Conceptual Structures and Structured Documents

Philippe MARTIN, Laurence ALPAY

INRIA - ACACIA project - BP 93 - 06902 Sophia Antipolis Cedex France
E-mail: phmartin@sophia.inria.fr, L.L.Alpay@open.ac.uk

Abstract. In the *first part* of this article, we present the benefits of using a structured document editor for *storing, editing, and structuring knowledge representations*, and show how this can be done for knowledge represented in the Conceptual Graph (CG) formalism. In structured documents, document elements (DEs) are typed and can be organized by structural or hypertext links. They can also be shared (included) by other DEs. The organization of DEs is specified by structure models (Document Type Definitions (DTDs)). A structured document editor displays documents according to their presentation models and lets the users to edit them and to structure them in accordance with their DTDs. We have defined DTDs and presentation models for allowing the storing and display of a CG, a type definition and a hierarchy of types or CGs. Furthermore, we used the functional interface of the structured document editor Thot (previously named Grif) (Quint & Vatton, 1992) so that when a CG, a type definition or a hierarchy is loaded or built in Thot, a similar element is loaded or built in the base of the CG workbench CoGITo (Haemmerlé, 1995).
In the *second part* of the article, we analyse the *semantics of the different kinds of associations* (by hypertext or structural links) *between these knowledge representations and other pieces of information*: contextualization, annotation, representation (we propose constraints on DE representations). Then, we give several ways to exploit these associations, using both Thot and CoGITo, 1) for organizing and retrieving knowledge and/or information, and 2) for combining searches by navigation on hypertext and structural links, and searches by knowledge-based queries. This paper does not advice the representation of document abstract structures with CGs.
Keywords: Document Representation, Information Retrieval, Knowledge Acquisition.

1 Introduction

With Conceptual Graphs (Sowa, 1992), information may be *represented* in a form that is logically precise and humanly readable. Structured documents *organize* information into various *elements* which are *typed*, which can be embedded (i.e. related by structural links) and which can also be related by hypertext links. The organization of the elements is represented in an abstract (or logical) structure which must follow a structure model called the document type definition (DTD). The *presentation* of the elements may be specified in presentation models and various additional presentation rules may be applied to the elements according to their types.

The abstract structures of documents may be exploited for *retrieving, managing and generating documents or document elements* (databases or specialized tools may be used for this). Information retrieval (IR) may also be done by navigation on structural or hypertext links between document elements (DEs)[1]. However, representing

1. A whole document may also be considered as a document element (DE); it can be included in other DEs or linked to them. Then, from now, for sake of simplicity and generality, "DEs" also refer to whole documents. And by "information retrieval" (IR) we mean DE retrieval, not only document retrieval. IR can be done with requests and/or by hypertext-like navigation.

DEs by predefined types (tags, marks), and some of their relations by direct predefined links, is insufficient for high level IR (i.e. semantic, flexible or accurate enough): more complex structures are needed for *indexing* the DEs, that is, for *representing* their content and their relations in a formal and machine-processable way. Knowledge-based approaches have now gained attention both in document retrieval systems (e.g. Croft (1987), Myaeng (1992)) and in hypertext systems (e.g. Marshall & al. (1991), Nanard & al. (1993)). In this article, we will present a) the benefits of using the Conceptual Graph (CG) formalism for indexing or representing the *semantic content* of DEs, and b) the principles that CGKAT (Martin, 1995), our knowledge acquisition tool, follows for allowing and guiding such indexations.

The main idea we exploit is that hypertext systems and structured documents also allow the users to *store, structure and edit not only raw information but also various knowledge structures*, including semantic networks. This knowledge may be handled via some inference mechanisms implemented on the internal representations used in these systems, e.g. inheritance and query mechanisms as in MacWeb (Nanard & al., 1993). DEs (information) may be connected by hypertext links to the knowledge structures (other DEs) which index them, i.e. which represent or annotate them in a formal way. Similarly, using structural and hypertext links, knowledge representations may be associated or mixed with various kinds of information (text, image, section, etc.), and they may be edited, structured and displayed using the various facilities of the hypertext system or structured document editor. We have made this possible with CGKAT : this tool interfaces a structured document editor with hypertext capabilities (Thot[1] (Quint & Vatton, 1992)), with a CG workbench (CoGITo (Haemmerlé, 1995)).

In the first part of this article, we show the benefits of using a structured document editor for representing and organizing knowledge, and how we have enabled this. Then in a second part, we present the principles we have chosen in CGKAT for enabling and guiding the representation/indexation of DEs with CGs.

2 Organizing Knowledge with a Structured Document Editor

Structured document editors, or markup languages like SGML (ISO 8879), permit the users to *organize* chunks of ***information (DE)*** and to *associate* them with some minimal machine-processable ***knowledge (DE representation)*** such as predefined types (marks) and attributes of various types (e.g. number, text and hypertext reference). In addition, using SGML or a similar DTD defining language, the syntax of a knowledge representation language can often be specified. Thus, a structured document editor like Thot, can offer an interesting environment 1) for editing knowledge using a language defined in a DTD, and 2) for associating this knowledge with information[2].

1. Thot is the new name of the academic version of Thot since November 1995.

2. Without a structured document editor, it is useless to define a knowledge representation language in a DTD, since then a) there is no editing environment, b) the user has to use SGML tags for representing knowledge, c) a specialized parser could parse this language without using SGML tags, and it could directly generate adequate representations in memory.

2.1 The Facilities Provided by a Structured Document Editor

A structured document editor like Thot can
1) display DEs of a document according to the chosen presentation model, in one or several windows (each window may display different kinds of DEs);
2) allow the users to apply specific presentation rules (position, font, color, etc.) to a DE provided that they are allowed by the chosen presentation model;
3) guide the document edition, first by presenting the empty DEs to fill in, and then by presenting for each selected DE in a window, a contextual menu which proposes only the editing commands allowed for the DE;
4) let the users retrieve DEs by hypertext navigation[1] or by queries on their types or attributes.

