Pergamon

Information Processing & Management, Vol. 31, No. 4, pp. 593–604, 1995
Elsevier Science Ltd. Printed in Great Britain

0306-4573(95)00004-6

COMFRESH: A COMMON FRAMEWORK FOR EXPERT SYSTEMS AND HYPERTEXT

F. A. KOKKORAS and I. P. VLAHAVAS
Department of Informatics, Aristotle University of Thessaloniki, 54006 Thessaloniki, Greece

(Received December 1993; accepted in final form June 1994)

Abstract—Intelligent hypertext is a promising approach to information systems, because it combines the power of inference of expert systems and the intuitive power of hypertext. In this paper we propose the "COMFRESH", a common framework for expert systems and hypertext. It is based on a Prolog interpreter and uses the conceptual graph knowledge representation formalism for browsing and reasoning. COMFRESH can be used as a knowledge based hypertext (intelligent hypertext) or as an expert system with hypertext capabilities.

Keywords: Hypertext, Expert systems, Conceptual graphs.

1. INTRODUCTION

The term *hypertext* or *hyperdocument* is used to describe networks of electronically stored data that in the simplest form are text sections called *nodes*. The conjunctions between nodes are called *links*, while the software that controls this network is called *hypertext system* and it is an information system.

A hypertext system enables the user to manipulate (create, delete, traverse, modify, annotate) nodes and links easily, to search the network for specific information, to create understandable semantic presentations of the network, to produce linear documents based on the network and to collaborate with other users in authorship.

Much work has been done so far to make hypertext systems user-friendly and sophisticated frameworks. The sense is that trying to produce such systems for the end users we create *cognitive overhead* (Conklin, 1987) for the authors of hyperdocuments, as they have to add more attributes to nodes and links. It is clear that neither all the systems are proper for any kind of information, not all the users have the same demands. Furthermore, the associations that an author has made should be understandable to different readers.

To bypass these disadvantages, hypertext systems must include some dynamic features more than just adding procedures to links and to nodes. Expert systems attached to hypertexts is a valuable approach as they give to the latter inference abilities. In addition, expert systems can explain better their decisions to the user by using the hypertext's features. This combination of hypertext and expert systems, usually called intelligent hypertext or expertext (Rada *et al.*, 1990), is a kind of information system that is currently under intensive research.

Most of the existing intelligent hypertext systems are based in a collection of expert modules. Each module uses specific representation of the information to serve a particular need of the user (browsing, inferencing, etc.). Thus, multiple representations of the documents are needed which is disadvantageous.

In this paper we propose an intelligent hypertext model that supports browsing and inferencing using a uniform representation; the conceptual graph (CG) knowledge representation formalism. We also present the "COMFRESH" (COMmon FRamework for Expert Systems and Hypertext) an implementation of the proposed model.

Our model uses the conceptual graphs (CGs) as structural units (elementary semantic nets) to create complex, query related semantic nets, instead of giving a firm, complex one. Thus, it can

serve users with different organizational needs. Browsing is supported via the concepts (components of a CG) which serve as hypertext links. To uphold inferencing and browsing the COMFRESH includes a Prolog interpreter with the extra ability to handle CGs. It also allows the addition of expert modules written in Prolog serving in this way a testbed for new tools. It is also flexible enough to work as an inference engine with hypertext capabilities.

The problems that occur in the existing intelligent hypertext technology are discussed in Section 2. Section 3 describes the CG formalism together with a hypertext model based on it. In Section 4 a short description of the proposed system is given and, finally, Section 5 concludes the paper and presents our future plans.

2. PROBLEMS IN THE EXISTING INTELLIGENT HYPERTEXT SYSTEMS

The problems related with the intelligent hypertext systems can be classified in three main domains: system design, end-user and author related problems. In the following we discuss these problems.

2.1. System design—the representation problem

One way to build an intelligent hypertext system is to use an expert shell in co-operation with a hypertext system and to associate rules of the expert shell with text blocks. In another approach, expert modules are used to acquire information from multiple represented documents. Both approaches do not use a uniform representation way of the information to support browsing and inferencing.

For instance, the large expert information system, called I^3R (Intelligent Interface for Information Retrieval) (Croft & Thompson, 1987), has a collection of independent but co-operating expert modules each serving text retrieving, using a different strategy. The disadvantage is that the data these modules use are structured in multiple ways to suit each expert module's function.

Logic Petri net model provides a bridge between the informal semantic nets of hypertext and the formal logic systems. Although it gives new abilities to intelligent hypertext systems, this model is inconvenient, and encoding the knowledge of a document into this representation may, however, be as difficult as the problem of building expert systems (Rada *et al.*, 1990).

