INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa • University of Ottawa

DocKMan's Knowledge Base Builder:

A Tool for the Organization of Knowledge

by

Jamie Prpic

Thesis submitted to the School of Graduate Studies and Research

of the University of Ottawa

in partial fulfillment of the requirements for the

Master's degree in Computer Science

Under the auspices of the

Ottawa-Carleton Institute for Computer Science

Department of Computer Science

University of Ottawa

Ottawa, Ontario, Canada

August 1998

Figures and Tables

Acknowledgements

I would like to thank the following people and organizations for their support and assistance through this sometimes difficult and always challenging experience.

Judy Kavanagh: Thanks for keeping me company in the lab for the past year and a half. It would have been much lonelier without you there. Your sense of humour helped preserve my sanity.

Wratko Hlavina: I learnt a great deal during our collaboration on the graph integration. Your outlook on life and ability to not take yourself or our profession too seriously brought much needed perspective to this work.

John Talbot: Your hard work and assistance in wringing out the last few bugs (here's hoping!) and documenting the system were invaluable.

Emre Erkol: Our collaboration during various courses helped get me to this moment. Thanks.

Will Elazmeh: Thanks for convincing me to forsake race car driving.

To the CSI4900 students I assisted: Samir Almousawi, Paulo Bellem, Katha Kulisangam, Oscar Vargas and Gurbir Samagh: Your hard work, perseverance, curiosity and interest in working on a 'technical frontier' such as Java and the Web (despite lack of documentation and bugs from operating systems through to browsers) were a fine example of the best qualities of software developers.

To the CSI 4116 and 5127 students who used DocKMan and persevered through various glitches and gremlins.

Dr. David Goforth: for having inspired this journey.

To my friends Chris, Paul and Tammy: Your friendship is a treasure. Your perseverance an inspiration.

To all those whose presence graced the lab, Kristen MacIntosh, Laura Davidson,

Dr. Ingrid Meyer and others, thank you for keeping us company.

Mitel, Nortel and the National Science and Engineering Research Council(NSERC): thanks for helping me keep the bills paid

Wil Hunt and Ryan Lassen of Symantec Corp. Your usually prompt answers to my many questions proved invaluable to the development of this thesis.

To my parents and brothers: Your support in the various twists and turns of my life (with more to come no doubt) are invaluable.

To my advisor Dr. Doug Skuce: whose efforts to tackle the difficult problem of information retrieval and knowledge management prompted me to select him as my advisor, thanks for having guided the development of this thesis.

Abstract

Keywords: data browsing, knowledge management, knowledge organization, information organization, information retrieval, knowledge base, graphical representations, Java applet.

Information, both print and online is growing at an exponential rate. The ability to locate individual items of information has not kept pace. Once this information has been brought to bear on a problem, the resulting knowledge is lost. It is difficult to organize and store the results of an information search.

We propose a tool to assist in the resolution of these problems. DocKMan's knowledge base builder. Through the use of a hierarchical concept organization and a frame-based inheritance scheme, we believe we have found a means of structuring information which allows easier browsing and retrieval of knowledge. The combination of this 'structuring tool' with a text analysis tool facilitates the construction of organized repositories of knowledge or 'knowledge bases'. Text can now be analyzed for specific pieces of information which are then inserted into the appropriate area of the hierarchy. Such a set of tools will prove especially useful to knowledge workers in technically oriented fields.

1 Chapter 1 - Introduction

1.1 Introduction

The retrieval and presentation of information is becoming an ever more important concern as the mass of print and online documents grows exponentially. Users usually assume, or hope, the answers they seek exist somewhere in the set of resources at their disposal e.g. online texts, books and manuals. However, they must spend inordinate amounts of time searching for an answer to their particular question. The current 'state of the art' in information retrieval normally involves entering several terms or small phrases in a search engine. The user must then wade through a substantial number of hits, some more relevant than others. Minutes to hours can easily be spent. In their annual internet usage surveys, GeorgiaTech's Visualization and Usability centre reports that : "The largest category of users spends between 5 and 15 minutes searching before they find the **first** [emphasis added] piece of useful information."[1] Since many users surveyed (20.7%) access the web for reference material "several times per day,"[2] this can result in significant amounts of time being used. Several problems with both the location and presentation of information are :

Volume

The number of documents returned in a typical query is quite high. Many of these are 'noise' i.e. of poor quality or completely irrelevant. Eighteen percent of web users responding to a series of questions about 'problems using the web', cited

"not being able to find the information they were looking for" as their biggest problem, surpassed only by 'speed of download' at 25%.[3]

Navigation

The only means of navigation through the knowledge space is through entering search terms and then scrolling through text. The only context provided is the 'bolding' of words.

Loss of query results

Users often conduct searches with specific questions in mind. The knowledge gained is often lost since there are few convenient ways of organizing and storing query results. The current state of the art is typified by so-called F.A.Q. lists.

Precision and conciseness

Natural language documents can have a low density of useful knowledge per minute of reading.

Uncontrolled terminology

Within a single document, words can have several meanings or synonyms

1.2 Proposed Solutions

This thesis proposes a step toward a solution to some of these problems of information organization, retrieval and navigation. The design and implementation of a

Web-based knowledge management tool 'DocKMan' , will enable user groups to successfully retrieve and organize knowledge. DocKMan consists of two components, the 'knowledge base builder' and a text analyzer and retrieval tool. The knowledge base builder is a tool to organize and structure information as a knowledge base. A knowledge base is essentially textual information stored in a highly structured format. This portion of the tool can thus be viewed as a kbms (knowledge base management system). Its features include the ability to:

- organize concepts into hierarchies
- structure facts and information into succinct statements
- attach additional information or 'facets' to statements
- inheritance schemes which allow statements to pass from super to sub concept
- navigate through a body of knowledge by various parameters e.g. concept, statement selection

DocKMan's second component, the text analyzer and retrieval tool, allows users to both conduct analysis operations on text such as :

- concordances: places a word in context, splitting the sentence to the left and to the right of the word
- frequency counts: on both nouns and verbs
- parts of speech analysis: breaks into formats such as subject/verb/object and noun before verb.

As well as to launch conceptual queries on a document such as

- What is the definition of
- What is a broader term (super) for?
- What parts does/have [word]?

The text analyzer and retrieval tool can be used in conjunction with the knowledge base builder to build knowledge bases. In chapter five, this tool's operation will be explained in greater depth. An example of constructing a knowledge base using it will also be presented.

1.3 Potential Users

DocKMan's first audience is knowledge workers in technically oriented fields. In addition it would be useful in Knowledge Engineering, where it would allow easier organization and construction of knowledge bases. An initial knowledge base for a topic could be built by a group of users or knowledge engineers and then increased by a user community. As answers to questions or facts were found, users could enter them in a knowledge base. Thus a structured repository of knowledge would be created.

1.4 Potential Applications

Several potential applications of DocKMan's knowledge base builder can be envisioned including:

1. The development of reference material: A knowledge base would allow browsing of a

domain, quickly finding the subject and its specific statements.

2. Specification: organizing the specifications or requirements for a system
3. Clarifying ideas: The act of building a knowledge base can assist in disambiguating user's ideas. They debate the different senses of terms and identify different perceptions of subject meanings.
4. Organized knowledge storage: Groups of users can work to organize knowledge by subject or other criteria such as questions.

1.5 Goals of Thesis

The goals of the thesis were to design and test a prototype of DocKMan's 'knowledge base builder'. We tested the tool on a group of students to see how useful and easy it was to build a knowledge base using DocKMan. As well, several students built a more extensive knowledge base for Java using the tool. Their experience will be briefly discussed.

1.6 Organization of the Thesis

Chapter 2 investigates related work. We examine the strategies behind several knowledge base systems and contrast them briefly with the DocKMan approach.

Chapter 3 presents DocKMan(DKM) , its user interface and discusses typical usage. It also explains some of the general concepts behind DKM. Finally, the general operations users can access are explained.

Chapter 4 discusses the technology underlying DKM and various aspect of the implementation .

Chapter 5 contains examples of use and walks through an example of knowledge base development.

Finally, Chapter 6 summarizes the thesis and presents some future work that could be done on DKM.

2 Chapter 2 – Related Work

2.1 Introduction

The task of organizing and representing knowledge has motivated the research efforts of many computer scientists. Different viewpoints exist about the means of accomplishing this task. The application of such knowledge representing systems has also been varied. I will survey several such systems, describing their general approach to the problem. In all, four systems will be analyzed , CYC, Code4, Ontolingua and one commercial product KnowItAll .

2.2 CYC

CYC, a system developed by AI pioneer Douglas Lenat , has the ambitious goal of capturing "a large portion of what we consider consensus knowledge about the world." [4] Lenat argues that a 'knowledge infrastructure' of common sense understanding is needed in order to enable a variety of knowledge-intensive applications. This knowledge is contained in a CYC knowledge base. They define a KB as a "formalized representation of a vast quantity of human knowledge: facts, rules of thumb and heuristics."[5] The facts are represented by a formal language called CycL.

The knowledge base consists of:

a) terms: the vocabulary of CycL

b) assertions: relate the terms between each other

The authors emphasize that CYC is not a frame based system but rather "a sea of assertions, with each assertion being no more 'about' one of the terms involved than another."[6]

2.2.1 CYC Terms

Concepts within the knowledge base are represented as CYC terms or 'constants'. These constants can represent "a collection (such as the set of all people), an individual object (a particular person), a quantifier (such as 'there exists'), a relation (a predicate, function., slot, attribute)."[7] The example below illustrates one concept entry:

> **#$Skin**
> A (piece of) skin serves as outer protective and tactile sensory covering for (part of) an animal's body. This is the collection of all pieces of skin. Some examples include #$TheGoldenFleece (representing an entire skin of an animal) and (#$BodyPartFn #$YulBrynner #$Scalp) (representing a small portion of his skin).
>
> **isa:** #$AnimalBodyPartType
>
> **genls:** #$BiologicalLivingObject#$AnimalBodyPart#$SheetOfSomeStuff #$VibrationThroughAMediumSensor#$TactileSensor "[8]

The entry begins with the concept name preceded by the symbols #$. Following this a brief English comment is made and some other concepts already in the knowledge base are furnished as examples. Finally, the relationship between this constant and others in the knowledge base is made explicit by CYC 'relations'.

2.2.2 CYC Relations

The example above shows two of Cyc's relations, the 'isa' and 'genls'. The isa relation means the constant is an 'element of' the listed constants. The genls relation

means it is a 'subset of' the listed constants. The authors note that these two relations are distinct. For example:

> #isa Jamie #$Mammal is true while #genls Jamie #$Mammal is false, since Jamie is not a subset.

2.2.3 CYC UI

The development of CYC also required the creation of methods and tools for browsing and editing vast knowledge bases. The most popular tool is a hypertext knowledge base browser. HTML pages describe CYC terms and show all the assertions in which the terms are involved. All occurrences of CYC terms have links to pages describing the term. Therefore, users can follow a "network of relationships". CYC also contains several application modules, one of the most important being the natural language processing module.

A natural language processor attempts to use CYC's knowledge in attacking some important linguistic problems. For example, Lenat explains the problem of understanding, showing: "How can one tell which meaning of "pen is intended in "The box is in the pen" and "The pen is in the box...statistics, collocation and frequency do not resolve such questions. But the task goes from impossible to trivial if one already knows a few things about boxes and pens. The same relationship characterizes CYC and Machine Learning, because learning occurs at the fringe of what one knows."[9] In order to break out of this box, Lenat believes we have to "prime the pump" by manually crafting axioms covering a

large portion of knowledge. Then Natural language understanding and Machine learning would enable further knowledge extraction.

2.3 CODE4

CODE4 or Conceptually Oriented Design/Description Environment version 4 was developed at the University of Ottawa after several years of using other tools to capture knowledge about software. From these experiences, the developers believed: "there is a need for knowledge bases which primarily act as amplifiers of human intelligence, rather than systems designed to run autonomously, and for systems that can assist average computer users to store and communicate knowledge more effectively than they do at present".[10] This goal differs in substantial ways from those of CYC in that the tool

a) allows average users to enter knowledge

b) allows a flexibility of knowledge representation i.e. does not have a fixed syntax, semantics and ontology† like CYCL .

c) favours expressiveness (the ability to enter and express knowledge) over a complex inferencing mechanism such as CYC's).

† Ontology is defined in Webster's as : "a branch of metaphysics concerned with the nature and relations of being or a particular theory about the nature of being or the kinds of existents." However for our purposes, it is a collection of general concepts.

The roots of CODE4 are found in several areas including:"

- frame based inheritance systems
- conceptual graphs
- object orientation
- description logic systems "[11]

2.3.1 CODE Concepts

The first unit of knowledge in CODE4 is the 'concept'. A concept can be either a type (car), or an instance (my car), in fact "the thing may be abstract or concrete , real or imagined, it may be an action, a state or fact. These concepts are organized into a hierarchy of which 'thing' is the top concept and all others subconcepts."[12] Concepts have associated properties and values. A concept, property and value together form a statement. This notion has been made easier for users to understand by underscoring its linguistic sense, e.g. statements have subject(concept) and predicates (properties). For example, CODE4 could contain a statement about a concept 'cat' with property 'meows' and value 'when hungry'. More specific concepts inherit properties from their more general ancestors.

2.3.2 CODE Hierarchies

Code 4 contains five kinds of hierarchy:

- Inheritance hierarchies (common to all frame based Knowledge Representation engines)

- Property hierarchy
- Statement hierarchy
- Relation hierarchies (semantic nets)
- Facet hierarchies

All concepts in CODE4 participate in the inheritance or 'isa' hierarchy. Statements normally inherit from a parent to its children.

2.3.2.1 Property or Predicate Hierarchy

This ordering can be seen as an 'implies' order. For example, if 'swim' is a subproperty of 'float', then any statement that X swims implies that X floats. In fact, "the property hierarchy forms a second "dimension" in a knowledge base, [in addition to the concept dimension] one found extremely useful for further classifying knowledge".[13]

2.3.2.2 Statement Hierarchy

Concepts inherit subsets of the property hierarchy. For example, human can inherit the property subhierarchy 'float => swims'. The statement hierarchy is formed from the combination of the concept and its property subhierarchy. Continuing the example, if the concept hierarchy included "child is a human," the statement hierarchy would comprise:

- Human can float
- Human can swim.