DEs can be ***organized*** by structural and hypertext links. They may also be ***reused*** and ***shared*** by others DEs using the mechanism of *inclusion*: an inclusion is a DE which is a "living copy" of another DE and which is connected to this DE by an hypertext link. This copy is "alive" in the sense that all changes made in the DE source are automatically reflected in the copy (and therefore cannot be directly modified). Inclusions may be exploited to build "*virtual documents*" or "***views***" on parts of other documents. Thus structured document editors permit the users to edit and store information in a modular way. In addition, the document structure may be exploited for a cooperative and concurrent editing of the same document. Alliance (Decouchant, 1995) is an application built on the top of Thot which manages such a cooperation.

With a structured document editor like Thot, documents can also be ***active***: Thot can call a user-defined function when a DE of a specified type, or one of its attributes, is the object of a specified event (selection, modification, saving, etc.). Thot provides a C functional interface for enabling user-defined functions to search information in the abstract structure and modify it. Thus Thot seems to be a good support for making an hypermedia interface for an application.

Therefore, in order to edit, access and manage knowledge, it seems an interesting idea to interface a knowledge base with a structured document editor. This is what we have done in CGKAT, with Thot and the CG workbench CoGITo[2]. Our first work was to define a *DTD and* a *presentation model* for a CG. For this, we have used the S and P languages[3]. Our second (much longer) work was to define *C functions* to modify or to exploit the CG base for each *event* on a DE which displays a CG[4].

1. Hypertext links are bi-directional: from a DE, all the DEs which refer to it, may be retrieved.

2. CoGITo is a CG workbench like Peirce or CGKEE, and like CGKEE it has a C functional interface for letting the users build and manipul its internal representations.

3. With the native version of Thot, a DTD and a presentation model are respectively written in the S and P languages. In the SGML version of Thot, SGML is used instead of S, and in the future, DSSL (ISO 10179) might be used instead of P.

4. In CGKAT, the KB handling functions are separated from the functions called by Thot. The latter collect information in the abstract structure, and call the former by a direct function call or via RPC. Then, if the modification is possible in the CG base, or if the user has requested information from the knowledge base, the abstract structure is modified.

2.2 Building CGs, Type Definitions and Hierarchies Using Document Elements

Figure 1 shows the main parts[1] of our ***DTD for a CG and a type definition using a CG***.

```
STRUCTURE CG;   ...   { Structural model for (a DE which displays) a CG }  ...
  CG = CASE OF
               CGgraph (ATTR  CGname = TEXT;   Comment = TEXT;
                                  Toward_elem=REFERENCE(ANY);  ... )
                              = ConceptOrRel_List ... ;
               SingleConcept = CASE OF Concept; ConceptInclusion;  END;
               TypeDefinition (ATTR ! DefKind = NSC, NC, SC, TC, RelType_def,
                                              Unspecified, Transform_in_CGgraph;
                                  ! Completed = Not_yet,Yes;   Comment;  Toward_elem)
                     = BEGIN  DefinedType = TEXT;  LambdaVar = TEXT;
                              DefBody = CGgraph;
                       END  with DefKind ?= TC,  Completed ?= Not_yet;
       END;
  ConceptOrRel_List = LIST OF (ConceptOrRelation);

  ConceptOrRelation = CASE OF
      Concept (ATTR IDinCG=INTEGER; Comment;Toward_elem; ...)
           = BEGIN  ConceptType = TEXT;
                       Referent = CASE OF Variable=TEXT; Individual=TEXT; END;
                     ? CGReferent (ATTR ... ) = CGgraph;
                     ? ...  { "?" means that the sub-element is optional }
             END;
      ConceptInclusion (ATTR  IDinCG;  Comment;  Toward_elem; ...)
           = BEGIN  TheIncludedConcept = Concept;  END;
      Relation (ATTR  IDinCG;  Comment;  Toward_elem;  ...)
           = BEGIN
                LinkToOrig (ATTR ...) = BEGIN  GRAPHICS;  ? ...   END;
              ? OtherLinkToOrig = LIST OF (LinkToOrig);
                RelationType (ATTR RelationFrame) = TEXT;
                LinkToDest (ATTR ...) = BEGIN  GRAPHICS;  ? ...   END;
              ? OtherLinkToDest = LIST OF (LinkToDest);
             END;
                   END; ...
```

Fig. 1. Main parts of our DTD for a CG (the "..." are omitted parts).

We now detail this DTD. It specifies that a CG may be 1) a *list* of concepts or relations (we have called that a "*CGgraph*"), or 2) a *single concept* (we shall see an example of the use for such a constrained CG), or 3) a type definition. The body of a type definition is a CGgraph and the kind of definition is specified by an enumerated attribute ("DefKind")[2]. This attribute also lets a user specify that a type definition must be transformed into a normal CGgraph (the reverse is done in a similar way). A

1. For space and clarity reasons, we have omitted the parts which are not needed for defining the abstract structure but which needed for the presentation of this structure. If you need the whole DTD and the related models, contact our team leader Dr Dieng (dieng@sophia.inria.fr).
2. For concept types, we distinguish four kinds of definitions: 1) necessary and sufficient conditions, 2) necessary conditions, 3) sufficient conditions and 4) typical conditions (in Sowa's terminology, a definition with typical conditions is called a "schema").

modification on a DE displaying[1] a CGgraph or a single concept is not accepted if the corresponding modification cannot be done in CoGITo, i.e. if it violates some constraints (e.g. relation signatures). Thus such a graph is validated at each step of its construction, syntactically by Thot and semantically by CoGITo. The body of a type definition is similarly validated since it is handled like a normal CGgraph until the user specifies, using the attribute "Completed", that s/he has completed this body.