2.2. End-user related problems

The interconnected nodes of a hyperdocument constitute a semantic net. There are two types of semantic nets in hypertext; the embedded and the independent one (Collier, 1987). The former is easier to handle and includes links that serve as pointers to other nodes which generalize or specialize the topic of the father node. In fact, it is just a net, not a real semantic net.

The independent semantic net is more powerful as it can be traversed and analyzed without visiting text blocks. It provides a uniform representation way for both browsing and querying but it is a firm structure. Thus, it can not serve users with different organizational needs.

An easy way to bypass the above problem is to have multiple nets available. The drawback here is that the author can not safely predict all possible nets in all instances. If he creates balanced nets (Rada & You Geeng-Neng, 1991), where automatic net reconstruction is possible, he is restricted to organize his information in a particular way.

Some hypertexts support the dynamic generation of views. The user posts a query based on the attributes of the various objects of hypertext (nodes, links) and the system retrieves the parts of the hyperdocument that satisfy the query (Clitherow *et al.*, 1989). Moreover, artificial intelligence techniques can be used to generate sub-graphs based on the parts of the hyperdocument that satisfy a query (Gallagher *et al.*, 1990). The latter method seems promising

but adding to many attributes to nodes and to links the authoring process becomes inconvenient.

2.3. *Problems in authoring process*

The most difficult task of the authoring process in hypertext is the linking one. Working on our earlier hypertext system (Kokkoras, 1992), we found that the linking process became extremely difficult when users were asked to work on hyperdocuments created either by others or by themselves but after a long pause.

It is obvious that in a real hypertext application with too many nodes, the system should be able to observe the author and warn him in illegal actions (linking, etc.) according to pre-defined rules. Furthermore, the system should be able to suggest possible target nodes based on the hyperdocument's structure or even to create automatically some links. In the first case, in the linking process, some global, link related constraints are needed to make the system able of eliminating the possible target nodes but the second case is difficult to handle with the existing hypertext models.

3. CONCEPTUAL GRAPHS AND INTELLIGENT HYPERTEXT SYSTEMS

A flexible, precisely defined and understandable knowledge representation notation can lead in an effective and easily modifiable expert system. If we want to create an effective and user-friendly intelligent hypertext, we should give great attention to the expert part of it.

The CG model for the representation of knowledge can be used to create sophisticated, intelligent hypertexts. This model is a general framework for expressing natural language semantics but it is also a practical way to express a large amount of pragmatic information by assertions. All the algorithms are domain-independent and every semantic domain can be described through a purely declarative set of CGs. In addition, the CG model can present high-order logical relations which are difficult to represent in a simple first-order logical formalism (Fargues *et al.*, 1986).

A definition of CGs and how this formalism can be used to create intelligent hypertext systems are presented in the following.

3.1. *Conceptual graphs: primitives and definition*

The elements of the CG theory (Sowa, 1984; Sowa & Foo, 1987; Sowa & Way, 1986) are *concept-types, concepts* and *conceptual relations. Concept-types* represent classes of entity, attribute, state and event. Concept-types can be merged in a lattice whose partial ordering relation < can be interpreted as a categorical generalization relation. Thus, CAR < VEHICLE represents that the CAR is a kind of VEHICLE.

A *concept* is an instantiation of a concept-type and it is denoted by a concept type label inside a box or between brackets (Fig. 1). To refer to specific individuals, a referent field is added. Two identical concepts having different referents are not comparable by the < relation.

Conceptual relations show the relations between concepts. Each relation will be constrained

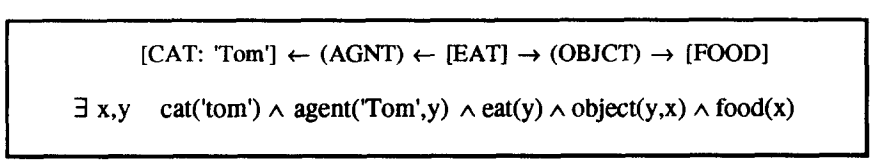

Fig. 1. A Conceptual Graph and its equivalent mapping into first order logic.