- Child can float
- Child can swim

This structure assists in indexing and creating appropriate statements, since the predicates of new statements should fit under existing one. In effect, statements can be thought of in both dimensions "under the super of the subject and under the super of the predicate(property). When entering a new statement the user must choose both the super of the subject and the super of the predicate."[14]

2.3.2.3 Relation Hierarchies

A typical example of this hierarchy is the 'part of' relation. It is a property that "applies recursively: the parts of things are usually themselves things that have parts."[15] For example:

- Concept: car
- Property: parts
- Value: wheels, engine

The values in the parts field can themselves be concepts.

2.3.2.4 Facets

Statements can be made recursively about statements. These are 'facet' hierarchies. Facets can be classified in three types:

Facets corresponding to noun complements following the predicate. e.g. John bought <a car> <from Ottawa Cars> <on Jan 4> would have 3 facets, of which the first is the statement value (the statement has subject: John, predicate: bought). The user is responsible for creating and naming these additional facets, e.g. 'seller' and 'date'.

Facets for additional information that would be part of a sentence such as modal, quantificational, temporal and adverbial modification. These do not correspond to noun phrases. "[16]

2.3.3 User Interface

The CODE4 UI features a control panel to control various views and parameters. It also serves as the interface for knowledge base actions such as saving, renaming, merging. There is also a feedback panel which indicates to users the results of each command. Common 'semantic' errors are indicated as well. For example, if a user tries to delete a concept with several properties, he would be offered several solutions:

- To move the properties "up" to the superconcept
- Delete the concept anyway and lose its properties

2.3.4 Browsing Knowledge

Knowledge maps:

Based on some starting concepts and relations users can display a 'knowledge map'. Below is a small sample of a finite state machine for the use of an ATM card using a 'relation' hierarchy.

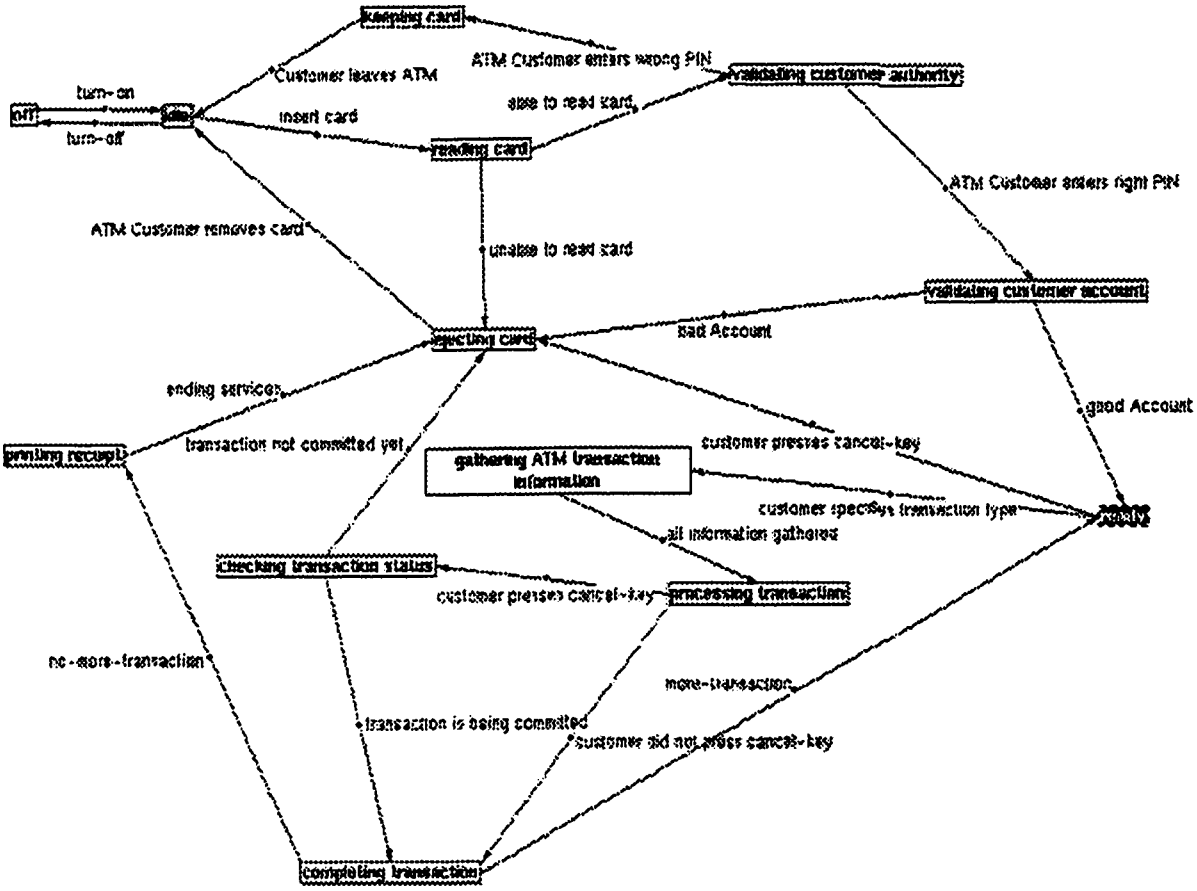


Figure 2.1

2.3.4.1 Browsers

There are several types of browsers in the system:

- Outline browsers: which displays information as text. Indentation shows the hierarchical relation.
- Graphical browser: as seen above, can be used to display hierarchies.
- Property matrix browsers: shows groups of concepts and allows the user to select properties of interest.

2.3.5 Use

CODE4 has been used primarily for software engineering in both research and commercial environments. Among the uses of CODE4 is as a 'knowledge specifier' for capturing requirements and design knowledge. Unlike other knowledge capture systems, it permits the entry of inconsistent statements. During design reviews, such statements can be refined. Therefore, the system acts as a tool to capture initial specifications and allow step-wise refinement of a design. During a trial in a commercial environment the design concepts and requirements were captured 'semi-formally'. i.e. with a precision beyond that of natural language documents but stopping short of very formal description techniques. While natural language doesn't describe a design precisely, formal descriptions are too rigid for all but a few applications. During the trial "human operators did what they are best at, spotting problems, while CODE quickly retrieved and displayed clearly all the knowledge needed in a single unified format."[17]

One of CODE's main trials occurred at Bell-Northern Research, during the design of a workbench for large distributed real-time systems called TELOS (now called ObjecTime). A knowledge base that would eventually include 300 concept descriptors amounting to approximately 200 pages of documentation was built. The main designers of TELOS would confer with a knowledge engineer to enter statements into the knowledge base. Designers would discuss, scribble rough diagrams and try to explain concepts to the knowledge engineer. Group reviews would occur once a week. Inconsistencies, conflicts and unclear terminology could be spotted by CODE queries such as:

- show all concepts added by John since last week,
- show every concept with the property end ports
- show all the things that can be sent a message by an actor
- show how the property components changes down the controller hierarchy

"The resulting displays were sufficiently detailed and restricted that the designers could quickly examine just the knowledge they wanted and could easily spot problems. Again, this reflects the major design philosophy of CODE; to organize and display knowledge sufficiently clearly so that humans can usually spot the problem."[18]

Therefore, with the assistance of a knowledge engineer to create the knowledge base, CODE4 proved a useful tool in managing software knowledge.

2.4 Ontolingua

Ontolingua is a product of Stanford University's Knowledge Systems Laboratory . [www.ksl.stanford.edu]. Its primary purpose is the cooperative development of sharable knowledge bases.

Declarative knowledge, statements and facts about a domain are a critical piece of large scale intelligent systems. Such knowledge properly structured would also prove important for knowledge browsing and sharing. Ontolingua is described as a tool for 'knowledge reuse', allowing the creation and preservation of reusable knowledge bases. Through the construction of knowledge bases in a standard format combined with logic they hope to build knowledge bases for various fields. This would enable a number of applications in science , engineering and defense.

The first step in building a knowledge base is defining a vocabulary specific to the domain. Ontolingua allows for the assembly, customization and extension of vocabularies. These vocabularies are known as ontologies. For them, an ontology is: "An explicit specification of some topic. For our purposes it is a formal and declarative representation which includes the vocabulary (or names) for referring to the terms in that subject area and the logical statements that describe what the terms are, how they are related to each other....Ontologies provide a vocabulary for representing and communicating knowledge about some topic and a set of relationships that hold among the terms in that vocabulary."[19]

The first step is to design the ontology. This begins by choosing a general subject area and making a list of its concepts. Then the library of ontologies can be searched for related terms and the list of concepts and statements can be revised. For example in creating a 'vehicle' ontology :

> **General Concept:** Our ontology will be a general ontology of vehicles which are typically bought and sold through the classified ads. This will include motorized as well as unmotorized vehicles.
>
> **Requirement:** This ontology will need to be able to describe any feature of a car as it is typically describe in classified ads.
>
> **Specific concepts:** A partial list of some of the concepts needed for this ontology are: vehicles, automobiles, make, model, model year and price.
>
> **Other relevant ontologies**: The product-ontology contains some terms such as list-price that will be useful for our ontology of vehicles."[20]

Within each ontology there is a 'frame' structure. There are both class frames and instance frames each of which has slots and values. For example

- Class: vehicles
- Slot: properties of a class e. g. mileage is an important property of a vehicle so create a slot for it.
- Ranges: the range of values , in this case Numbers.
- Value: e.g. 100 000 kilometers

Slots also have facets. Facets represent information about the slots. Usually facets are a constraint on the slot. For example the class 'vehicle' has a slot for list-price. If we want to note that all the vehicles have one price associated with them, we do so in a 'facet'.

2.4.1 Axioms

An axiom is a sentence in first order logic that is assumed to be true without proof. They must be entered in prefix notation. The symbol '=>' indicates logical implication, '⇔' indicates equivalence . The standard boolean operators apply plus 'exists' to indicate existential quantification. Variable names begin with a question mark.

For example, you would represent the statement:

"If any two animal are siblings, then there exists someone who is the mother of both of them, with the axiom:

(=> (sibling ?sib1 ?sib2)

(exists (?mom) (and (has-mother ?sib1 ?mom)

(has-mother ?sib2 ?mom)))"[21]

2.4.2 Instances

An instance is a specific item or term of a class. For example an instance of Ford-vehicle would be a Ford-Mustang. After creating this instance, a user would fill in the slots with the values relevant to a Mustang.

2.4.3 Summary

The final result of an Ontolingua knowledge base entry is displayed in HTML format as seen in the figure below:

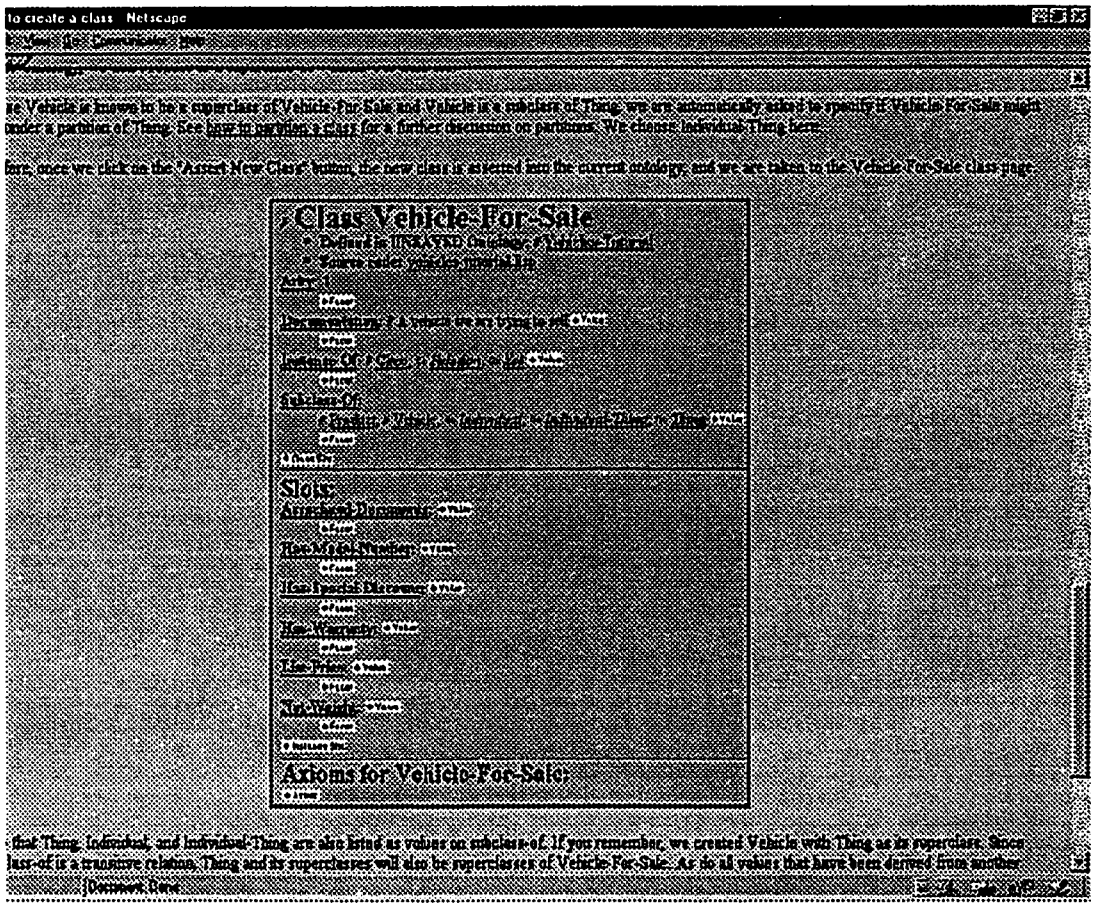


Figure 2.2

Having organized knowledge into this frame structure, with classes and slots and axioms it becomes possible to query the knowledge base using first order predicate logic.

2.4.4 Ontolingua in Context

Ontolingua shares many knowledge organization features with DocKMan. They both use a frame-based scheme with concepts, predicates or slots and facets. However, Ontolingua has a

much higher level of formality and an inference mechanism. It requires users to enter knowledge in first order logic. DocKMan has eschewed this formality for greater flexibility in knowledge entry. The experience of the builders of CODE4 formed one of the reasons for this decision. They note that previous systems are: "too hard to use: they lack flexibility, require too much specialized knowledge and have a long learning curve."[22]

2.5 KnowItAll

Know It All [www.grasp.com] is a commercial product developed by several researchers from MIT's media lab. They identified several of the problems previously discussed including knowledge collection and organization. They indicate that while there are several tools for finding information (search tools, bookmarking utilities) there are few for organizing it into knowledge. Lotus Notes and Web-based intranets have become one of the first tools for building knowledge bases [rather than AI tools] since they recognize that "the creation of new knowledge-in-action is ultimately a collaborative effort based on the construction of shared mental models through conversation...[and] their hypermedia capabilities enable them to establish and preserve a rich, multi-dimensional conversational context."[23]

However, the ideal toolset they envision would give several capabilities vis a vis the information gathered including:

- collection: the initial gathering process should be trivial
- categorization of collected information

- analysis: merging and mixing of facts with respect to varying goals or projects
- source storage and rechecks: the source of the information should be stored as well as some ability to check accuracy.
- delivery: the tool should be able to easily export to other products (spreadsheets, word processors, presentation software).
- integration of information discovery and conversion to other formats.