In this DTD, a *concept* has a type, an individual or generic referent, and may also include a CGgraph in its referent part. In order to let the user set hypertext links between DEs displaying coreferent concepts (whatever CGgraph these concepts are included in), we had to specify two kinds of DEs for concepts: 1) "Concept" (new concept), and 2) "ConceptInclusion" (inclusion of concept). A *relation* is composed of a relation type and links to the connected concepts. Concepts and relations have an attribute for identifying them in a CG ("IDinCG") and another one for referring to one of the DEs that they might represent ("Toward_elem"). Reference attributes are hypertext links. DEs displaying CGgraphs, concepts and relations also have a textual attribute "Comment". Figure 3 shows a DE including others DEs among which the graphical representation of a concept including a CG in its referent part.

When a document is loaded in memory, if some of its DEs display CGgraphs or completed type definitions, the corresponding CoGITo internal representations for these elements are built. When the document is closed, these internal representations are removed. Thus the loaded documents always ***reflect*** the CG base content. The CGs may, via the DEs which display them, be ***associated*** with other information and organized in various ways inside structured documents, using structural and hypertext links and inclusions. The same DE displaying a concept or a CG may be incuded (shared) by various other DEs (e.g. CGs, sections, documents); it is then accessible by hypertext navigation from these DEs and conversely. The associations between CGs and information may be exploited automatically for many purposes, e.g. for retrieving or combining DEs and then for generating new documents. We shall detail an example of such an exploitation done by CGKAT, in section 3.2.

Associating meta-information with CG representations

When a CG is created, some ***meta-information*** (information on its context) may be associated with it, e.g. its author, its creation date, the reason for its creation, the DEs source which have led to its creation, the viewpoint on these sources which was considered for the CG creation (i.e. the considered aspects), the author of the sources, the context in which these sources were stated, the context in which the represented entity or situation is generally used, etc. Meta-information on a CG may be represented using a concept which includes this CG in its referent part (then, conceptual relations may be connected to this concept for expressing meta-information on the CG).

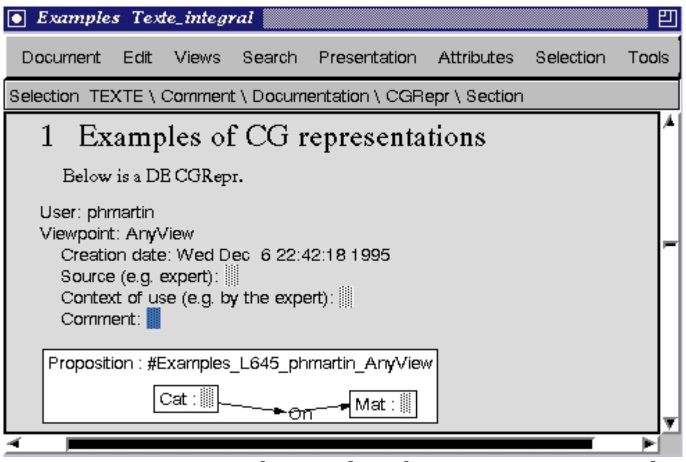
CGKAT also proposes a shorter way for specifying meta-information on a CG: via a "*CGRepr*" (see Figure 2 and Figure 3) or via the attributes "*Comment*". Such attributes can store attribute-value pairs. Filling the slots of a CGRepr is just a more

1. In this article, we prefer to use "display" instead of " represent" for graphical representations.

ergonomic way to fill the attribute "Comment" with some predefined kinds of meta-information. We have defined a ***relation of specialisation on "CGs with their associated comments"***: for testing if a CG Y specializes a CG X, before using the projection operation, CGKAT checks that the attribute-value pairs in the comment of Y "specialize" the ones in the comment of X. This is the case if the comment of Y includes at least all the attributes of the comment of X, and if for each of these attributes, the value for Y is the same or a subtype or an instance of the value for X. Thus in CGKAT, a request on the CG base may be expressed using a CG request and additionnal constraints on the meta-information of the retrieved CGs.

```
STRUCTURE CGRepr;   ...  { Structural model for a representation with a CG } ...
  CGRepr = BEGIN
                User = TEXT; {Author of the CG}       Viewpoint = TEXT;
                Documentation = BEGIN  CreationDate = TEXT;    SourceAuthor = TEXT;
                                       ContextOfUse = TEXT;    Comment = TEXT;
                                END;
                TheRepr = CG;  ...
           END;  ...
```

Fig. 2. Main parts of our DTD for a CGRepr (the "..." are the omitted parts).


Examples Texte_integral
Document Edit Views Search Presentation Attributes Selection Tools
Cat :
On
Mat :
Selection TEXTE \ Comment \ Documentation \ CGRepr \ Section

1 Examples of CG representations

Below is a DE CGRepr.

User: phmartin
Viewpoint: AnyView
Creation date: Wed Dec 6 22:42:18 1995
Source (e.g. expert):
Context of use (e.g. by the expert):
Comment:

Proposition : #Examples_L645_phmartin_AnyView

Note: this CGRepr may have been created for representing the content of one or several DEs, e.g. sentences like "A cat is on a mat.", or images of a cat on a mat. Then, these DEs and this CGRepr may be connected by hypertext links.

These DEs may also be connected to several other CGRepr. See Figure 4 for examples of connections.

Fig. 3. A Thot window showing a CGRepr inside other DEs.