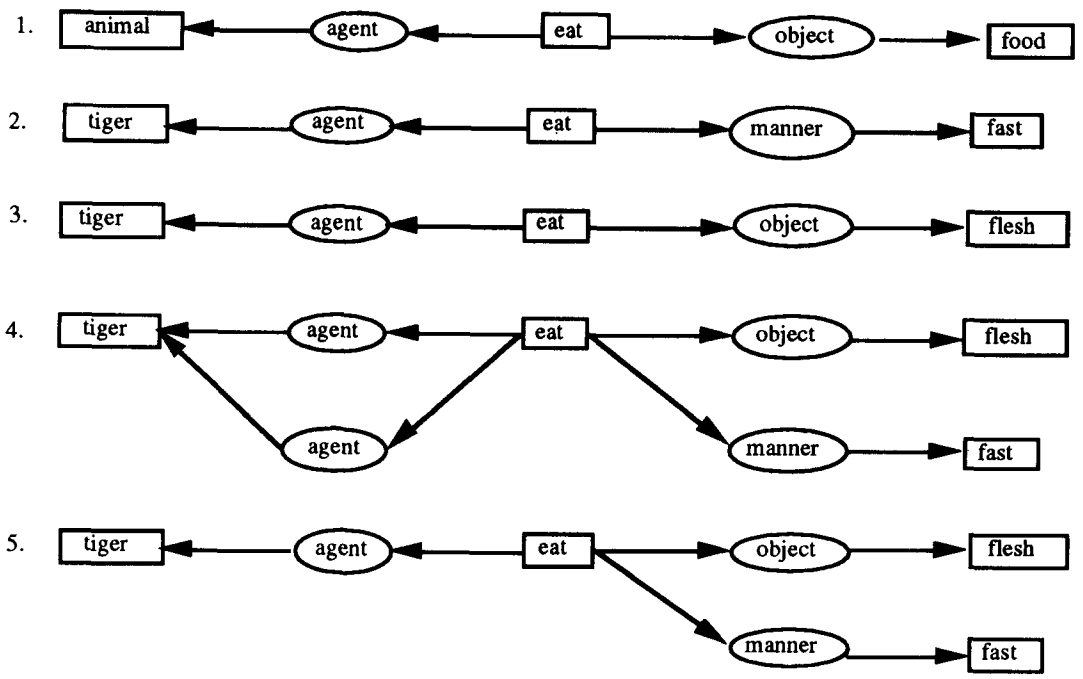

Fig. 2. The maximal join.

as to the concepts it can connect. As with concepts, there should be pre-defined but expandable set of relation-types in a given system. They are denoted by a relation label inside a circle or between parentheses (Fig. 1).

A *CG* (Figs 1 and 2) is a connected graph formed by concept and relation nodes. Each relation is linked (only) to its requisite number of concepts, and each concept to none or more relations. A CG represents information about typical objects or classes of objects in the world and can also be used to define new concepts in terms of old ones. The definition of the relation $<$ on concepts can be extended to a partial relation on CGs.

There is a precisely defined mapping from a CG into first-order logic; it gives a conjunction of predicates, one corresponding to each node of the graph (Fig. 1). A number of operations (formation rules) are also defined on CGs, by which one can derive allowable graphs from a canonical basis. The main operations are:

- *Restriction* takes a graph and replaces any of its concept nodes either by changing the concept-type to a subtype or adding a referent where there was none before.
- *Joining* joins two graphs with a common concept over it, to form a single graph.
- *Simplifying* removes any duplicate relations between two concepts.
- *Contraction* tries to replace a subgraph of a given CG by a simple concept (or relation) using the definition of this concept.
- *Expansion* is the opposite to contraction operation (Fig. 3).

The sequence of examples in Fig. 2 form what is known as the *maximal join*. It is a join of two graphs followed by a sequence of restrictions, internal joins and simplifications so that as much matching and merging of the original graphs as possible is performed. The maximal joint can be regarded as a generalized unification operation (Jackman, 1988). The extension of the relation $<$ to CGs does not confer a lattice structure on the set of all CGs, because it is possible to exist several maximal overlaps between two CGs.

Deduction with CGs is performed via "top-down resolution algorithm". A query expressed as CG can be answered either with direct match with a fact-CG of the knowledge base, or with indirect matching by using inferencing rules. The Sowa's classical "Oz" example is given in Fig. 4.

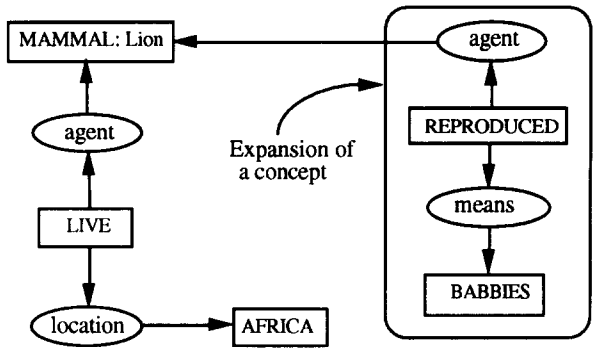

Fig. 3. Browsing with Conceptual Graphs.

3.2. A conceptual graph based intelligent hypertext model

The CG formalism can be used in hypertext to support both inferencing and browsing. The knowledge base of such a system includes CGs that correspond to the knowledge of a document base, among with rules about how to use this knowledge. Some of these rules (e.g. formation rules) might be common for all hyperdocuments.