2.5.1 User Interface

Below, we see a screenshot of KnowItAll's full screen view.

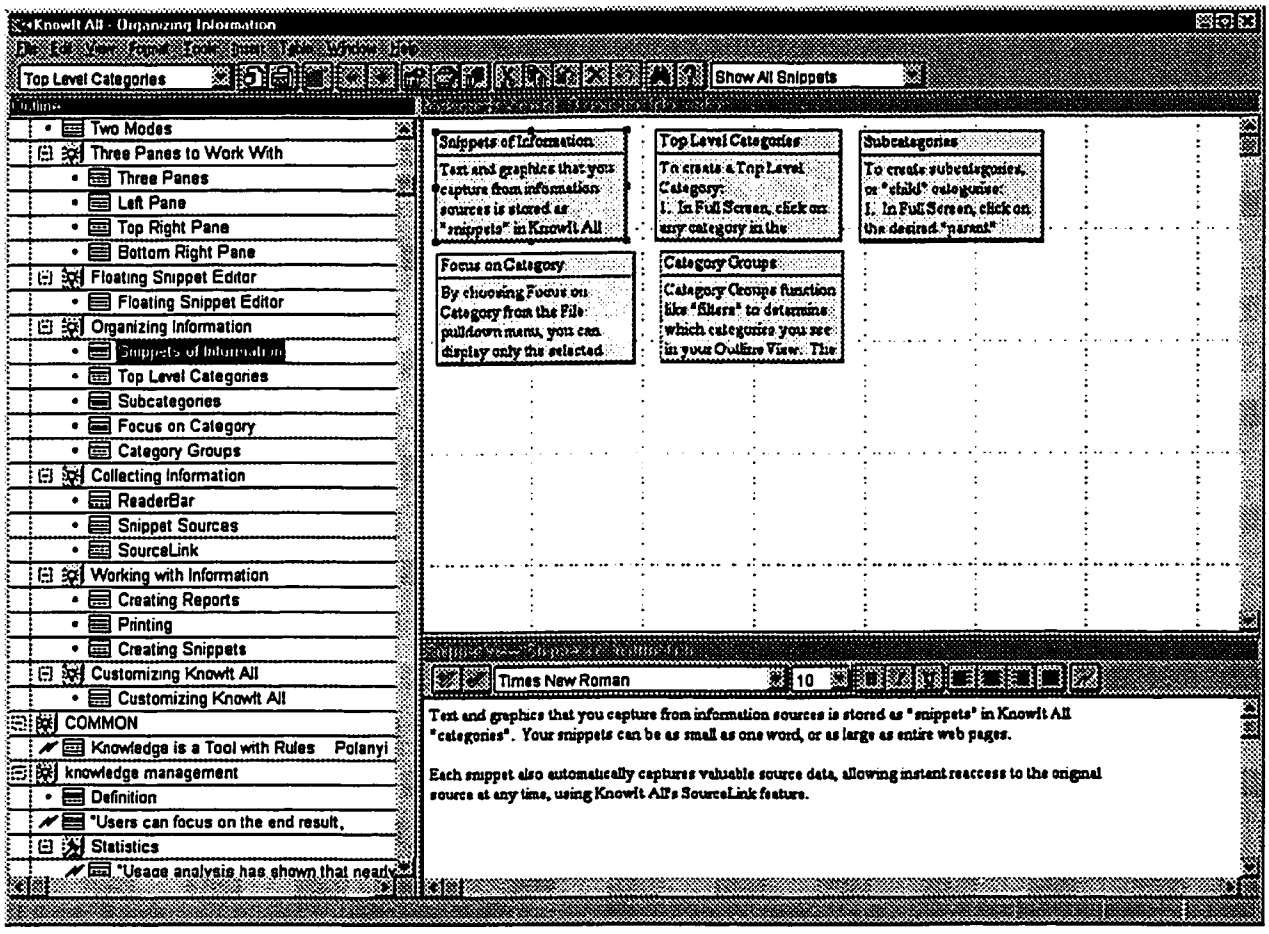


Figure 2.3

KnowItAll contains two modes, the reader bar and the full screen view. The user can 'cut' snippets of information from various sources such as Netscape and Word using the reader bar. They then use the full screen mode to begin organizing their information. The full screen contains three panes:

Left Pane: The Outline View, features the categories users can use to structure information, which include:

- Question
- Concept (equivalent to Dockman's 'subject' hierarchy)
- Person/Place/Thing
- Conclusion

Top Right Pane: Provides three views

- The Notecards View shows the selected category's snippets as a series of 'notecards', each of which includes the accompanying text.
- The Details View is a listing of the selected category's subcategories and snippet titles.
- The Graphical View is a graphical view of the selected category's subcategories.

Bottom Right Pane: The Snippet View: Gives the full text of the currently selected 'snippet'.

2.5.2 KnowItAll in Context

Superficially, KnowItAll contains a striking resemblance to Dockman's knowledge base builder.

It provides both a tree hierarchy and a graphical view of concept or other forms of hierarchies. It allows the user to encapsulate facts or bits of information in 'snippets'. However it is simpler in concept since it:

- does not provide any inheritance mechanism
- doesn't provide as high a level of organization (subject/predicate/complement) for its 'snippets' as Dockman does for its statements
- there is no concept of 'facet' as there is in Ontolingua or DocKMan.

Another notable feature it shows is the ability to 'grasp' information by highlighting it in Web browsers or Word documents. It can then integrate these snippets into the knowledge base.

2.6 Yahoo

Yahoo is the Web's most popular information retrieval tool, being used within the last six months by over 90% of users surveyed in GeorgiaTech's usability study. [24] It offers users a navigation system of categories organized into 'topic' hierarchies. These hierarchies are not based on any defined relationship (such as an 'is-a' relation), but rather on the judgment of expert reviewers. These reviewers search the net and find the 'best' documents and classify them based on a series of keywords. These are then integrated

into a hierarchy. For example, an article on "Research in the cure of tuberculosis of lungs by x-ray conducted in India in the 1950's...might be classified as Health: Diseases and Conditions: Tuberculosis or Regional:Countries:India:Health".[25] Users interested in this particular topic may then drill down through either path to the web page in question. A search engine is also available to allow users to search for documents within a hierarchy (e.g. limiting the search to Regional:Countries:India).

Among the differences between Yahoo and DocKMan , the most important is DocKMan's further structure beyond an initial hierarchy. DocKMan has, as will be seen in greater detail in the next chapter, a further structure of 'statements' and 'facets' ,which enable users to pinpoint individual pieces of knowledge more precisely than Yahoo. The following table summarizes several of the principal differences between the systems surveyed and DocKMan.

Table 2.1

CYC	DOCKMAN
-formalized representation of facts	-flexible representation of facts
-relations 'isa' and 'genls'	-isa and part-of relations
CODE4	
-inheritance hierarchy	-same
-subject/statement/facet division	-same
-greater restriction on user entry	-less restriction
-not web-enabled	-web-based
ONTOLINGUA	
- formal representation	-informal
- axioms in first order logic	-no first order logic
- inference engine	-no inference engine
KNOW-IT-ALL	
-designed for collaborative use	-same
-hierarchy construction ability	-same
-structure information in various categories	-greater degree of organisation (subject/predicate/complement) division
-no support for inheritance	-support for inheritance
YAHOO	
-hierarchical organization	-hierarchical organization
-limited structure: division into a topic hierarchy	-greater structure with subject/statement/facet division.
-document-level granularity	-smaller pieces of knowledge such as 'statements' and 'facets' identified.
-no support for inheritance	-support for inheritance

3 Chapter 3 – The Knowledge Base Builder

3.1 Introduction

DocKMan is a tool to assist both knowledge engineers and the user community in retrieving and structuring knowledge. We believe we have identified a structure for organizing knowledge that can complement document-based information. This structure will be explained and contrasted with that of a document. Some of the advantages and disadvantages of it will be outlined. Following this we will present the interface and operations that allow users to structure their knowledge in DocKMan.

3.2 KB Structure

A knowledge base is a highly structured means of storing and displaying information. Documents already have a certain structure that facilitates knowledge organization and retrieval. Tables of Contents, Chapter, Section and Paragraph divisions all form means of structuring the text.

A DocKMan knowledge base structure is composed of three primary elements:

- hierarchies
- statements
- facets

Each item will be described in turn.

3.2.1 Hierarchy View

Hierarchies are a highly useful means of structuring a domain. A scheme which reflects a standard relation such as 'is-a' or 'part-of' can assist the user in orienting themselves through a body of knowledge. The advantages of subject browseable groups are indicated by a Bellcore study, which cites both library research and user feedback to conclude that: " An interface which reflects the structure of the classification system [or some other means of organization e.g. is-a or part-of hierarchies] essentially provides suggestions to a user about further options to pursue following a search. That is, after a search a user can select from the node labels of the classification system near the search hits to identify the subdivisions that may help further refine the search. The classification system can also be used to restrict searches so as to reduce the computational cost and avoid overwhelming users with spurious information."[26]

In terms of the types of hierarchical displays which should be presented to the user, studies indicate that "versions of tables of contents that allow the user to expand and contract levels of the hierarchy have been shown to decrease browsing times in comparison to stable fully expanded versions."[27]

We have inserted both an expand/contract treeView and a fully expanded graph view hierarchy browser within DocKMan.

We have defined four categories of hierarchies:

- 'is-a' hierarchy: e.g. Chevrolet 'is-a' car
- 'part-of' e.g. Ontario-Canada.

- 'topic' : a subjective evaluation of the order of subjects Often resembles the 'part-of' hierarchy e.g. Yahoo.
- Question hierarchy: a type of 'topic hierarchy'. Previous research by the LAKE group on a sample of 'user help' queries has indicated a hierarchy of question types .

3.2.2 Statements

A statement provides simple facts about the subject. They must have at least two parts, a subject and a predicate[†] and optionally a complement or value that usually corresponds grammatically to the direct or indirect object of the verb. Below we see a series of statements about Object class from our knowledge base about Java.

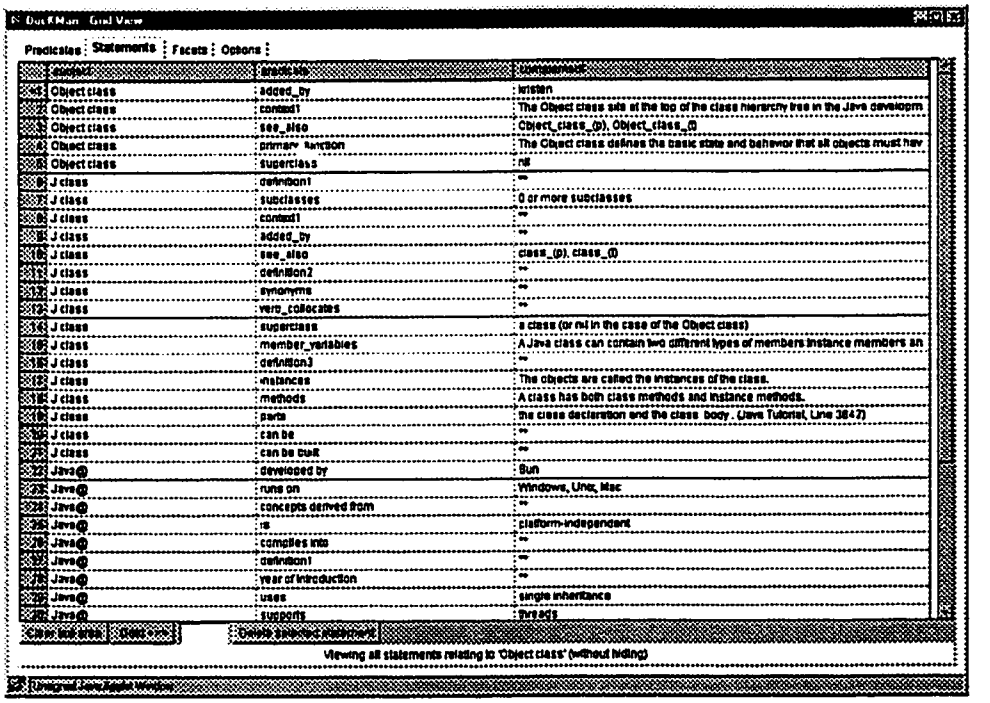


Figure 3.1

[†] Often verbs such as 'is' or 'has' are left out and more meaningful ones substituted.

3.2.3 Facets

In addition to the statement's complement there may be more information or qualifications to convey about the statement. Facets can be divided into several categories:

- linguistic facets: additional indirect objects e.g. Statement: Thread creates new thread groups (with facet) requires: security manager permission.
- annotation facets : source, statement creator e.g. source: Java in a Nutshell p. 89
- more detailed knowledge about the statement: range of values, possible modes e.g. Statement: Car has tires Facet quantity: 4.

Two examples of which are the range of values permissible in the statement's complement, the modality of the statement (e.g. sometimes or always applicable) .

In summary, subjects have statements providing facts about the subjects which in turn can be further qualified by facets. The usefulness of this or a similar structure for concepts was demonstrated by the TELOS experience cited in Chapter 2. [15].

3.3 Inheritance

Within each hierarchy there is a relationship between the different subjects or concepts. This implies that the statements which describe superconcepts can also be used to describe subconcepts. For example in the figure containing statements about 'object class', we see many statements inheriting from superclasses, including the 15th statement about 'member variables'. This implies that the statement 'J class' 'member variables' 'a

java class can contain two types of variables, member variables and instance variables' is applicable both to 'J class' and the sub-class 'Object class'. The predicate name, in this case 'member variables', is associated to a fully inheriting predicate inheritance type. Any statement carrying this predicate will have the same inheritance.

The inheritance type is indicated in a series of radio buttons to the centre right of the main screen.