Automatic layout of hierarchies and graphs

The P language in Thot is a "language of boxes", i.e. it lets the users define the characteristics (size, color, relative position, font, etc.) of default presentation boxes for DEs according to their types or the content of their attributes. The P language allows to specify default automatic layouts for a DE of kind "Tree", e.g. a vertical or horizontal layout, or an indented list layout. However, it does not permit the users to specify an automatic placement for the nodes of a graph which was not defined as a tree in its DTD (e.g. a CGgraph). In this case, the placement must be done by the user or be calculated by a C function. Since general hierarchies are graphs, if a tree form cannot always be used for displaying them, they must be defined as graphs, and then default automatic layouts cannot be specified using only presentation models.

In the future, CGKAT may exploit DEs of kind Tree for 1) managing the display, browsing and editing large hierarchies (of types or CGs) by successive generations of documents displaying parts of the hierarchies[1]; 2) organizing and displaying the answers to a request.

2.3 Related Research

A structured document editor like Thot seems adequate for developing a syntactic editor for a language, especially when the graphical aspects are important. For instance, Schaar (1994) developed a programming environment for the graphical state/transition language Argos. Schaar preferred Thot over *presentation editors* like UIL and over *object-oriented graphical editor* toolkits like Unidraw since Thot provided him with the abstract structure that was needed for handling the objects of his language.

Hurwitz & Rich (1993) have written a SGML DTD for a CG but it seems that they did not exploit it to make a CG editor. Compared to dedicated *CG browser-editors*, e.g. GRIT (Leane, 1993), our tool inherits of all the functionalities of a structured document editor. This means, for instance, that the users can associate and mix knowledge with other multi-media information, and 1) organize knowledge and information in many ways and according to many viewpoints; 2) present them in many ways using presentation models or specific presentation rules. However, only simple presentation rules may be specified for a DE in a presentation model, and only according to its type. Further implementation may then have to be done for complementing these rules, e.g. for implementing graph layout algorithms, or for giving a specific presentation to a concept according to its type, a current task or a user model.

In the next section, we show how associations between information and CG elements can be exploited for structuring or accessing information or knowledge.

3 Conceptual Graphs for Structuring or Indexing Documents

There are two ways to organize DEs (DEs may be whole documents) for information retrieval: 1) directly, with structural links or cross-references; 2) indirectly, with an index. Even with typed links, cross-references bear little semantics. A structured index is a better aid to IR. In ***indexes***, most elements are a *representation of the content* of a DE or of a set of DEs. Such a representation may use one or several of the following things: a lexical expression (e.g. a term or key-word), a DE mark, a concept type (or topic or class), an individual concept (or instance), a more complex description of the content of the DE using for example a graph or a logical expression.

It is preferable that the representation of a DE be unambiguous and unique (hence an atomic or defined concept type is far better for indexation than a term or a keyword). In addition, the more *precisely* and *semantically* a DE is represented and

1. At present in CGKAT, hierarchies of types are not stored in structured documents but in special separated files, and the display, browsing and edition of these hierarchies are done via menus (these hierarchies must be loaded in CoGITo first).

this representation *precisely* and *richly connected* to other index elements, 1) the more inferences can be automatically made for retrieving something, so that the user may formulate requests at an abstract level[1], and 2) the more the user has facilities for finding what s/he is looking for by hypertext navigation (e.g. Bernstein (1990) showed that the problem of disorientation which is common with navigation on direct/untyped hypertext links, is greatly reduced when the user can be guided by the semantic of the relation). The problem is that semantic relations and representations are difficult to automatically extract from documents, so they are at present seldom used in document retrieval or large-scale hypertext systems[2].

Representations of DEs may be built for indexing purpose or for knowledge extraction/modelling purpose. In this last case, the indexation of DEs by some elements of the knowledge base may be exploited 1) *for retrieving or reassembling DEs* on semantic criteria and using knowledge-based techniques, 2) *for documenting knowledge elements*, which is useful for explaining them, comparing them or re-using them.

3.1 Representing Document Elements in the CG Formalism

If the CG formalism is used for *indexing* DEs, a user or a program should be allowed to *associate* any type, concept, CG or type definition[3] to a DE. However, to allow reliable uses of DE representations, we distinguish two kinds of DE representations: simple ***"annotations"*** and genuine ***"representations"***.

While we do not put any constraint on annotations, what we now call a *DE "representation"* must only describe the DE and its content[4], i.e. it should not describe things which are related to the DE or its content but which are not (referred to) in the DE. Thus for instance, a representation of a DE may include a conceptual relation between two concepts representing DEs, only if these last DEs are sub-elements of the first one. By itself, a relation or a type is not complete enough to have a logical or semantic interpretation, and therefore may not be a DE representation. However, a

1. This is also the conclusion of the Conventions for the Application of HyTime (CApH, 1995) (Hytime is the ISO 10744 hypermedia document representation standard) which advocates the use of a *topic map* (≈semantic network) and presents some kinds of connections between DEs and a topic map. The CG formalism allows the building of more precise representations, and in section 3.1 we analyse the various types of connections between CG elements and DEs.

2. However, let us note that for document retrieval, Myaeng (1992) exploits CGs extracted from documents by natural language processing (NLP). For hypertext systems, NLP is also used a) for extracting a small semantic network from source texts (the network is then used as an index, but the represented DEs are only words or expressions and sometimes whole sentences) (Nanard & al., 1993), or b) for generating semantic relations between big DEs.

3. A type definition may be a relation type definition (using necessary and sufficient conditions) or a concept type definition (using necessary and/or sufficient or typical conditions). For instance, the sentence "Cats like milk" may be represented by: "Typical conditions for Cat(x) is [Cat]<-(Agent)<-[Like]->(Object)->[Milk].". (In Sowa's terminology, a definition with typical conditions is called a "schema").