One advantage of using CGs in hypertext is that we do not need to create complex semantic

A person is a citizen of Oz if and only if any of the following conditions are true:
1) This person is born in Oz.
2) One of his parents is a citizen of Oz.
3) This person is naturalized in Oz.

clause (inference rule):
[CITIZEN: *x] ← (MEMB) ← [COUNTRY: 'Oz']
⇐ [PERSON: *x] ← (AGNT) ← [BORN] → (LOC) → [COUNTRY: 'Oz']
clause (inference rule):
[CITIZEN: *x] ← (MEMB) ← [COUNTRY: 'Oz']
⇐ [PERSON: *x] ← (CHLD) ← [PERSON: *y] and [CITIZEN: *y] ← (MEMB) ← [COUNTRY: 'Oz']
clause (inference rule):
[CITIZEN: *x] ← (MEMB) ← [COUNTRY: 'Oz']
⇐ [PERSON: *x] ← (RCPT) ← [NATURALIZE] → (LOC) → [COUNTRY: 'Oz']
clause (fact)
[PERSON: 'Tinman'] ← (AGNT) ← [BORN] → (LOC) → [COUNTRY: 'Oz']
↓
(CHLD) → [GIRL: 'Dorothy']

goal clause : "Who is a citizen of Oz country ?" ⇐ [PERSON] ← (MEMB) ← [COUNTRY: 'Oz']

result :
[CITIZEN: 'Tinman'] ← (MEMB) ← [COUNTRY: 'Oz']
[CITIZEN: 'Dorothy'] ← (MEMB) ← [COUNTRY: 'Oz']

Fig. 4. Deduction with CGs. The classical Sowa's "Oz" example.

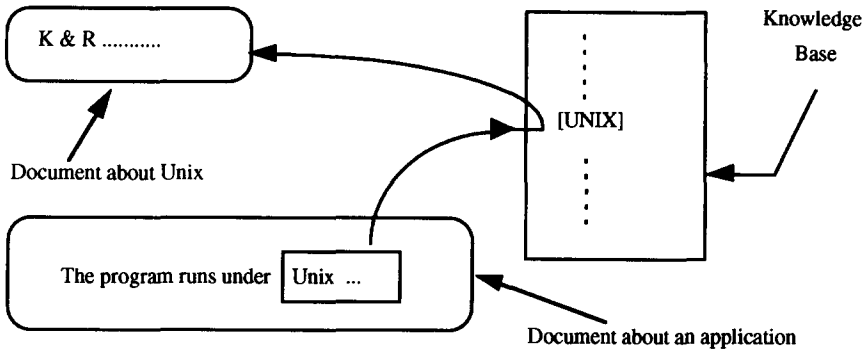

Fig. 5. Use of a concept as a hypertext link.

nets manually. The CGs are already semantic nets (the conceptual relations correspond to the links and the concepts correspond to the nodes). Moreover, there are techniques to create CGs automatically by document parsing. Using CG algorithms (maximal join etc.) we can create complex query based, graphical views of our data. We pose a query in CG form and the system displays graphically CGs that satisfy this query. When these views are inadequate, we can ask the system to replace some concept-nodes of the graph (expansion operation) with their CG definition (Fig. 3) or to display the document by which a CG is derived from. From the other hand, a CG definition can be suppressed to a single concept-node (contraction operation) to prevent a messed graph representation (Fig. 3).

A word of a document, that matches a concept of a concept-type hierarchy, can serve as link to the document that describes this concept (Fig. 5) or to the graph definition of the concept itself (Fig. 3). Thus, a word inside a document can serve as an embedded link in classical hypertext, by means of a concept. Embedded links can be also used to connect a document with comments, annotations and non-textual data such as graphics, sound or video.

To support inference, an inference mechanism should be attached to the system. There is a precise mapping of CGs into first-order logic and some of the CG algorithms are just special cases of Prolog's built-in logical inference and pattern matching. There is also a conceptually powerful technique, *metalevel programming*, in which we can also write rules about how to use other rules (*metarules*). Thus, logic programming and Prolog offer a conceptual common basis (logic theory) and a practical technology to handle this formalism (Sowa & Way, 1986).

Having in mind that Prolog is the best language to handle CGs, the next question is how to implement a CG based hypertext. One way is to create a stand-alone program written in Prolog, capable of doing certain operations. The disadvantage of this approach is that if someone wants to make modifications to the system he must modify its source code (Prolog code). This is an impossible task for the end-user.

Instead of using Prolog to create just a stand-alone program, we can additionally attach a Prolog Interpreter (Prolog Inference Engine—PIE) as a part of the final program. This interpreter can also be written in Prolog. In this way, we have already attached to the system the right programming language for the kind of data we use. The result is a program that can be used as a testbed for new ideas regarding intelligent hypertext. These new ideas can be implemented as modules written in Prolog and activated by the PIE within the same framework. The value of the embedded PIE becomes obvious in the following, simple example.