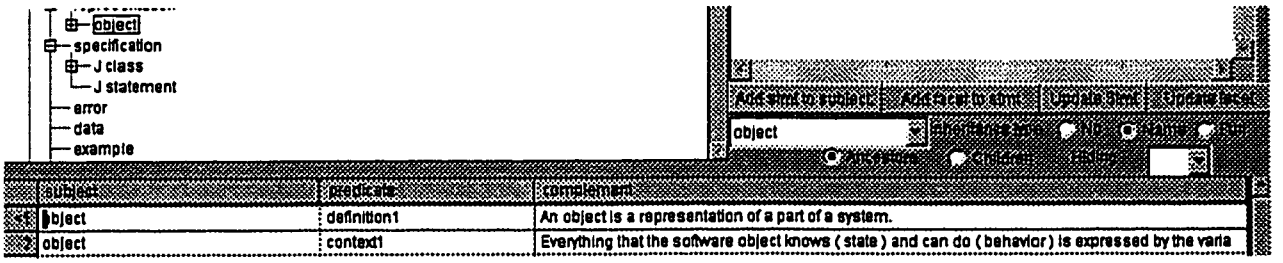


Figure 3.2

DocKMan has three types of inheritance:

- Full: in which the entire statement inherits to the descendants
- Predicate-only: in which only the predicate or verb inherits. This denotes a 'template statement' where a 'child' subject may have a particular predicate, but not the same 'complement' as the parent. Thus, the user should fill in the complement for the children.
- None: statement is not a characteristic of the descendants.

The same inheritance scheme also applies to the facets.

In effect, the knowledge base structure described conforms to that of frames, found in AI theory.[28] Each frame stores information about its subject in one or more statements (or slots) which can inherit to descendants.

3.3.1 Multiple Inheritance

Subjects within concept hierarchies can often have more than one parent. For example, the concept 'Jamie' can have parents 'graduate student' and 'male'. It would then inherit the full and predicate-only inheriting statements from both. Multiple inheritance increases the complexity of the hierarchy considerably. However, its usefulness in describing networks of relationships outweigh this disadvantage. Most object-oriented languages for example (even Java in a fashion via interfaces) have integrated it within their structure.

3.4 Hiding

When users add statements to a subject they can use an existing predicate. They may do this to:

1. Fill in the complement field of a 'predicate-only' inheriting statement.
2. Make the complement more relevant to the currently selected subject

When the user selects the subject again, they may want to see only those statements 'closest' to the subject. For example, when presented with the following hierarchy:

KbTop
    Object

and the statements 'kbTop | definition | ' 'Object | definition | ['|' is used as a separator for statement components.] An object is a representation of a part of a system' with predicate inheritance for definition of:'predicate-only' the following would occur:

KbTop definition would appear in the statement grid with hiding 'OFF' and would not appear with hiding 'ON'. Thus, hiding 'ON' could be called 'overriding.'

Now that we have examined the basic knowledge base structure, we will survey the user interface.

3.5 User Interface

3.5.1 Select Knowledge Base Window

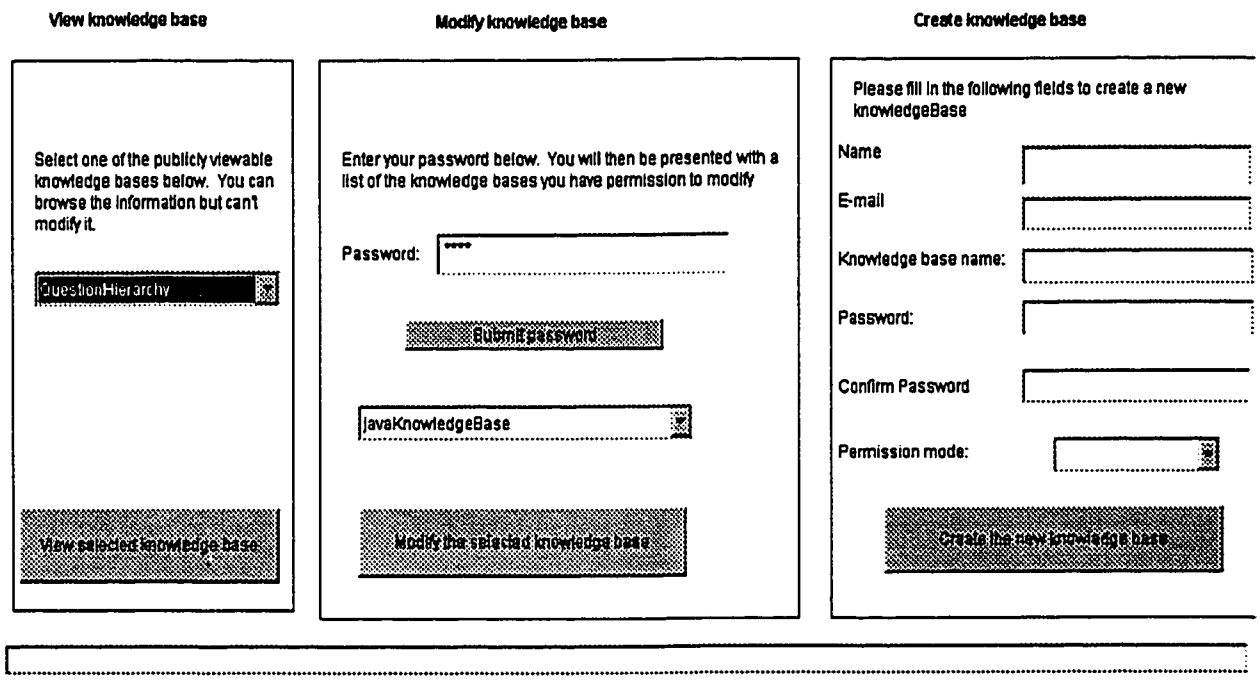


Figure 3.3

Upon loading DocKMan the first window the user sees is the knowledge base selection and creation window. This window is like a file requester : You can either create a new kb or you can view/edit an existing one. The three panels represent one of the following operations:

View knowledge base: Use the menu to select a publicly viewable kb (Read-only access). Then click on 'View selected knowledge base.'

Modify knowledge base: Enter your password. Then use the menu to select the kb for which you have read/write access. Then click on 'Modify selected knowledge base'.

Create knowledge base: Fill in the various fields such as : name, email, kb name, password and permission mode such as private, publicly viewable or publicly modifiable. Then click on 'create new knowledge base'.

3.5.2 DocKMan Main Screen

After selecting the kb in the 'Select Knowledge Base Window', the main DocKMan frame appears.

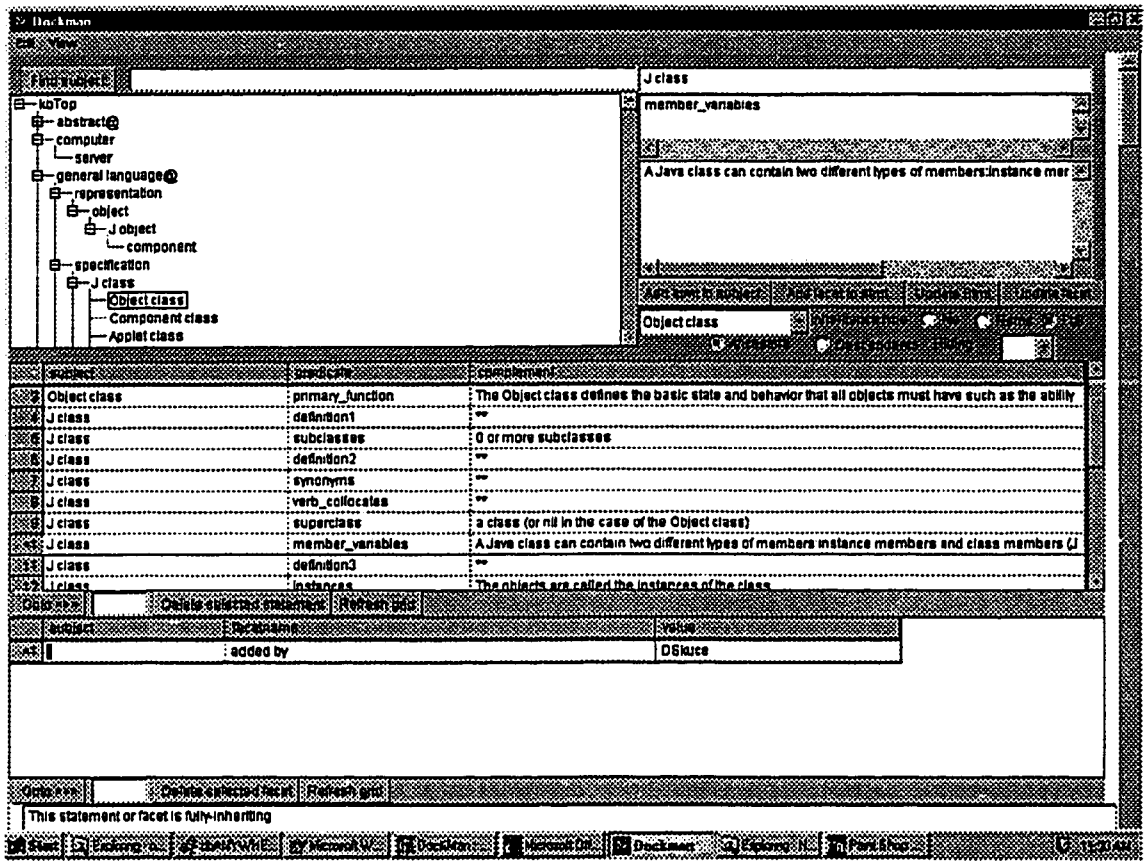


Figure 3.4

The window is divided into several sections:

Tree View: (Upper Left) Textual tree view representing the 'is-a' hierarchy. Click on a subject in the tree view (or graph view) makes it the current subject. Branches can be expanded or contracted by clicking on the '+' or '-' symbols which appear next to non-leaf subjects.

Statement Grid: Below the tree view there is a grid with three columns : subject, predicate and complement. These are statements which fully or predicate-only inherit from the current subject. Click on a statement in the grid make it the current statement.

Facet Grid: Below the statement grid there is a three column facet grid with headings : facetname, value and subject. These are facets associated with the current statement. Click on a facet in the grid make it the current facet.

Enter statement and facet information in the text areas in upper right of the DocKMan window. You can't enter text directly in the statement or facet grid. It must be entered in the text areas described below, a limitation of this type of grid.

Statement/Facet entry area: (Upper right) There are three text areas corresponding to the current statement (or current facet) :

Subject Area: Displays the subject currently selected in the tree.

Predicate / facetname area: Displays and allows editing of the predicate name or the facetname of the currently selected statement or facet.

Complement / value area: Displays the complement or value of the currently selected statement or facet. The two grids in the lower half show the statements and facets respectively.

To conserve space in the DocKMan window, these last two text areas serve a dual purpose:

(a) To edit the Predicate/Complement pair of the currently selected statement. or

(b) To edit the Facet/Value pair of the currently selected facet.

In addition to these 'display features' the following menus and buttons allow you to manipulate the knowledge base .

Edit: (Menu) Insert (new subject), delete subject, rename or choose an existing subject to add as a parent of subject.

Add stmt to subject: (Button) Add a statement to the current subject or to any ancestor in the pull down menu just below this button.

Update Stmt: (Button) Modify current statement

Delete Selected Statement: (Button) Delete current statement.

Add facet to stmt: (Button) Add facet to current statement.

Update Facet: (Button) Modify current facet.

Delete Selected Facet: (Button) Delete current facet.

Hiding toggle: (choice list) sets hiding on or off. A toggle, automatically refreshes the statement grid.

Ancestors/Descendants: (radio button) Allows the user to see inheriting statements belonging to the ancestors or those statements which belong to the children.

View : (Menu) Composed of several choices including:

a) View hierarchy as a graph: launches a 'graph view' window containing the 'is-a' hierarchy. The graph view has hierarchy editing capabilities.

b) View predicate table: Allows the user to view the predicate table and modify predicate names and their inheritance types.

c) View statement and facet tables: Creates a new window, giving the user a view of the current predicate and facet tables.

I will survey each of the elements in Fig. 3.4 in turn, explaining in detail the operations available to the user and the importance of these operations in knowledge management. Any concepts underlying these operations will also be detailed.

3.5.3 Knowledge Navigation

Users can navigate through the hierarchies and statements by a series of clicks. Keeping in mind the relationship 'subjects have statements which in turn have facets', the table below summarizes the navigation actions and results:

Table 3.1

Action	Result
Click on a subject in the tree	Statement grid refreshes. Statements for that subject + all statements with fully and predicate-only inheriting predicates appear.
Click on a statement	Facet grid refreshes. All facets belonging to the statement + any facets that are predicate-only and fully inheriting. These facets must originate from statements which belong to an ancestor b) whose statements have the same predicate name as the current statement.
Set hiding on	Statement grid refreshes. Same result as click on a subject in the tree except that only those fully and predicate-only inheriting that are not equal to predicates found in previous statements.
Set hiding off	Statement grid refreshes. Same result as above except that all statement with fully and predicate-only inheriting predicates appear.

3.5.4 Hierarchy Construction

The are four main actions in building hierarchies:

1) Insert new subject: allows the insertion of a subject not currently found in the hierarchy.

2) Insert/ delete parent links: inserts or deletes a link between two subjects. The child is the currently selected subject. Multiple links can be selected by using the Ctrl button.

3) Delete subject: deletes the subject and reconstructs the tree.

4) Change subject name: renames the subject. The name cannot be currently in use in the hierarchy. Users cannot add or change the name of a subject to one currently found in the hierarchy table. Should they attempt to do so, they will be asked if they wish to create a new 'sense number' for the subject.

Both the insert subject and change subject name dialog boxes are straightforward, with a text field prompting the user for a string and a submit button. The delete box offers three options as seen below:

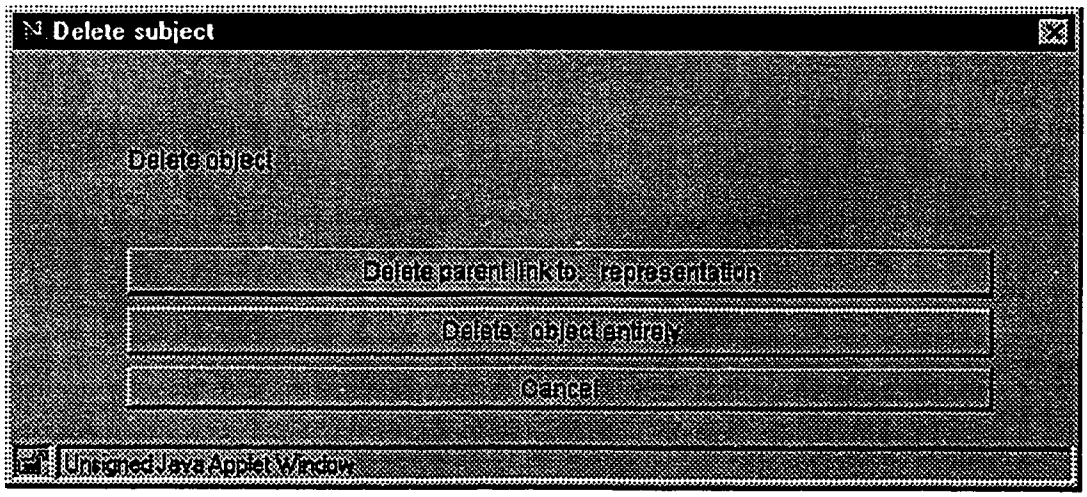


Figure 3.5

The first option allows the user to delete the link between the selected subject and one of its parents. In the case where a subject has several parents, the immediate parent of this particular instance of the subject name is shown. The second option deletes the subject entirely.