4. By "*content of a DE*", we mean the sub-elements of the DE and the real or imaginary entities or situations described or referred to by these sub-elements. By "*DE itself*", we mean the DE seen as a whole, i.e. a single entity which includes, refers to or describes others DEs.

type definition may be used to represent a DE, if the content of this DE is a definition. A CG which includes more than one concept may represent the content of a DE, but not the DE itself and its content, since a DE is an single entity which must be represented by an single element; as a matter of fact, relations between DEs can only be represented by conceptual relations between concepts, not between CGs.

Therefore, in CGKAT there are ***two kinds of hypertext links*** for associating CG elements with DEs: an hypertext link of type "Representation" and an hypertext link of type "Annotation". (The next section shows how the user can use these associations for retrieving DEs via requests on the CG base). When a user begins a representation of a DE, this DE is automatically connected by an hypertext link of type "Representation" to a DE of kind CGRepr which contains the representation. More precisely, the represented DE is connected to the list of all its representations (a list of CGRepr) and each CGRepr is directly connected to the DE it represents (a similar mechanism exits for annotations). A DE may be represented differently by several users, and according to different views (if a particular view is specified, it is a partial representation) but each user may only build one representation of a DE by view.

Only three categories seem necessary[1] to classify the kinds of information that may be extracted from a DE and *represented*:
1) the real or imaginary ***situation*** *(state or process)* described or referred by the DE (situations may be related by spatial or temporal relations),
2) the ***proposition*** *(description, assertion, hypothesis, etc.)* stated by the DE (logical and rhetorical[2] relations may apply on a proposition),
3) the ***format or medium*** of the DE *(word, sentence, logical expression, image, paragraph, multi-media-document, etc.)* i.e. what is usually represented in a document's abstract structure (in CGKAT, there is little benefit in representing the abstract structure with CGs since Thot stores and lets the users search and manage such a structure very adequately).
Thus a DE may be represented according to three aspects. If a concept is used for representing an aspect of a DE, its type is a subtype of either "Situation", "Proposition" or "Statement" (in the terminology of Sowa (1992), but we think that "Medium" is more adequate than "Statement"). If a concept of type Proposition represents a particular DE, it actually represents the fact that this DE is a particular description, so it should be individual, and it may include a CG in its referent part for representing the situation described by the DE (see Figure 3 and Figure 4).

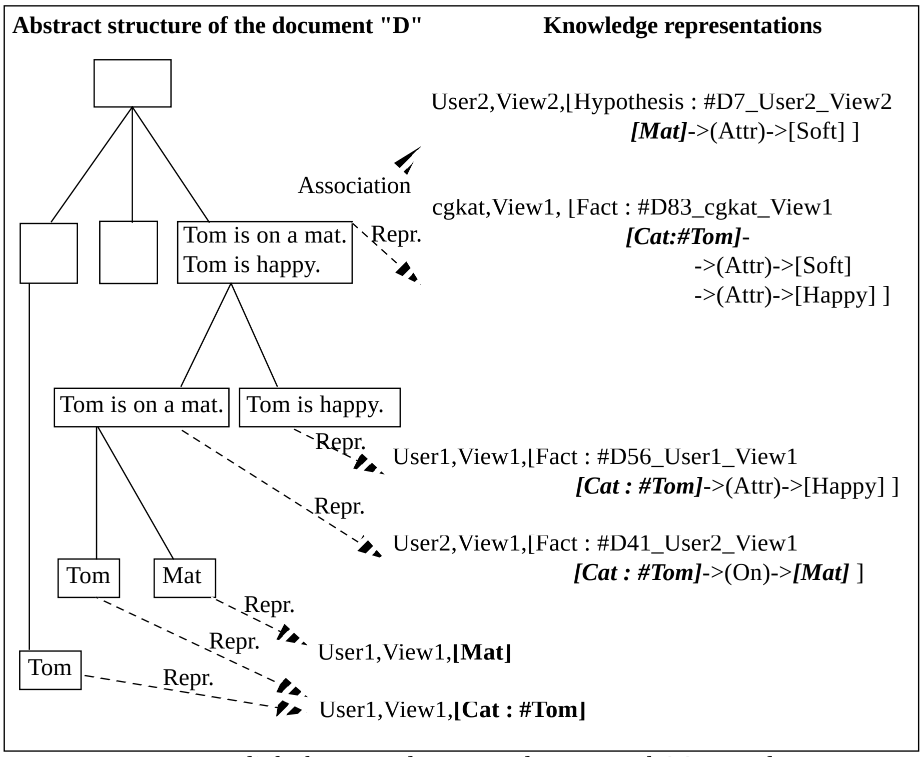
We distinguish DEs which are ***"symbols"*** (e.g. words, collocations, symbols) *referring* to entities or situations, from "***assertions***" (e.g. images, sentences, sections) which *describe* situations. An assertion is composed of symbols and may be repre-

1. We have derived these three aspects from the three distinctions classically used in linguistic or knowledge representation: a) symbol or sign, b) concept or idea or intension, c) referred object or extension. More precisely, for these distinctions, we adopt the terminology and ontology of Sowa (1992) which is derived from the "Situation Semantic" (Barwise & Perry, 1983): a) symbol or statement, b) proposition, c) situation.

2. Examples of rhetorical relations: Summary, Concession, Antithesis, Circumstance, Purpose.

sented by a CGgraph, a concept type definition (if the assertion is a definition) or a single concept of type Proposition (in this last case, CGKAT generates[1] an individual referent for the concept; this allows a user reuse this particular concept in other CGs for representing relations between the represented DE and other DEs). Since a symbol is not an assertion, CGKAT generates warnings when a CGgraph, a concept type definition or a concept of type Proposition is used for representing it.