Consider that we want to know if the concept [person: "Tom"] exists, in this form, in a knowledge base. Consider also that the knowledge base includes the concept [man: "Tom"] instead of the above, while in the concept hierarchy exists a statement ([man]<[person]). The only way to get the right answer (which is: "No, there is not such a concept") is to perform the matching operation without using the concept hierarchy. If the system does not support such an operation we can write a Prolog predicate that does, and activate it via the PIE.

Actually, all the tools of an hypothetical system can be implemented in Prolog and activated by the embedded PIE. However, it is more efficient, to implement the basic tools as part of the kernel of the system.

4. THE COMFRESH SYSTEM

In this section we give details on implementation and evaluation of the COMFRESH.

4.1. Implementation

The COMFRESH (Fig. 6) is a single user, knowledge based hypertext system that uses CGs for browsing and reasoning. It is written in Prolog and consists of:

- A *document base*, which is a file (hyperdocument) that includes portions of text (nodes). Each node displayed in a separate window, while it can be as large as 64 kbytes. More than one document bases are allowed. Furthermore, COMFRESH supports links between nodes belonging in different document bases, but with some limitations, as we will describe in the following.
- A *knowledge base* that consists of CGs, concept-types, concept-type hierarchy, concepts and relations that correspond to the document base's knowledge. Although many knowledge bases are allowed (one knowledge base for each document base), only one of them is used at any time. By default, this is the knowledge base of the hyperdocument, the active node belongs in.
- A *kernel* that includes the system's interface and a Prolog interpreter. The interface is as user friendly as possible with dialogue boxes, pull-down and pop-up menus. Each node is displayed in its own window while several nodes can be displayed in overlapping windows. Figure 7 displays a typical screen of COMFRESH. On the top of the screen there is a pull-down menu with options regarding file handling, text editing, browsing and knowledge base manipulation. COMFRESH's help system is a hyperdocument in COMFRESH format with its own knowledge base. A status line at the bottom of the screen reminds the user the main available options. In Fig. 7 two nodes and the Conceptual Graph browser are opened. The word "Kiklades", in the text of the upper left window, is a link to the node displayed in the central window. These links are

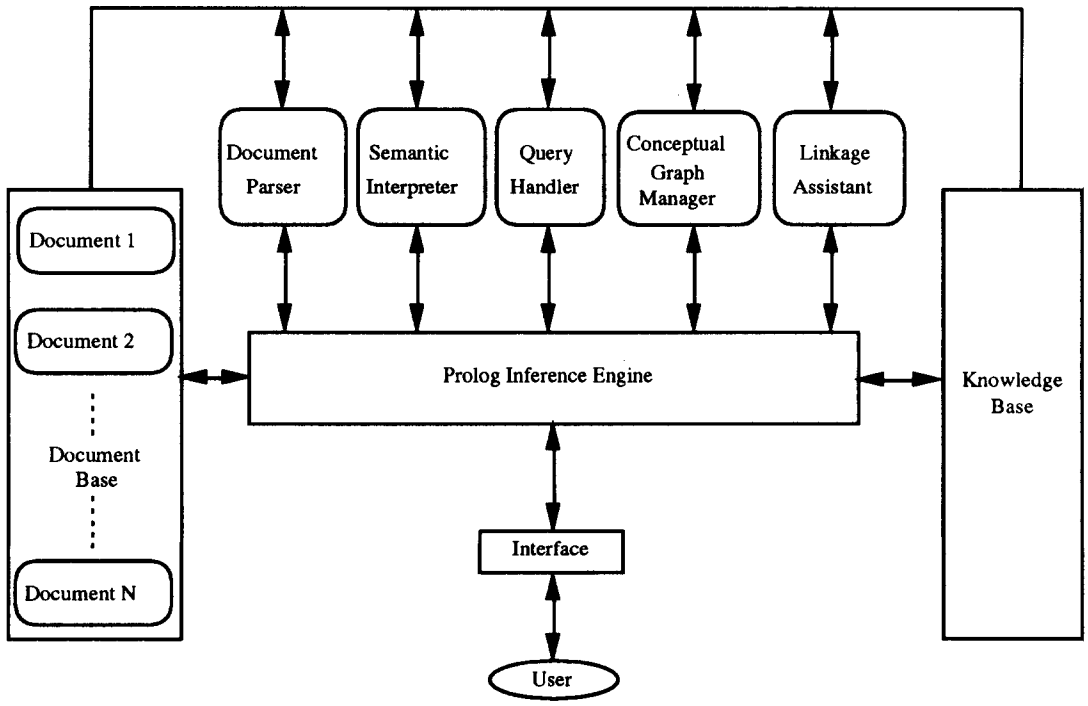

Fig. 6. The COMFRESH system.