Finally, the 'add parent link' list offers the user a choice of all the subjects they can have as a parent of the current subject. By pressing the 'Ctrl' key, a user can select multiple subjects to add as parents of the current one. Below this list is 'delete parent link' which lists all the selected subject's current parents. The user can select one or more of these subjects and delete the link between the selected subject and the parent.

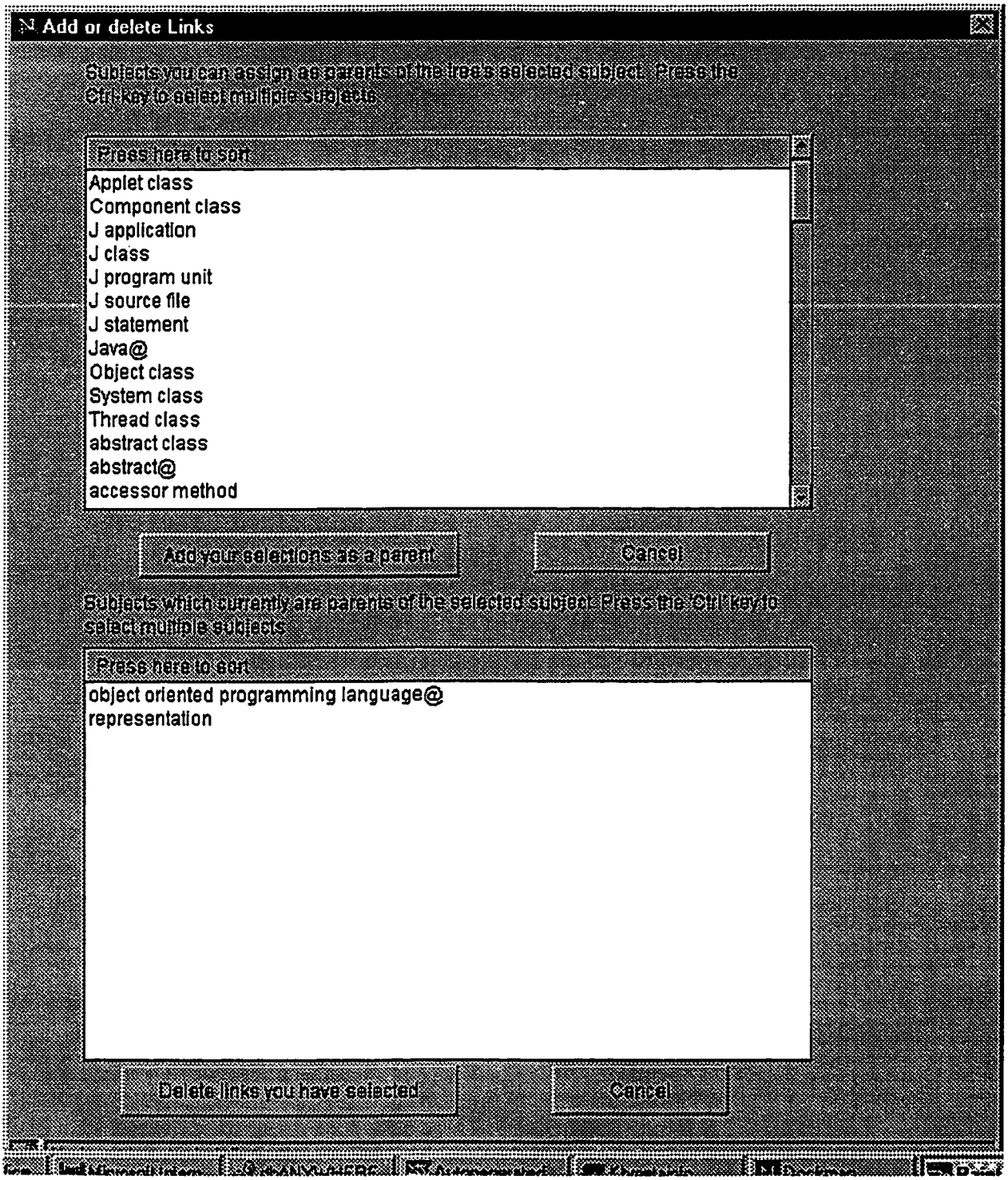

Add or delete Links
Subjects you can assign as parents of the tree's selected subject. Press the Ctrl-key to select multiple subjects.
Press here to sort
Applet class
Component class
J application
J class
J program unit
J source file
J statement
Java@
Object class
System class
Thread class
abstract class
abstract@
accessor method
Add your selections as a parent
Cancel
Subjects which currently are parents of the selected subject. Press the Ctrl-key to select multiple subjects
Press here to sort
object oriented programming language@
representation
Delete links you have selected
Cancel

Figure 3.6

3.5.5 Graph View

In addition to the treeView, users can also navigate through the knowledge base via Dockman's graph view.† Unlike the tree, which must repeat a subject name for each of its parents, the graph has one 'node' per subject. Parent-child links are represented as lines or edges. Therefore, this representation more accurately reflects the underlying structure of the knowledge base concept hierarchy than the tree. Below is a portion of the graph view for a Java knowledge base.

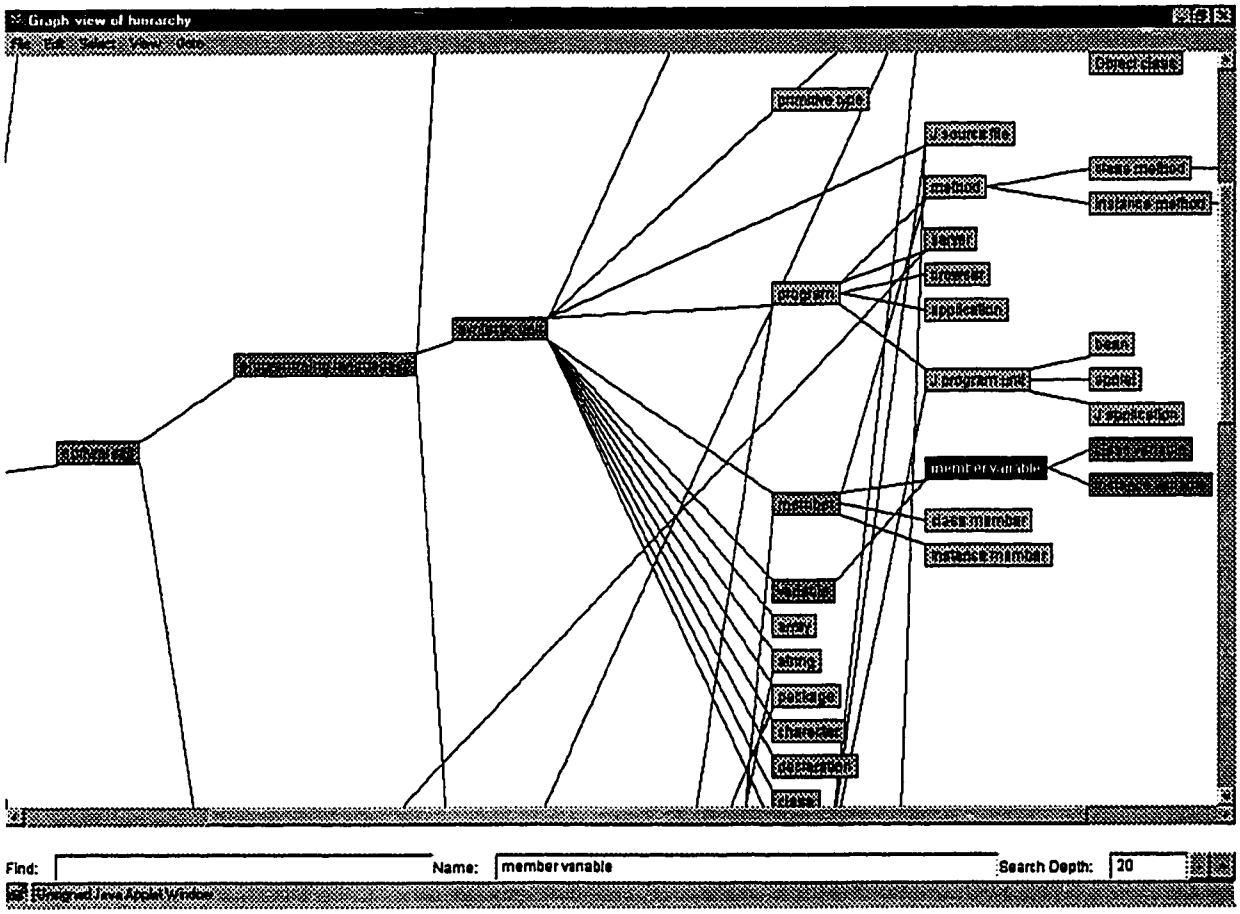


Figure 3.7

† The Graph Viewer was developed by Wratko Hlavina and integrated into the knowledge base builder with his cooperation.

As an aid to navigation, the selected subject is darkened. The children are coloured in mauve, the parents in pink and all other ancestors in red. This allows the user to clearly see any multiple inheritance relationships. The graph also offers a variety of navigation and concept manipulation features.

3.5.6 Adding Statements

In order to add a statement, the user begins by specifying a predicate in the predicate/facetname area. Optionally, a complement can be included. Then they can:

1) Choose either the current subject (the default) or one of its ancestors from the choice list below the 'Add stmt to subject' button.
2) Press the 'Add stmt to subject' button

After pressing this button one of three dialog boxes appears.

If the predicate already exists anywhere in the predicate table, and the user has not changed its inheritance:

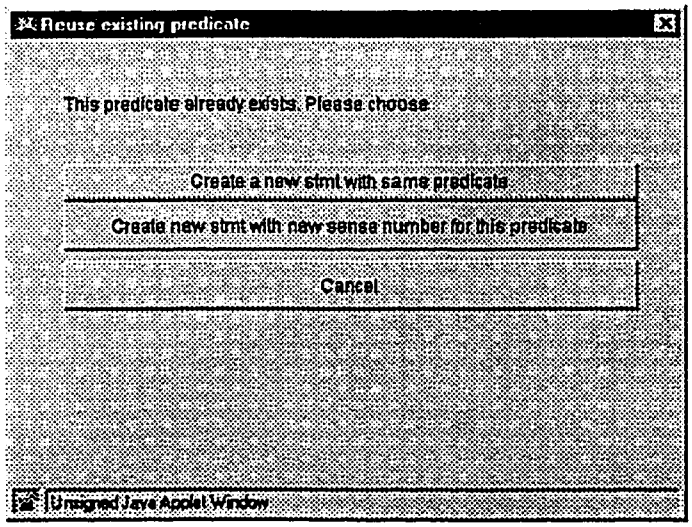


Figure 3.8

The first two buttons are meant to signal the user that they should think through the 'sense' they wish to attach to the predicate before proceeding. For example, a user wishes to enter a statement with subject 'racecar_driver' and predicate 'drives'. In another branch of the hierarchy, the user enters a statement with subject 'cowboy' | drives | cattle. The user should then create a new sense number for this very different meaning of the predicate drive.

If the predicate already exists in the predicate table, and the user has changed its inheritance:

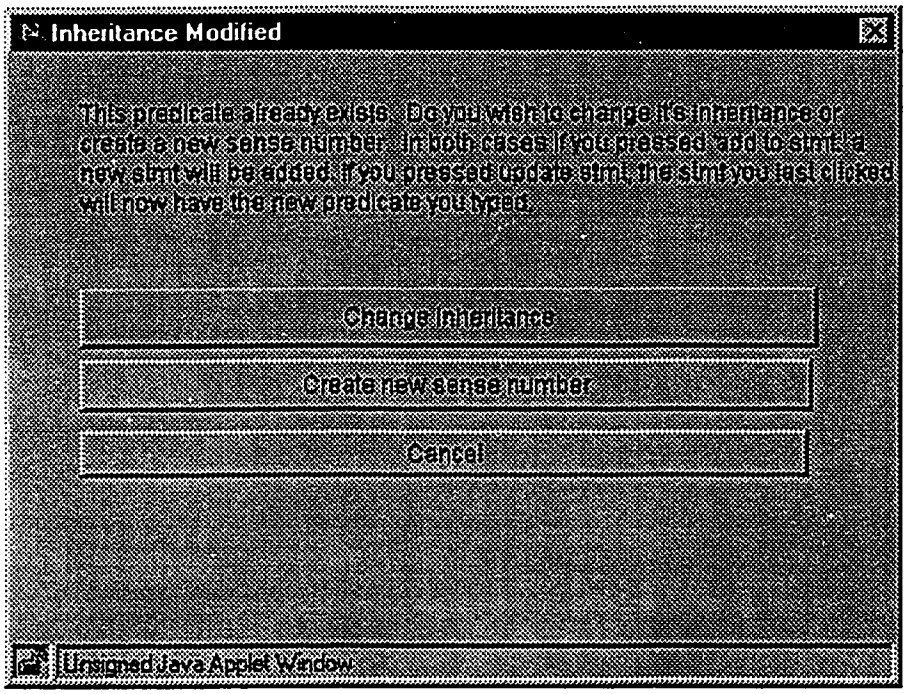


Figure 3.9

Finally, if the predicate is new, a simple confirmation dialog appears, asking the user if they want to create a new predicate.

3.5.7 Updating Statements

When updating individual statements, the user need only click on a row. They can edit the predicate and complement as needed in one of the two text editing areas and press the update row button. Then, one of the three dialog boxes presented in 'Add Stmts' will appear.

3.5.8 Adding and Updating Facets

The same actions as presented above apply. A statement must be selected before a facet can be added. A facet must be selected before it can be updated. This requires clicking the mouse on the facet the user wishes to update.

3.5.9 Predicate Table

In addition to these features, the users can change and modify both the name and inheritance of a predicate directly through the 'view predicates' option of the menu. On doing so, the following view (Figure 3.10) of the predicate table appears.