If some DEs E1,...,En are represented by concepts or CGs, a ***default representation*** for a DE including E1,...,En may be automatically ***generated*** with 1) a maximal join on the CGs of the representations, and 2) a generalization on their meta-information (Figure 4 gives an example, see the CGRepr created by "cgkat"). Other techniques could be used. Such generations save a lot of work to the user even if s/he must check them. The lack of constraints for annotations reduces the benefits of similar generations for them.



Abstract structure of the document "D"
Knowledge representations
User2,View2,[Hypothesis : #D7_User2_View2
[Mat]->(Attr)->[Soft]]
Association
cgkat,View1, [Fact : #D83_cgkat_View1
[Cat:#Tom]-
->(Attr)->[Soft]
->(Attr)->[Happy]]
Tom is on a mat.
Tom is happy.
Repr.
Tom is on a mat.
Tom is happy.
Repr.
User1,View1,[Fact : #D56_User1_View1
[Cat : #Tom]->(Attr)->[Happy]]
Repr.
User2,View1,[Fact : #D41_User2_View1
[Cat : #Tom]->(On)->[Mat]]
Tom
Mat
Repr.
Repr.
User1,View1,[Mat]
Tom
Repr.
User1,View1,[Cat : #Tom]

Fig. 4. Hypertext links between document elements and CGRepr elements (***[Mat]*** and ***[Cat : #Tom]*** : ConceptInclusions of **[Mat]** and **[Cat : #Tom]** Hypertext navigation is possible between inclusions and on structural links. The CGRepr elements may be organized inside one one several documents).

1. CGKAT generates a unique referent name for the concept using 1) the identifier of the DE for the concept, 2) the name of the document where this DE is, 3) the name of the concept author and of 4) the name of the view taken for this concept creation (see Figure 4 and 5).

Easing knowledge representation

Representations of DEs are interesting for document generation (see next section) but in a knowledge modelling context, extracting knowledge of a document via representations of DEs may not be easy or natural. Instead of building DE representations and then synthesizing them, the user should also be allowed to work on a big structure and then use parts of this structure for the generation of DE representations. In CGKAT, the implementation of this function will exploit inclusions.

For *easing knowledge reuse or IR* by many users, the types used in the concepts or relations of DE representations should be derived from an ***ontology*** which is ***shared*** and understandable by all the users. For this, CGKAT exploits the semantic dictionary WordNet (Miller & al., 1990) for providing 1) a default browsable and updatable hierarchy of 90,000 concept types, and 2) a facility for accessing the concept types in this ontology using lexical terms (so the concept types corresponding to the known meanings of the terms are given). This facility eases the use of the ontology and guides knowledge representation. CGKAT also proposes a default hierarchy of 200 relation types: thematic, mathematic, spatial, temporal, rhetoric and argumentative relations (see (Martin, 1995) for details).

3.2 Information/Knowledge Retrieval and Structuring

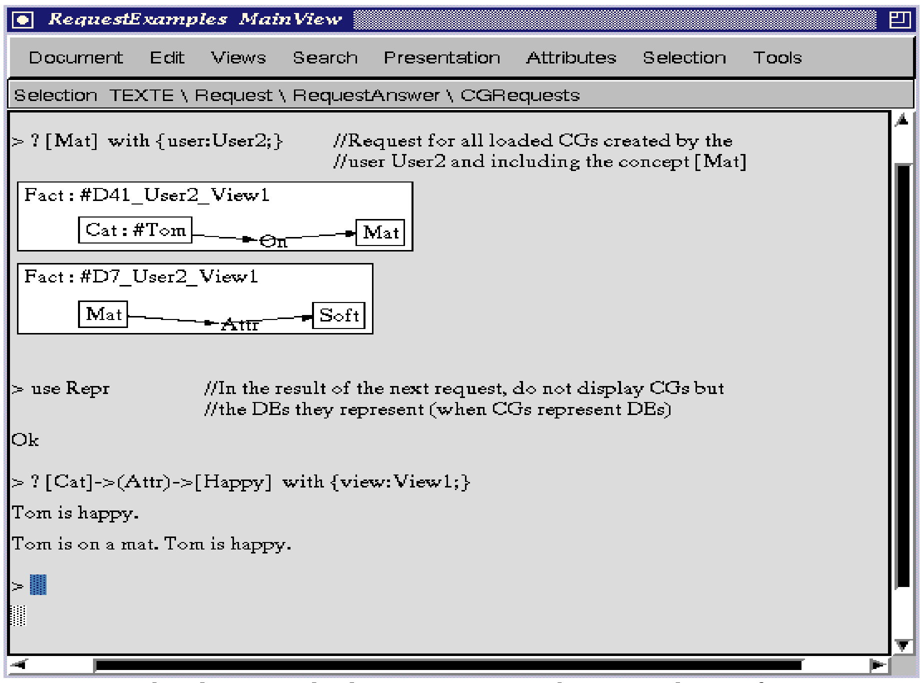
We have seen how in CGKAT a DE may be associated with other DEs by structural links, bi-directional hypertext links and inclusions. Some of these DEs may not only be raw *information* but may be *knowledge representations* and may *index* (represent or annotate) other DEs (*information*). Thus, 1) knowledge representations may be organized and accessed using information structuring/retrieval techniques (e.g. structural and hypertext links), and 2) since knowledge representations index other DEs, these DEs may be organized and accessed using knowledge structuring/ retrieval techniques (e.g. the specialization relations that may be calculated between CGs). Moreover, the main originality of our work is that these two complementary kinds of techniques may be *used in combination* for accessing a piece of information or a piece of knowledge.