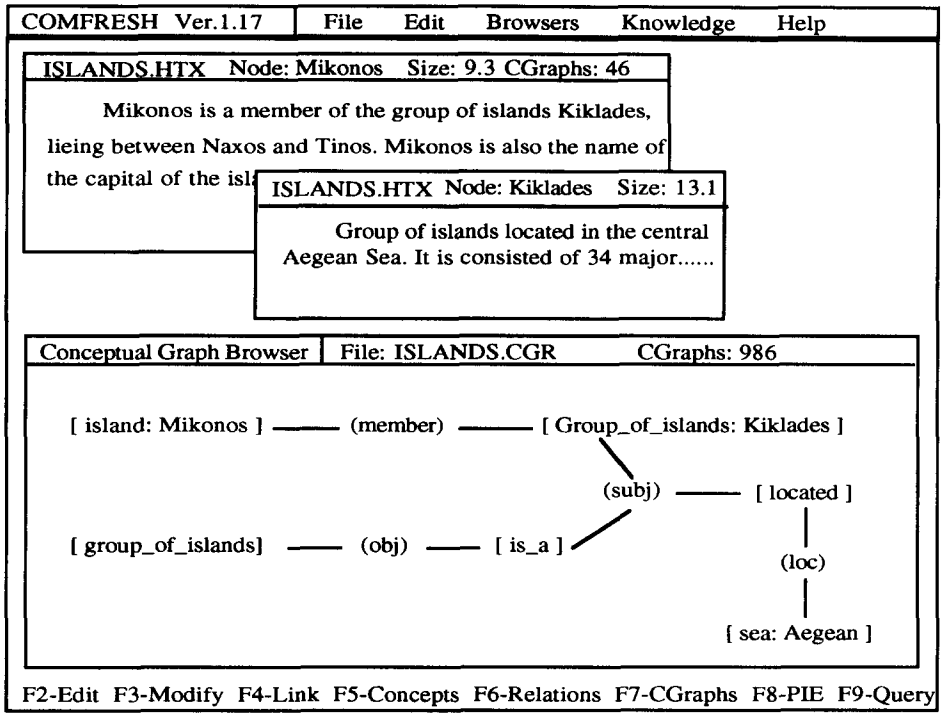

Fig. 7. A typical screen of COMFRESH.

distinguished from the rest text by their different color. The way the Conceptual Graph browser works is described later in this section.

- *Several modules* for the user/author written in Prolog.

COMFRESH in its current implementation includes five modules: a parser, a semantic interpreter, a query handler, a knowledge manager and a linkage assistant.

The *parser* uses syntactic rules to generate parse trees corresponding to all or the user desired sentences of the documents.

The *semantic interpreter* translates these trees into CGs and asserts them into the knowledge base if only they fulfill the canonical formation rules. Currently, there are some user-driven actions, both in parser and the semantic interpreter, but our plan is to make them work as user-independent as possible. Usually, these user driven actions concerning the replacement of certain parts of a sentence (verbs, nouns, etc.) with synonyms existing in a given knowledge base. The user can also force the semantic interpreter to abandon some parts of a sentence with no interest. In the following example, the way the user intervenes becomes clear.

Consider the sentence: "Mikonos is a nice place for vacation". The user should point out that the word "Mikonos" must be a reference in the concept-type [island]. The words "place" and "vacation" can be used by the system to reduce the possible concept-types.

The *query handler* lets the user to construct queries concerning the knowledge base. A query is expressed as a CG and is constructed either directly or indirectly. In the first case the user selects the appropriate items (concepts and relations) from a combination of menus. In the latter case the user selects a type of query from a previously defined set of types, expressed in natural language. In Fig. 8 we can see two types of queries. Each of them is coupled with a semi-structured CG. The user fills in the empty fields of either the natural language or the CG expression (text in *italics* in Fig. 8) to make the question complete. This action is similar to filling in the reference fields of some concepts. With the query completed, the system tries to answer it (top-down resolution algorithm) based on the CGs of the knowledge base. Any query can be saved for future use.

The *knowledge manager* includes the canonical formation rules, which are heavily used by the semantic interpreter and the inference engine. It also supports other operations such as

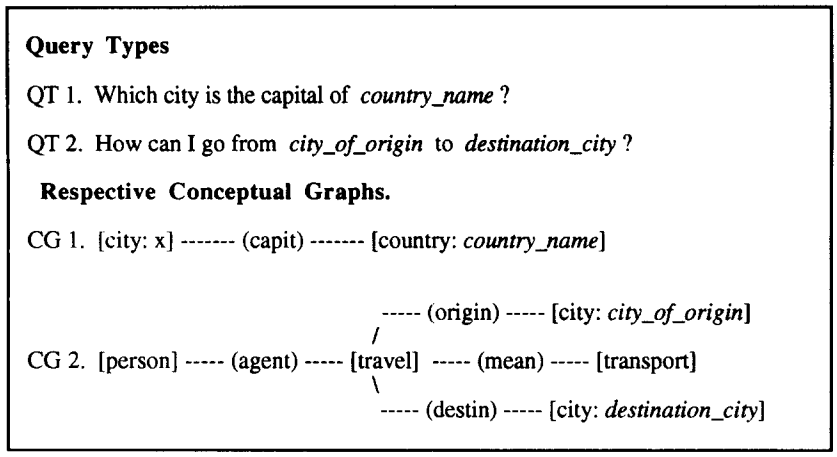

Fig. 8. Pre-defined query types expressed in natural language and CG form.

review of the knowledge base, expansion of the recognized concept and relation types and manual assertion of new CGs.