After making changes to any of the elements of the table, the user can press the 'save predicate changes' button to commit changes to the database.

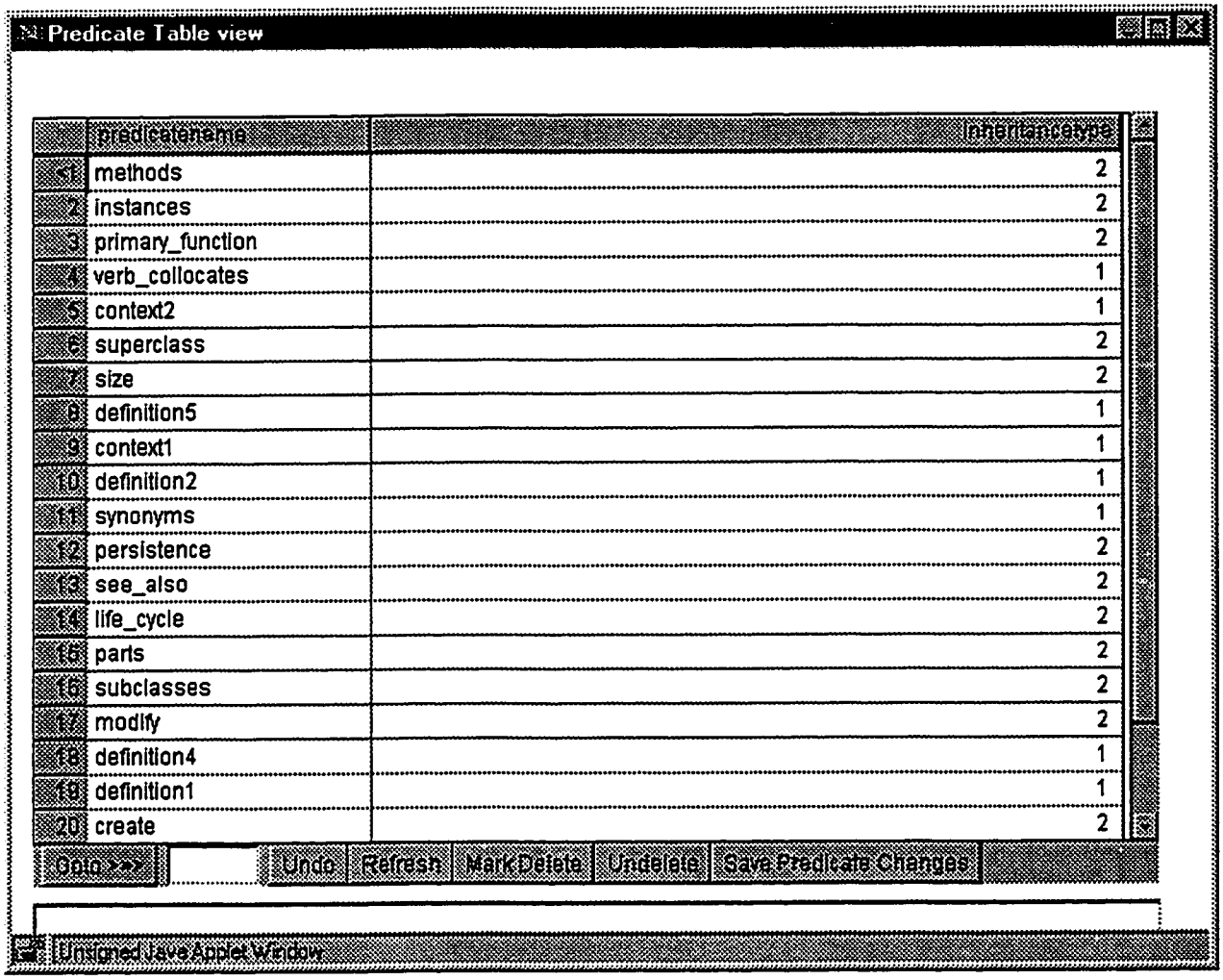
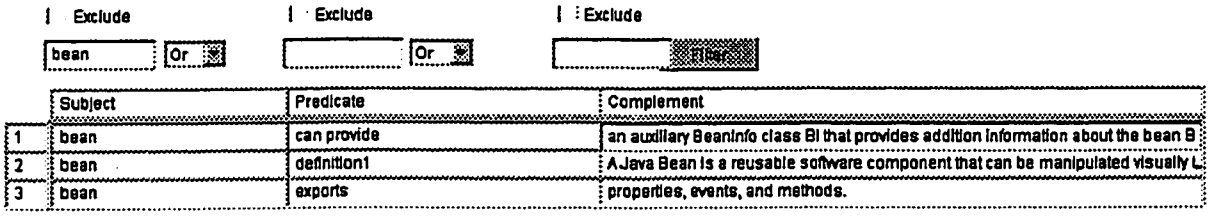


Figure 3.10

3.5.10 Sorting and Filtering of Statement and Facets

By clicking on the button 'view statements in a separate window', the user can see the statements in a separate grid which allows sorting and simple boolean searches. They can also see the entire statement or facet table and conduct similar operations. For example, if we wanted to know all those statements relating to 'Java beans', we could select in the 'view' menu the option 'Sort and Filter entire statement table'. Sorting on the subject column and having discovered the term 'bean', we would then use this name in the subject filter. The figure below shows the results of this operation.

Sort: Click on the column title , pressing the Ctrl key allows multiple sort. Filter: Enter text in the area above the column.Select one of OR or AND. Checking the Checkbox EXCLUDES rows having that string in the column. Please note, there is no dynamic refreshing of this grid, when you change subjects



	Subject	Predicate	Complement
1	bean	can provide	an auxiliary BeanInfo class BI that provides addition information about the bean B
2	bean	definition1	A Java Bean is a reusable software component that can be manipulated visually L
3	bean	exports	properties, events, and methods.

Figure 3.11

4 Chapter 4

4.1 Introduction to the Technology

This chapter describes the implementation of the DocKMan system. The software environment in which DKM operates will be surveyed. We will also examine some of the problems and restrictions of this environment. Following this, we will present the general architecture of the system and briefly detail the role of each class within it.

4.2 Technology Choice

There were several competing technologies that could have been used to publish the contents of a database to the web. The requirements for the system included:

- operate in a Web browser environment
- operate on a variety of platforms, including PCs and Suns
- offer a variety of components, including graphs, trees and grids for display and data manipulation.
- easy binding of the database tables to on-screen components

An alphabet soup of competing technologies promised to fulfill these requirements. These included Microsoft's Active Server Pages with ActiveX Data Objects, Advanced Data Connector, ActiveX Controls and a variety of Java development environments,

including our final choice, Symantec Visual Cafe for database development. Each of these options will be briefly described, and the choice of Symantec's environment justified.

4.3 Microsoft Technologies for Web Database Development

Active Server Pages and ActiveX Data Objects (ADO) combined allows developers to build "scripted HTML pages".

ADO used within the scripting allows access to databases and a display of the results on an HTML page. Unfortunately, this approach is only useful to "present static client information, and data aren't updateable. For example, the client might receive a text table, but not be able to interact with or change the table data."[29] Since one of our requirements was the ability to manipulate data on the client, this technology was excluded. The second Microsoft technology for Web-based database access is Advanced Data Connector (ADC). It allows data onto the client computer, where it can then be manipulated and updated. It can also be bound to ActiveX Controls. These are software components(e.g. grids and trees) developed by independent software vendors. They can be inserted into HTML pages, using the ActiveX Control Pad.

All our requirements seemed to be fulfilled by this solution which provided the ability to insert off the shelf components and bind them to data. However several disadvantages emerged, including:

- no support for Netscape Navigator

a) With Microsoft Internet Explorer not available for UNIX it would prevent users from accessing DocKMan from UNIX platforms.

- no easy binding of components such as trees or grids to data
- no fully Integrated Development Environment (IDE) to connect the disparate elements (Active X controls, Active Data Connector (ADC)).
- poor documentation
- more recent than JDBC (therefore potentially more bugs)

Having discounted this solution, we examined various Java tools to enable access to databases. While there were several on the market including Borland JBuilder and Visual J++, we settled on Visual Café, which was well-reviewed.

4.3.1 Software Environment

DKM relies on several pieces of software for its operation. Figure 4.1 illustrates the relationship between DKM and its components.

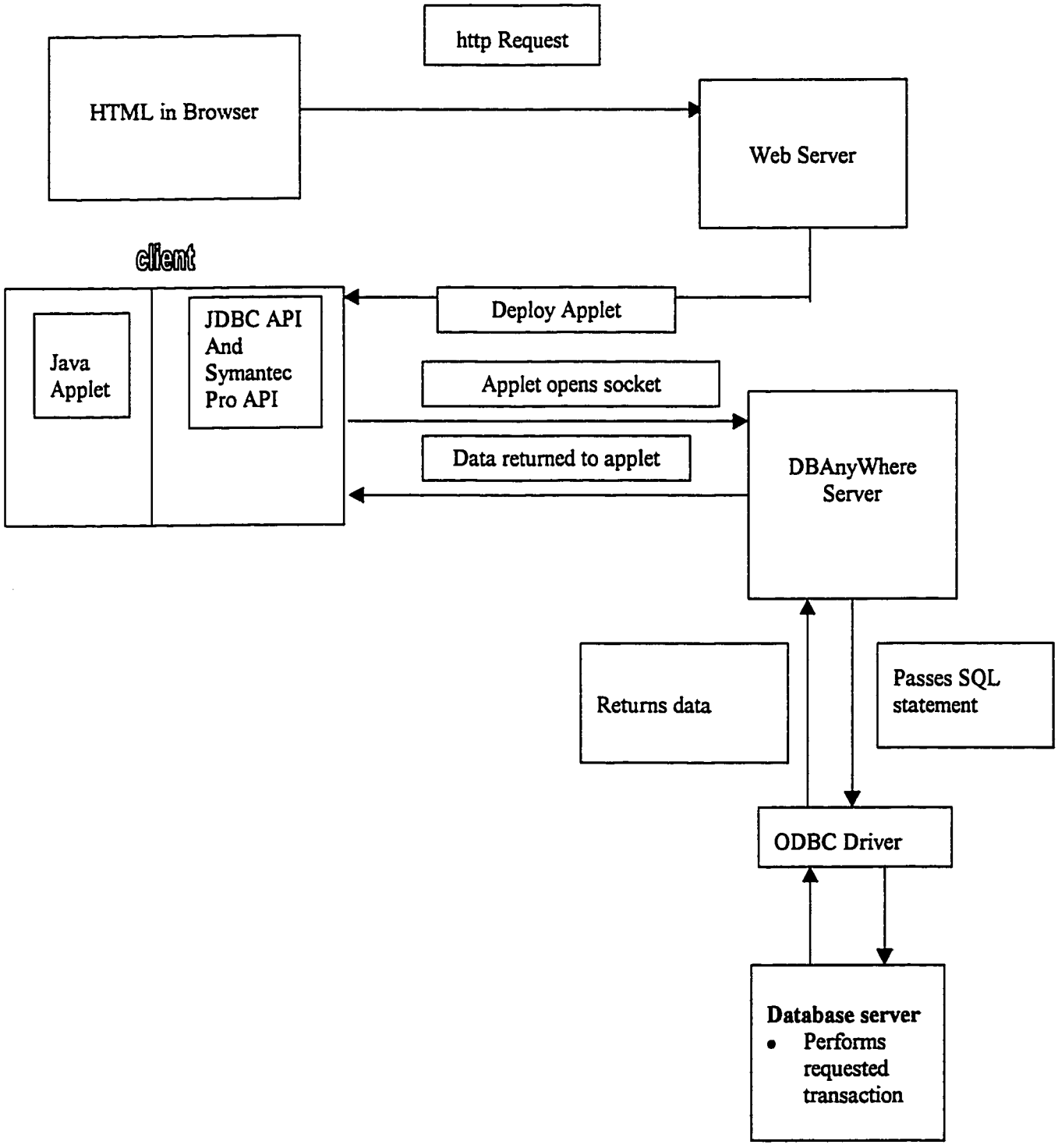
As the diagram illustrates four pieces of software are crucial to the correct operation of DKM. These include:

- Java enabled web browser
- Symantec's dbAnywhere middleware server
- Database server

- Web server

Next we will explain the role and drawbacks of each component .

Figure 4.1



4.3.1.1 Web Browsers

Two browsers dominate the market, Netscape and Microsoft Internet Explorer. Both browsers have the ability to run Java applets within their environment. Unfortunately several problems have arisen over the course of development that make us doubt this claim. During various iterations, the applet has been fully functional in various combinations of

- Symantec Visual Café's Applet Viewer
- Netscape Communicator 4.04 with the JDK 1.1 patch
- Microsoft Internet Explorer 4.0

'Re-development' was required at various stages to ensure proper operation in a browser. For example, the applet's treeView control did not scroll properly in Netscape. The reasons behind these failures lie in the current lack of standards for a Java virtual machine and the new state of Java tools. One Java guru notes: "Tools for integration of Java, object databases and the Web are in a primitive state in 1996, and most corporations are unable to implement client/server systems in Java."[30] Sun Microsystems has proposed turning Java over to a standards body . Recently, Sun has developed its own virtual machine that is intended to be an industry standard, replacing individual virtual machines such as Netscape's and Microsoft's. Unfortunately, Microsoft prefers to write their own version.

4.3.1.2 Sun's Java Runtime Environment

Activator is Sun's implementation of the Java Runtime Environment (JRE). Users can specify this environment as the default in their HTML pages rather than Netscape's or Internet Explorer's Java Virtual Machine (JVM). The activator is fully JDK 1.1 compliant and a new version will be shipped as different versions of the JDK emerge. Among the reasons Sun cites for the creation of Activator are the differing implementations of the JVM in various browsers. For example, Sun states that

> "The JDK 1.1 features in Communicator 4.0.4 are still not complete or fully compliant with the Java compatibility kit [JCK] test suite."[31]

Among the features which Netscape acknowledges aren't compatible are

- support for multiple JAR files
- some thread methods

4.3.1.3 Web Server

We currently use Microsoft Windows NT Workstation's Peer Web Services as a web server. This is a subset of its Internet Information Server (IIS) product. The web server requires that all class files be found in a directory called InetPub/wwwroot . or one of its subdirectories.

4.3.1.4 DbAnywhere

Symantec's DBAnywhere serves as a 'database middleware' that accepts database requests from a client (in this case DKM a Java applet). These requests take the form of SQL statements. DBAnywhere accepts all of ANSI SQL except the JOIN clause. The figure below illustrates in detail the connection between dbAnyWhere and the database.