For instance, a user may ***navigate*** from a DE to its representation, then navigate to each piece of knowledge (e.g. a CG) which re-uses this representation (inclusions must have been used for enabling this). In this way, a user may know which other pieces of knowledge a representation is connected to, by which semantic relation, and in which context (and from a piece of knowledge, s/he may go to another DE i.e to another information or knowledge). The ***context*** may be formally represented in the CG formalism or via our shortcut for meta-information representation (see section 2.2). It is also given by the structural and hypertext links connected to the knowledge. Although this notion of context is important, the other knowledge oriented hypertext systems we have encountered, can only display user-selected parts of a single flat semantic network (no context can be defined with embedded nodes or via the clustering of the network).

CGKAT also permits the users to ***access DEs with conceptual requests***. A request or a textual command must be written by the user in a DE of kind RequestAnswer (another kind of DE we have defined). Then, the result of the command is displayed by CGKAT inside this DE and another DE of kind RequestAnswer is created for allowing the user to express another command (see Figure 5).

At present in CGKAT, conceptual requests on the CG base can only be searches for specializations of a CG request with possibly associated meta-information (i.e. constraints). For each retrieved CG, CGKAT may present 1) a "graphical representation" of this CG using the DE with which the CG has been built, and/or 2) the DEs which are represented by the CG, and/or 3) the DEs which are annotated by the CG. The user may choose the kind of result s/he wants. In any case, the presented DEs are not simple copies of DEs but inclusions, i.e. living copies. Thus the ***result of a request*** is a *generated DE* which is a ***view on*** the knowledge ***base*** or on other ***documents***, and this view corresponds to the *criteria chosen by the user.* Such results of requests may constitute a whole document or may be combined and completed for producing documentations. They may also be the basis for explanations.

We are working on a very complementary way to make searches in the CG base (and then in the documents since we keep the principles of result presentation): via the search of ***paths*** between source and destination sets of concepts specified by the user. If the retrieved paths are aggregated (this is an option), this method is also a method for generating CGs, but then the layout of these CGs must also be generated. A DE of kind Tree may also be used for organizing the results of a search, if these results are numerous.


RequestExamples MainView
Document Edit Views Search Presentation Attributes Selection Tools
Selection TEXTE \ Request \ RequestAnswer \ CGRequests
Fact : #D41_User2_View1
Cat : #Tom
On
Mat
Fact : #D7_User2_View1
Mat
Attr
Soft

```
> ? [Mat] with {user:User2;}        //Request for all loaded CGs created by the
                                    //user User2 and including the concept [Mat]

> use Repr              //In the result of the next request, do not display CGs but
                        //the DEs they represent (when CGs represent DEs)
Ok

> ? [Cat]->(Attr)->[Happy] with {view:View1;}
Tom is happy.
Tom is on a mat. Tom is happy.

>
```

Fig. 5. A thot document displaying requests on the CGs and DEs of Figure 4.

In addition to menus, CGKAT includes textual commands for CG manipulation (e.g. directed join, maximal join, etc.) and also a ***script language***[1] to let the user combine the results of commands or queries and write scripts for generating *views* and answering "frequently asked questions". Such scripts may be associated with a DE and may be explicitly or implicitly activated (scripts may be considered as dynamic hypertext links). *Generated views* really enable a user to *combine* searches by queries and by hypertext navigation since a view is the departure point of many hypertext links toward the sources and contexts of the DEs collected by the view.

CGKAT also exploits the Thot index facility (Richy, 1994) and the representations of occurrences of terms in a document, for generating a ***glossary*** of these terms which ***synthesizes their representations*** (the terms and theirs representation are alphabetically sorted and duplicates are eliminated). For each term, this glossary gives 1) the type of the concept used for representing the term, 2) the name or the type of the user who created this representation, 3) the name or the type of the view s/he used, 4) the name or the type of the source author. This information is not just copied but made with inclusions of the DEs where it comes from (i.e. sub-elements of a CGRepr). Thus hypertext navigation is possible from this information to the representations. The Thot index facility mechanism also allows navigation from a term in the document to its corresponding entry in the glossary. Then, from an occurrence of a term, a user has access both to the representation(s) of this occurrence and to the synthesis of representations of other occurrences of the same terms. Moreover, when a document is composed of other documents, the Thot index facilty mechanism can synthesize the glossaries of these included documents. Such a glossary is thus a powerful aid to information and knowledge retrieval (and then for knowledge documentation and sharing). Besides, we have seen that ***thematic indexes*** on any combination of conceptual criteria may also be generated via requests with CGs.

3.3 Related Research

CGKAT integrates three kinds of information structuring and retrieval techniques: the hypertext ones, the structured document ones and the knowledge-based ones.

Knowledge acquisition (KA) tools generally only permit the users to document some predefined kinds of knowledge elements with unstructured parts of text, using hypertext links, and generally have no other IR mechanism, even though the benefits of using hypertext systems for KA are often highlighted. However, few hypertext systems ***integrate a knowledge-based approach***, i.e. allow the represention of knowledge and then exploit it for enabling hypertext navigation, semantic queries *and the combination of these two techniques*. MacWeb (Nanard & al., 1993) is an exception to this. CGKAT has approaches or functions similar to MacWeb, especially in the use of views, but MacWeb is a tool developed from scratch whereas CGKAT combines the facilities of two complementary specialized tools, a structured document editor and a CG workbench. Moreover, CGKAT allows to represent and to exploit

1. CKAT can call 'sh' (shell, the standard UNIX system command interpreter) and conversely.

contexts, via the CG formalism and via the use of structural links. RIME (Kheirbek & Chiaramella, 1995) is an IR system which, like MacWeb, implements everything in the same underlying formalism. RIME uses the CG formalism but is more oriented towards document retrieval, so CGKAT ressembles much more MacWeb than RIME. Other advantages of CGKAT over these tools are 1) to provide synthesis of representations of terms using special *glossaries*, 2) to explicitly handle many *representations* and *annotations* of a same DE by *many users* and for *many views*. Bürsner & Schmidt (1995) have shown that such views are of a great help in KA.