The review of the knowledge base is performed in CG level. The user opens the knowledge base file and displays any CG in graphical form. Many CGs that satisfy user defined criteria can also be displayed. These criteria are filters that allow the user to inspect CGs having a common property. Furthermore, the user can change parts of a CG or group of CGs, or even to delete a CG. In any of these cases the corresponding nodes are displayed in an editor window and it is up to the user to make the same changes into the text. For example, the user may want to see all the CGs concerning the population of capital cities in order to update the population numbers. In fact, any of the filters mentioned above is a kind of query.

The *linkage assistant* serves in many ways. When a new node is added in the document base, this module finds all the occurrences of the concept-types in this node and prompts the user to decide which of them will serve as embedded link. This is very useful when a concept appears many times in the text.

Another case when the linkage assistant is invoked, is when the user creates an embedded link manually. Here the user selects a word of the text and the linkage assistant searches for related, target nodes. A node is related to the selected word if:

- the header of the node (nodename) is semantically related to the selected word,
- the selected word is semantically related to a concept that is part of a CG of this node,
- the selected word is semantically related to any part of the node's text.

The higher the rule is in order, the highest its priority. Moreover, any of the above rules can be ignored. Usually, the last rule is ignored because is time consuming.

We give here an example of the above process. Consider a document base with nodes regarding Greek islands. Let us say that in three of these islands (nodes 1, 2 and 3 respectively) there is a middle-aged castle, and that there is also a fourth node (node 4) talking about *middle-aged buildings in Greek islands*, having exactly this title (nodename). We have selected the word "castle" inside node 1 in order to create a link and we ask the linkage assistant to suggest the target node. According to the rules stated above the linkage assistant answers that the best target node is node 4 (instead of nodes 2 or 3).

The last case the linkage assistant is used is to check the correctness of the manually added CGs. For the latter function the tool is based on the constraints of the conceptual relations as well as on user defined constraints, to make warnings or suggestions concerning the particular CGs. For example, a constraint may want the relation (capital) to joint two concepts of the type [city] and [country].

From the implementation point of view there is another module, the *toolbox*. This contains global predicates used by the main modules. It is worth noting that some predicates from one module are used in another. For example, the query handler uses predicates of the linkage

assistant module during the construction of a query CG.

COMFRESH is a program of medium size. In its current version it consists of about 8500 lines of code. The main part is the kernel with a total of 4600 lines of code shared by the interface and the embedded Prolog interpreter. The parser is about 1500 lines of code, while the toolbox module is about 1000 lines of code. The size of the other modules varies from 300 to 500 lines of code.

4.2. Evaluation

The COMFRESH system is still a prototype and it is used in our labs for evaluation. For this purpose a hyperbase was created regarding 30 Greek islands. A total of 50 text nodes (≈4000 sentences) were created; a node for each islands, plus 20 more nodes concerning aspects such as groups of islands, ancient civilizations, historic periods, architectonic features of buildings etc. The size of ech node varies from 2 to 20K with an average size of 7K per node.

Most of the text material was written in the COMFRESH's editor. The rest of it was written in stand alone editors and were imported in COMFRESH for further manipulation. Each text file was parsed to create the knowledge base. The evaluation team had previously decided what kind of data would contribute to the construction of the knowledge base. Such data were: name and population of islands and capital cities, monuments and their location, famous beaches, places of entertainment, landmarks, hotels, ways for accessing each place and so on.

Sixteen (16) relations (agent, subject, object, location, member, origin, destination, mean etc.) and 35 concept types (city, capital, island, person, travel, shop, transport, rent etc.) were defined. A total of about 1700 concepts and 2100 conceptual graphs were produced after the parsing procedure.

Several typical queries were applied to measure the performance of the interferencing algorithms. For this test we used a typical 486-based PC but the source code is easily portable to any workstation supporting Prolog. Table 1 displays some queries and their CG form. In Table 2 we give details regarding each query (the type of operation performed, the size of the query CG, the time needed by the system to give either an answer or all the possible answers and finally the total number of answers). It is worth saying that although the two queries are identical for the end user, the underlying operation is different. The times in the first time column in Table 2 concerns the matching of a CG that is located in the middle of the knowledge base (the knowledge base is scanned linearly).