The browser runs the Java, JDBC and dbAnyWhere classes. The middleware is responsible for handling transactions to and from the database. Finally, the dbAnyWhere drivers interact with vendor specific database client libraries which speak to the database server.

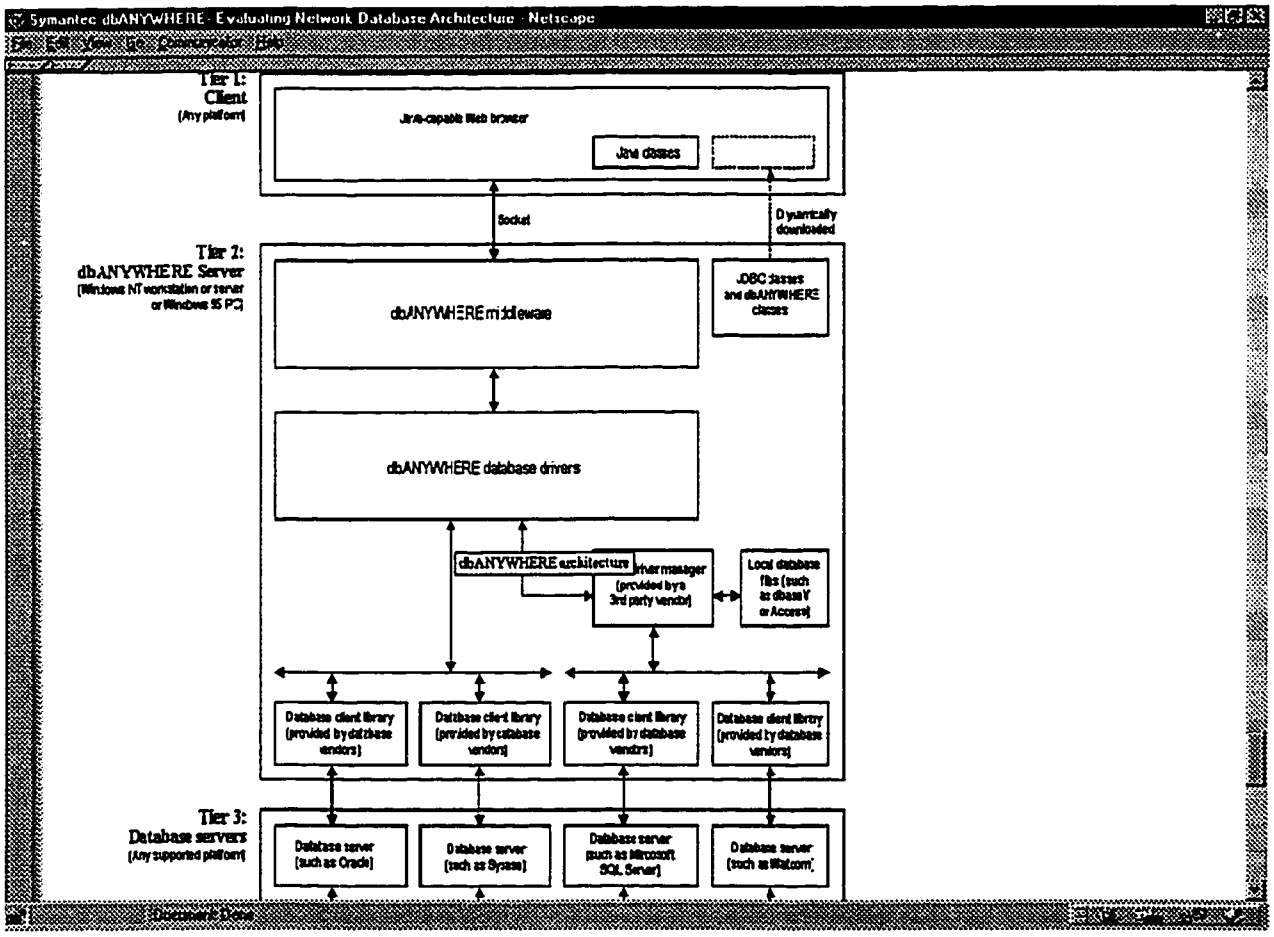


Figure 4.2 (Source: Visual Café for Java Database Development, Documentation Edition)

Several reasons motivate the choice of this architecture:

- Enhanced security: the middle tier can control and limit client access.
- Bandwidth management:
  a) data caching enables the server to satisfy requests directly from its cache it the data has been recently accessed.
  b) Results of large queries cached in dbAnyWhere

The server can also handle multiple connections running in separate threads. DbAnyWhere also provides an API for accessing the database. While not having the vendor neutrality of JDBC, it helps hide the granularity of many of the JDBC methods. There is more power per line of code than JDBC. Unfortunately, we recently discovered that Symantec's API has different behaviour with various database servers. For example, when converting from a Sybase server to Oracle, we notice it generates four SQL queries for every one launched from the client. Symantec has confirmed this in one of their online help entries where they note: " Four sql statements are generated for every query when using Oracle" . Future use of Oracle would be contingent upon converting the Symantec API calls to JDBC, substituting resultSet objects for Symantec's RelationView objects.

The dbAnyWhere server assists in management of the database through the use of optimistic concurrency control. This is where more than one user can modify some database object at once. Inconsistencies are resolved at a later time. For example, a lock (on a table or record) is optimistically assumed to be granted. If the lock is refused, the

object returns to its original state. If it is granted, the object remains in its changed state. The benefits of this approach are that response time is immediate, however you cannot be sure that the action will be performed until the lock has been accepted or rejected.

4.3.1.5 Database Drivers

JDBC is a programming level interface which allows communication with databases in a uniform manner. In this, it is the Java equivalent of Microsoft's ODBC (Open Database Connectivity). There are four types of JDBC drivers:

Type 1 – JDBC/ODBC bridge

Type 2 – Native-API, Java driver

Type 3 – Network Protocol, All Java driver

Type 4 – Native Protocol, All Java driver

Only the type 2 driver will be explained, since this was the type used in the implementation. There are three tiers, the JDBC client and driver, the 'middleware' and the database, each will be explained in turn . The driver, executing on the client,

- passes SQL commands over the network to a JDBC server
- receives data from the server
- manages the connection

4.3.1.5.1. JDBC Server

- Manages multiple database connections
- Packages data to the client
- Connections are made either

a) Via a database vendor's client library

b) ODBC

4.3.1.6 ODBC Drivers

When ODBC is used to achieve a database connection, one ODBC driver must be set for each database being connected to. Drivers are set in the using the 'ODBC 32-bit' Icon in the Control Panel of Windows NT. The following dialog then appears:

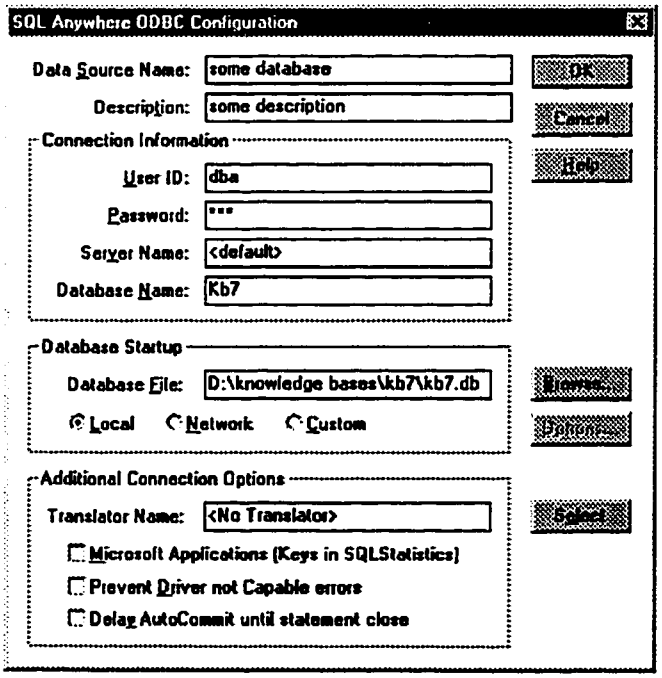


Figure 4.3

After the DocKMan administrator has created a new database in SQL Anywhere, he need only specify a name for the database, the user dba and password sql, as well as the full path of the file. Then the driver name can be entered in the 'driverName' table of the kbmetainfo database and is ready for use.

4.4 Table Structure

Before detailing the implementation, the basic structure of the database it uses must be explained. In Figure 4.4 below, we see both the kbmetainfo database which stores information about system users and the knowledge base database tables .

Figure 4.4

KnowledgeBase - Database Tables

Hierarch
PK - subject
PK - subconcept
Stores the edges of the graph in a hierarchy

Subject
PK – Subject
- SenseNo
Stores the subjectName and the number of times it has been used.

Statement
PK - stmtno
- subject
- predicate
- complement
Stores statement information

Predicate_inheritance
PK- Predicatenumber
- Predicatename
- Inheritancetype

Stores the name and inheritancetype of a predicatename

SenseNumber
PK – Baseword
- Senseno

Stores the predicate name as a baseword. Also, stores the senseno (number of times the word has been used in different contexts)

Facets
PK - facetno
FK - stmtno
- facetname
- value
- subject
Stores information pertinent to facets. The subject is included in order to easily track and display the originating subject.

Facetname_inheritance
PK - FacetnameNumber
- Facetname
- inheritanceType

Stores the name and inheritancetype of a facetname.

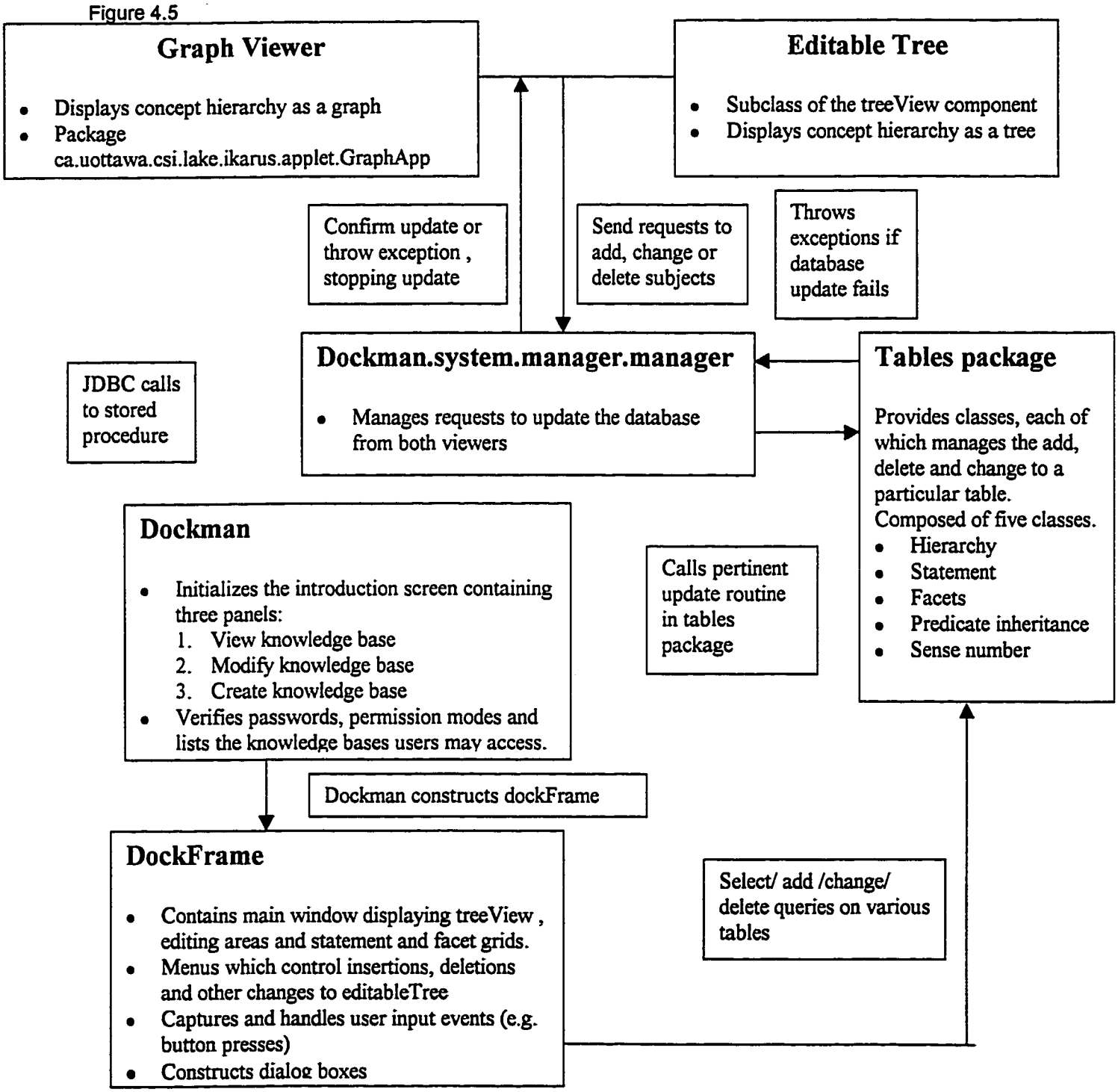
Facet SenseNumber
PK – Facetname
Senseno

Stores the facetname in the facetname field, and the number of times it has been used in the senseno field.

4.5 Dockman Class Architecture

The diagram below illustrates Dockman's class hierarchy. I will elaborate on the role of the most important of these classes:

Figure 4.5



4.5.1 Dockman

The main applet class DocKMan begins by establishing a session with the dbAnyWhere server. It provides a full URL and database driver information to the server. Once it has initialized the main applet window it presents the user with three choices, viewing, modifying or creating a knowledge base.

Viewing:

A stored procedure is invoked via a pure JDBC call, since Symantec's dbAnyWhere API couldn't call these procedures. It returns a list of the knowledge bases which have 'publicly viewable' or 'publicly modifiable' permissions. The user then selects a knowledge base from one of the list and his selection is mapped to a driver name via a call to a second stored procedure. The driverName is then furnished to dockFrame.

Modifying:

The user supplies a password. The remaining steps are the same as the viewing case.

Creating:

The user supplies a knowledge base name, password and identifying information These are saved in the kbName , password and userInfo tables respectively. A stored

procedure is then called to assign an ODBC driver name to the knowledge base and this name is furnished to dockFrame.

4.5.2 dockFrame

- initializes the three main user display components treeView, statementGrid, facetgrid
- updates the components depending on user input
- allows for the entry of statements, facets and subjects.