The CG formalism provides a logic-based and semantic way to represent knowledge. CGKAT exploits the relation of specialization between CGs for queries. Other ***kinds of queries*** are necessary for IR, e.g. the search of paths, and the search of nodes according to the number of relations they are connected to (Beeri & Kornatsky, 1990). In addition, for document retrieval or any other search using superficial representations of DEs for indexing them, the IR techniques must also handle the "recall" and "precision" factors (here, the indexation goal is not to really represent information for enabling users to directly find it with queries, but to index DEs by minimal discriminant representations for enabling searches of some DEs among many others).

4 Conclusion

We have presented the benefits of using a structured document editor for editing, presenting and structuring knowledge representations, and shown how this can be done for CG representations. Then, we have shown how to associate knowledge elements with other pieces of information, and we have analysed the semantics of these associations: contextualization, representation, annotation. Thus this work specialises some guidelines given by CApH (1995). Finally, we have given several ways first to exploit such associations for organizing and retrieving knowledge or information, and second to combine searches a) by navigation on hypertext and structural links, and b) with knowledge-based queries. CGKAT also helps to build knowledge by providing default ontologies for concept and relation types. For concept types, CGKAT exploits the semantic dictionary WordNet. CGKAT has been applied to retranscriptions of interviews of experts in road accident analysis (Alpay, 1996).

5 Acknowledgements

The authors thank the members of the ACACIA team, especially Dr Rose Dieng and Dr Olivier Corby for their advice on the writing of this article. We are also grateful to the reviewers for the comments on our paper, and the numerous corrections of the English.

6 References

Alpay L. (1996). Modelling of reasoning strategies, and representation through conceptual graphs: application to accidentology. INRIA Research Report, 1996.

Beeri C. & Kornatsky Y. (1990). *A Logical Query Language for Hypertext Systems*. In Proc. of DEXA'90, Vienne, Autriche, August, 1990.

Bernstein M. (1990). *Hypertext and technical writing*. In Proc. DEXA'90, Int. Conf. on Databases and EXpert systems Applications, Vienn (Austria), August 1990.

Bürsner S. & Schmidt G. (1995). Building views on conceptual models for structuring domain knowledge. In Proc. of KAW'95,Gaines, B.R. Eds, University of Calgary, Banff, Alberta, Canada, February 26-March 3, 1995.

CApH (1995). Drafts of the "Conventions for the Application of HyTime" (CApH). Available at ftp.techno.com//pub.CApH.docs.

Croft W.B. (1987). *Approaches to intelligent information retrieval*. In Information Processing and Management, Vol 23, no. 4, 1987.

Decouchant D. (1995). *Structured Cooperative Editing and Group Awareness*. In HCI International'95, 6th International Conference on Human-Computer Interaction (Editors: Anzai Y. and Ogawa K.), Elsevier Science, Yokohama, 9-14 July 1995.

Haemmerlé O. (1995). *CoGITo: une plate-forme de développement de logiciels sur les graphes conceptuels*. Ph.D thesis, Montpellier II University, France, January 1995.

Hurwitz A. & Rich W. (1993). *GML for Conceptual Graph Networks*. IBM Technical Report No 03.485, May 1993.

Kheirbek A. & Chiaramella Y. (1995). *Integrating Hypermedia and Information Retrieval with Conceptual Graphs*. In HIM'95, Konstanz, Germany, April, 1995.

Marshall & al. (1991). *Aquanet, a hypertext tool to hold your knowledge in place*. In Proc. of the 3rd ACM Conf. HTX'91, ACM Press, San Antonio (Tx), 1991.

Martin Ph. (1995). *Knowledge Acquisition Using Documents, Conceptual Graphs and a Semantically Structured Dictionary.* Proc. of KAW'95, Gaines, B.R. Eds, University of Calgary, Banff, Alberta, Canada, February 26-March 3, 1995.

Miller G.A., Beckwith R., Fellbaum C., Gross D. & Miller K. (1990). *Five Papers on WordNet*. CSL Report 43, Cognitive Science Laboratory, Princetown University, July 1990. (Papers and system available at clarity.princeton.edu//pub).

Myaeng S.H. (1992). *Conceptual Graphs as a Framework for Text Retrieval*. Conceptual Structures: current research and practice (Editors: Nagle T.E., Nagle J.A., Gerholz L.L., and Eklund P.W.), England, Ellis Horwood Workshops, 1992.

Nanard J., Nanard M., Massotte A-M., Djemaa A. Joubert A., Betaille H. & Chauché J. (1993). *Integrating Knowledge-base Hypertext and Database for Task-oriented Access to Documents*. In Proc. of DEXA'93, 4th Int. Conf. on Database and EXpert systems Applications (Eds: Varik V., Lazansky J., Wagner R.R.), Prague, Sept. 1993.

Quint V. & Vatton I. (1992). *Hypertext Aspects of the Grif Structured Editor: Design and Applications*. R.R. 1734, INRIA, Rocquencourt, July 1992.

Richy H. (1994). *A hypertext electronic index based on the Grif structured document editor*. In Proc. of Electronic Publishing, vol. 7, num. 1, pp. 21-34, March 1994.

Schaar Ph. (1994). *Un environnement de programmation pour le langage graphique Argos*. CNAM engineer report, IMAG, Grenoble, France, March 1994.

Sowa J.F. (1992). *Conceptual Graphs Summary*. Conceptual Structures: current research and practice (Editors: Nagle, T.E., Nagle, J.A., Gerholz, L.L., and Eklund, P.W.), England , Ellis Horwood Workshops, 1992.