The matching operation is a task that is used frequently in most of the inference procedures. Let us say taht we want to find out if a CG taken from a knowledge base is matched with a given CG. The computational complexity in such an operation is proportional to the second power of the number of relations inside the given CG. The upper limit of the number of relation matches needed to match the two CGs is given by the following relation:

$$\text{RelMatches} = (1+R) * R/2$$

where R is the relations of the given CG.

Table 1. Typical queries applied in COMFRESH

1	How can someone go from city of Athens to island Kriti (*direct ways*) —(origin)—[city: Athens] [person:*]—(agent)—[travel:*]—(mean)—X —(destination)—[island: Kriti]
2	How can someone go from city of Rodes to the island of Mikonos (*complex ways*) —(origin)—[city: Rodes] [person:*]—(agent)—[travel:*]—(mean)—Y1 —(destination)—[city: X] AND —(origin)—[city: X] [person:*]—(agent)—[travel:*]—(mean)—Y2 —(destination)—[island: Mikonos]

Table 2. Response times of COMFRESH for the queries of Table 1

Operation		Query		Stop at . . .		
		Rel.	Con.	1st answer	Find all answers	Total answers
1	Matching (use of hierarchy)	4	5	1.5 s	3.1 s	6
2	Matching (use of maximal join)	8	10	3.5 s	8.1 s	4

In a knowledge base consisting of N CGs, the mean upper limit of the number of relation matches needed to match one of these CGs with a given CG is RelMatches $* N$ /2, given that the knowledge base is scanned linearly.

5. CONCLUSIONS

In this paper we presented an intelligent hypertext model and we described COMFRESH, an implementation of it. Our system combines the intuitive power of hypertext systems and the power of inference of expert systems, to support multiple strategy information retrieval. The advantage of COMFRESH is that it uses a unique representation way for browsing (hypertext function) and querying (expert system function). It can serve readers with different organizational needs, because it offers elementary nets (CGs) to construct complex and query based graphs via the inference engine. The latter is a Prolog interpreter capable of handling CGs.

As it described previously, COMFRESH is a dual system. It can be used as an expert system with hypertext features in the explanation of its conclusions, or as a hypertext system with artificial intelligent techniques in searching and retrieving of information. We currently work on the second approach but the first is also in our plans together with a multi-user version.

Furthermore, we plan to explore how do different kinds of data affect the usability of COMFRESH. Highly organized data (such as geography related data) can produce powerful applications. Less organized information expected to affect primarily the authoring process, but the knowledge representation model that was used can handle almost all kinds of knowledge, helping in this way to bypass this problem.

REFERENCES

Collier, G. (1987). Thoth-II: Hypertext with explicit semantics. Paper presented at *Hypertext '87*. Chaper Hill, N.C.: University of North Carolina.

Conklin, J. (1987). Hypertext: An introduction and survey. *Computer, 20*, 17–41.

Clitherow, P., Rieckel, D., & Muller, M. (1989). VISAR: A system for inference and navigation in hypertext. *Proceedings Hypertext '89*. New York: ACM.

Croft, W. B., & Thompson, R. H. (1987). I^3R: A new approach to the design of document retrieval systems. *Journal of American Society of Information Science, 38*, 389–404.

Fargues, J., Landau, M. C., Dugourd, A., & Catach, L. (1986). Conceptual Graphs for semantics and knowledge processing. *IBM Journal of Research and Development, 30*(1), 70–79.

Gallagher, L., Furuta, R., & Stotts, P. D. (1990). Increasing the power of hypertext search with relational queries. *Hypermedia, 2*(1), 1–14.

Jackman, M. K. (1988). The maximal join for conceptual graphs. In J. F. Sowa, Foo & Rao (Eds), *Conceptual graphs for knowledge systems*. Reading, Mass.: Addison-Wesley.

Kokkoras, F. (1992). *3DHYP: Implementation of a hypertext system using PROLOG*. Diploma thesis (in Greek).

Rada, R., & You Geeng-Neng (1991). *Balanced outlines and hypertext*, Department of Computer Science, University of Liverpool.

Rada, R., Dunne, P., & Barlow, J. (1990). Expertext: From semantic nets to logic Petri nets. *Expert Systems with Applications, 1*, 217–229.

Sowa, J. F. (1984). *Conceptual structures: Information processing in minds and machines*. Reading, Mass.: Addison-Wesley.

Sowa, J. F. & Foo (Eds) (1987). *Conceptual Graphs for Knowledge Systems*. New York.
Sowa, J. F., & Way, E. C. (1986). Implementing a semantic interpreter using Conceptual Graphs. *IBM Journal of Research and Development, 30*(1), 57–69.