and query knowledge at the level of detail they desire, we have proposed simple notations for restricted knowledge representation cases and techniques allowing users to leave knowledge terms undeclared. To support this approach, we have presented a top-level ontology, and developed Web-accessible knowledge-based tools and Unix-like tools for indexing, retrieving and generating information. At present, knowledge has to be formulated and stored by users in Web-accessible documents. To improve cooperation, we are extending WebKB to support the building of a Web-accessible knowledge repository by multiple users.

Acknowledgments

This work is supported by a research grant from the Australian Defense, Science and Technology Organisation.

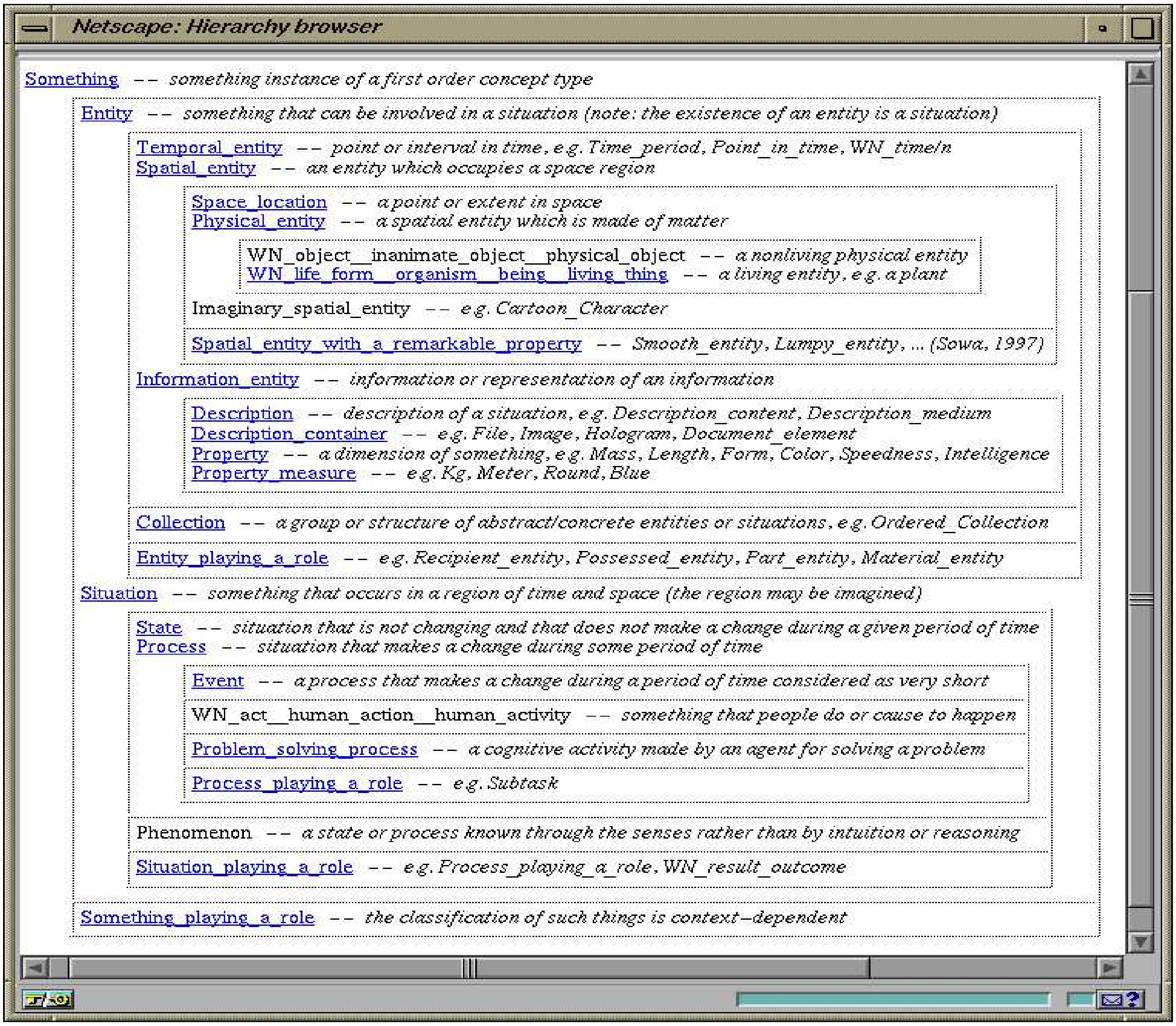

Figure 10: Uppermost concept types of the WebKB top-level ontology *(types at the same level in a same box are exclusive; types beginning by WN come from the WordNet ontology).*

References

[1] S. Decker, et al., Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information, in R. Meersman et al. (eds.), Semantic Issues in Multimedia Systems, Kluwer Academic Publisher, Boston, 1999.

[2] Ph. Martin, Using the WordNet Concept Catalog and a Relation Hierarchy for Knowledge Acquisition, in Proc. Peirce'95, 4th International Workshop on Peirce, Santa Cruz, California, August 18, 1995. http://www.inria.fr/acacia/Publications/1995/peirce95phm.ps.Z

[3] Ph. Martin, *Exploitation de graphes conceptuels et de documents structurés et hypertextes pour l'acquisition de connaissances et la recherche d'informations*. PhD Thesis, University of Nice - Sophia Antipolis, France, 1996. http://meganesia.int.gu.edu.au/~phmartin/PhD.html

[4] J. Nanard, M. Nanard, A. Massotte, A. Djemaa, A. Joubert, H. Betaille, J. Chauch, Integrating Knowledge-based Hypertext and Database for Task-oriented Access to Documents, Proc. DEXA'93, Prague, Springer Verlag, LNCS Vol. 720, Prague, 1993, pp. 721-732.

[5] Schuler, W. and Smith, J.B., Author's Argumentation Assistant (AAA): A Hypertext-Based Authoring Tool for Argumentative Texts, in Proc. ECHT'90 (INRIA, France, November 1990), Cambridge University Press, pp. 137-151.

6 Vitae

Dr Philippe Martin is a Research Fellow at Griffith University's School of Information Technology (Australia). Philippe received an engineer degree in Software Engineering (Sept. 1992) and his Ph.D. in Software Engineering (Oct. 1996) both from the University of Nice - Sophia Antipolis (France). This Ph.D. was undertaken at the INRIA of Sophia Antipolis (France), ACACIA project. Since then, Philippe has been member of the Knowledge Visualisation and Ordering (KVO) group at the University of Adelaide in 1997 and at Griffith in 1998. His main research interests are knowledge representation, sharing and retrieval.

Professor Peter W. Eklund is the Foundation Chair of the School of Information Technology at Griffith University and research leader of the KVO group. Peter graduated with Honours in Mathematics from Wollongong University (Australia) in 1985, was awarded an M.Phil. degree from Brighton University (UK) in 1988, and received his Ph.D. in Computer Science from Linköping University in Sweden in 1991. Peter was a visiting research scholar at Hosei University (Japan) in 1992 and Senior Lecturer in Computer Science at The University of Adelaide from December 1992 to 1997. He was appointed Professor and Foundation Chair of Information Technology at Griffith University in January 1998. His main research interests are knowledge ordering and visualisation.