We saw in Chapter three some of the navigation and statement/facet entry operations a user could engage in. Now the code that executes on various user actions will be explained.

4.5.3 Navigation Actions

4.5.3.1 Subject Selection in TreeView

When this action occurs, the editableTree class (described in detail later) fires a changeItemSelection event. This event is captured and handled in the treeView_itemStateChanged method. Other views (such as the graph) are also alerted as to the change. (The discussion in the Model-View-Controller architecture section provides more details). The following pseudocode describes the handling of this event.

Find the ancestors of the selected subject

Set the choice list of ancestors

Update the statement grid

The first and third items correspond to the ancestors and modifyStatementGrid methods which will be surveyed in subsequent pages.

4.5.3.2 Modify Statement Grid

This method displays the statements for the chosen tree subject and any superconcept's statements which are predicate-only or fully inheriting. It also implements the notion of hiding seen in Chapter Three, by excluding superconcept statements that have the same predicate name as a 'lower' predicate.

The method begins by issuing a SELECT on the database that captures all statements associated with the selected subject. Then, it retrieves the superconcept's subjects and stores them in a hash table where the subject is a key and the level or distance form the subject is a value. It then begins to issue a select for each superconcept from the lowest to the highest level. This order is crucial since it allows hiding of higher level statements. A separate select is required for each subject due to an unfortunate restriction of the ODBC driver. It limits the length of SQL statements, thus preventing us from issuing long selects. The following pseudocode shows how each statement captured by the select is processed:

```
Retrieve the statement's predicate and it's inheritance type
IF the inheritance type is Predicate-only OR fully inheriting
IF the predicate has NOT previously been used (hiding)
OR hiding is false
```

OR the level has not changed (multiple parents)
THEN add the statement to the display grid

The underlying data structure, a "relation view" accumulates each of the statements which will be displayed in the grid. It constitutes a 'virtual table' . It sees every new statement attached to it as a 'new record' and labels it as such. This is despite the fact the record currently exists in the database table. A future implementation of in place editing in the grid would require comparing the previous virtual table to the changes made in the grid's cells. Then, the modified rows would have to be rewritten to the database.

Action: Click on a row of the statement grid.

This leads to invoking of the statementgridRowSelected method, which sets the current statement number to the row clicked and invokes the modifyFacetGrid method.

4.5.3.3 ModifyFacetGrid

This method implements facet inheritance. This is defined as any facet which belongs to a particular statement and any facets which originate from statements

- belonging to an ancestor
- having the same predicate name as the current statement
- which are inheriting

For example, consider the hierarchy

computer
    server

and statements 'computer runs programs' and 'server runs server programs'. If the first statement has the facet 'src: Jamie' and facet name 'src' is fully inheriting, then upon clicking the second statement, the facet 'src: Jamie' will appear in the facet grid.

The implementation begins by selecting all those facets which have the statement number of the selected statement as a foreign key in the facet table. Hiding is again implemented as it was in modify facet grid, except the 'facetname' substitutes for the predicate name.

We continue by looping through the ancestors of the selected subject, a) selecting statements that have one of these ancestors as a subject and have the same predicate name as the original statement.

If any records are returned :

- capture the statement numbers
- select the corresponding facets.

In addition to the navigation functions, dockFrame responds to user requests to add, update and delete both statements and facets.

4.5.3.4 Adding and Updating Statements

Users add statements to a subject. They can select the current tree subject or any ancestor of the current subject. Once the add statement button is clicked a dialog box appears. As seen in Chapter three, there are three dialog boxes based on two factors:

a) the predicate state (one of: new or exists)

b) a user's action, which can be one of:

created a new predicate

modified the name to that of an existing predicate

entered an existing predicate, but changed it's inheritance type.

The following table illustrates the various scenarios that can occur.

Table 4.1

No.	stmt action	sub-action	Predicate state	User feedback	Pred table	stmt table
1.1	add	None	New	Confirm intended new pred y/n	add new pred if confirmed	add new stmt
1.2	add	not modif inheritance	Exist	Confirm reusing exisiting pred (y)	no effect	add new stmt
1.3	add	not modif inheritance	Exist	OR: create new sense of this pred	add new pred with new sense no	add new stmt
1.4	add	modif inheritance	Exist	Confirm: change inh	change inh mode	add new stmt
1.5	add	modif inheritance	Exist	OR: create new sense no	add new pred with new sense no	add new stmt
2.1	update	modify pred	New	Confirm intended new pred (y)	add new pred	update stmt
2.2	update	modify pred	Exist	Confirm reusing exisiting pred (y)	no effect	update stmt
2.3	update	modify pred	Exist	OR create new sense no	add new pred with new sense no	update stmt
2.5	update	modify inheritance	Exist	Confirm: change inh (y)	change inh mode	update stmt
2.6	update	modify inheritance	Exist	OR create new sense no	add new pred	update stmt

For example, when the user clicks on 'add statement', the method inheritanceChoice() is invoked. It calls methods accessing the predicateInheritanceTable() to determine if

- the predicate entered by the user currently exists

- if the inheritance mode entered by the user is different from the one currently found in the table

Depending on the results it creates one of three dialog boxes.

- create new predicate
- reuse existing predicate
- create new sense no.

These boxes then prompt the user with some of the questions in the 'user feedback' column of the preceding table. Finally, the predicate and statement tables are modified based on this feedback.

4.5.4 Database Interaction Classes

A group of classes was created in order to interact with the database.

Some of the methods associated with each class and their usefulness to other system components will be explained.

4.5.4.1 Hierarchy

As a reminder, the hierarchy table consists of the subject and subconcept fields. These correspond to edges in the hierarchy class. The class manages the following functions:

- Inserting new subjects: places subjects that don't currently exist in the hierarchy table.
- Insert/delete parent/child link: inserts or deletes a line in the hierarchy table consisting of a currently existing parent subject and a 'selected_subject' as the child
- Deleting subjects
- Changing the subject name

The hierarchy class in accomplishing these actions must enforce a number of rules to maintain the logical integrity of the database table. Among these are:

- existence Checks
- invalid character or empty string entry
- circularity checks

For each of these an exception is thrown should the checks fail.

4.5.4.2 Existence Check

On inserting new subjects, the string entered by the user must be checked. If it exists in the database it is not a new subject. The user then has the option of choosing to create a new sense number for this subject or to cancel the operation. Also, a check is made for invalid characters (the ^ or ' characters).

4.5.4.3 Circularity

The hierarchy built by a user has the characteristics of an acyclic graph. The introduction of cycles into it is logically inconsistent . In order to avoid these we only allow the user to introduce new subject names or to insert a 'link' between

- a selected subject and
- one of the currently existing subjects EXCEPT for

  a) the selected subject

  b) any descendant or immediate parent of the subject .

  While users may initially think in 'circular patterns', this restriction is meant to force users to disambiguate their mental model of the hierarchy and make it more precise.

4.5.4.4 Insert Parent Child Link

The 'insert parent child link' method builds the list of subjects that become the parents of the currently selected tree subject. The pseudocode below describes the implementation:

Build an associative array[†] with parents as the key and all children as the value.

[†] An associative array has a single key that can hash into multiple values. It is built by combining the hash table structure with a vector object in the value field.

Using the associative array

Get each of the children of a subject

For each Child

Add to a vector

Call the method again using 'child as the new parent

[This allows accumulation of all the descendants of the topmost parent]

4.5.4.5 Deleting Subjects

In DocKMan every subject has at least one link, these must be removed or reassigned to descendants when deleting subjects. In order to do this, each of the children of the subject are reassigned to all of the subject's parents (This ensures that the children are not also 'sawed off') Then the subject is deleted wherever it occurs in the table.

4.5.4.6 Deleting Links

This method requires both a subject and its parent and deletes the line in the hierarchy table corresponding to (parent,subject) .

4.5.4.7 Renaming Subjects

In order to avoid introducing circularities, only new subject names or subject names with new sense numbers can be introduced. Should the user type a currently existing name, an exception is thrown. Modifications of the database and tree is quite easy, involving a search and replace through both.

The other classes implement a group of straightforward methods which delete, add or change various values in a table. For example the method which deletes a parent-child link first verifies if the child still exists. If not, it deletes the current subject.

4.5.5 Model-View-Controller Architecture

The graph and the tree form two views of the same hierarchy. The model/view/controller architecture, first popularized by Smalltalk, demonstrates a good model for building a system making different, synchronized presentations of one underlying data set.[32] For example a CAD system could have one underlying representation of an object with several views (close-ups, side view) and several user interfaces for manipulating the object.

The advantages of this architecture include:

- a separate design of program pieces
- binding between the model and view is made at run time.

The general architecture of the model sees several controllers manipulating one model and several views changing as the model's state changes. The model manages the data and conducts all transformations on it. It has no knowledge of the controllers or the views. The system itself maintains the links between the model and its views and notifies them when the model changes.

Within Java, two classes provide support for this architecture:

- "'Observer' : any object that wishes to be notified of changes in an observable object
- 'Observable': any object whose state may be of interest, and in which another object may register an interest."[33]

Within Dockman, four classes implement the model:

- System: dockman.system.manager.
- Controllers: treeView and graph
- Views: treeView and graph
- Model: database table 'hierarch'

The treeView and graph both act as controllers in the sense that they are the point of 'user entry' for modifying the model. We will briefly examine each class in turn, showing how they implement the architecture. It should be noted that only the subject hierarchy viewers, (the treeView and graph) currently implement this model. When

'multiple views' of the statement and facet grids are built, they should be integrated into this architecture.

4.5.5.1 dockman.system.manager

Once the treeView and graph have captured user input (mouse clicks) and sent them to the main frame class, it can call several methods in manager to add, delete and change subjects or links.

Each of these has a call to the setChanged() and notifyObservers (AddSubjectEvent, InsertSubjectEvent etc...) events of Observable. SetChanged() indicates the state of the model has changed. notifyObservers() then updates all the registered observers.

Each observer must implement the update() method. Whenever notifyObservers() is invoked, the observer interrogates the observable object to ascertain its state and adjusts its view correspondingly. The following steps describe the operation of the model:

1) the user manipulates a controller(graph or treeView). It makes a change to the model via a public method in manager

2) The method modifies the data, and calls setChanged and then notifyObservers().

3) The update method of each observer is called, indicating that a change of state has occurred.

4.5.6 Editable Tree

Editable tree extends symantec's 'TreeView class'. It provides one view (the graph provides another) of the hierarchy created by the user. A manager object is required when creating an instance of the treeView class. This object contains the Symantec Session and server objects required to establish database connections. The tree is built by first constructing an associative array from the edge table

Thus, the edges of the graph are placed into an associative array, where each subject name points to any children it may have. Following this, the constructTree method uses this data structure to build the tree recursively in a depth first manner. It should be noted that a subject can appear multiple times under different parents in the treeView, since a new 'tree node' must be created for each parent, child pair.

4.5.6.1 Ancestors

One of the key methods of the tree is 'ancestors'. It identifies all the nodes having the same name as a subject and returns all the parents of those nodes, as well as their 'level' or distance[†] from the subject. The method begins by finding the nodes with the same name through a simple search of the vector data structure which underlies the tree. Following this, the subject and its level (0) are placed in the hashtable. Then,

For each subject node

[†] Level : the current subject is at level zero, level is one for a parent (there can be several), two

```
Get the node
While the node's parent is not equal to null
        Place parent name and level in the hashtable
```

If the name is already found in the hashtable, the method tries to compare the level at which the subject previously existed with the current value of the level variable. This is necessary because of multiple inheritance. A previously explored branch of ancestors may have contained the same subject name. However, there may have been fewer subject names between the selected subject and this one. In fact, we always want to place the highest level found in the hash table. For example, given the two branches of the hierarchy below:

```
KbTop(5)
        set(4)
                sequence(3)
                        program(2)
                                class_method(1)
                                        accessor_method^2(0)
```

and the second tree branch:

```
kbTop(4)
        member(3)
                method(2)
                        class_method(1)
                                accessor_method^2(0)
```

We would always want kbTop to have the highest level (e.g. for it to appear at the end of the ancestry list). Therefore, we would want that to be reflected by kbTop having level 5. Also, to ensure that hiding works, it is critical that the lowest levels be 'selected'

---

for a grandparent etc...

first. Thus, member should 'block' or hide any equivalent predicates found in set (which is of level 4) when hiding is turned on.

Finally, the results of ancestors are sorted by these levels and returned to the calling method , which is typically modifyStatementGrid or modifyFacetGrid. There are several other methods of note in editable Tree. Among them is findNodes, which returns a vector of all nodes having a particular subject name and findChildren which returns the children of a given node.

Also there are a variety of methods for:

4.5.6.2 Inserting Sub Concepts

Given a parent and a subconcept, it loops through all the nodes containing the parent and creates a new child node for each of the parent's nodes.

4.5.6.3 Deleting All Occurrences of a Subject

The philosophy of the model view controller architecture requires that the view (in this case the editable Tree) seek information about its current state in the model (in this case the hierarchy table). As we have seen, when all subjects are deleted in the 'hierarch' class of the tables package, many of the database rows are updated in order to attach children to the deleted subject's parents. The edge table is then re-read and the tree adjusted to reflect the new database state.

4.5.6.4 Changing Subject Names

Finds all occurrences of the old subject names and replaces them with new text.

4.5.6.5 Inserting Links

Inserting a link requires attaching the complete subtree of the selected subject to all occurrences (i.e. nodes) of the parent subject found in the tree. The following pseudocode briefly describes the implementation:

Find all nodes containing the parent subject

Create an associative array with the parents as the key and a vector of children as the value

For each of the parent subjects:

Use the construct tree method (detailed previously) to build and attach the appropriate subtree.

4.5.6.6 Deleting a Link

It simply finds all the nodes having the currently selected subject as text. It then loops through them. If the parent of a node is equal to the parent from which we want to delete the link, then remove the node and its subtree.

5 Chapter 5 – Using the Knowledge Base Builder

5.1 Usage

DocKMan's knowledge base builder has been used extensively since November 1997. Both students (undergraduate and one PhD student) and members of the Language Analysis and Knowledge Engineering (LAKE) lab have developed knowledge bases using the tool. During this time, user feedback has greatly contributed to the refinement of the knowledge base builder. In this chapter, we will detail who used DocKMan and the experience and problems they encountered. Following this, we will examine in depth the construction of a Java knowledge base. Both the process of gathering and structuring the knowledge as well as the interaction with DocKMan will be examined.

The first extensive trial of DocKMan took place in November 1997 when a dozen students from Dr. Doug Skuce's Web and Information retrieval course used it to build a knowledge base about vehicles. Instruction consisted of a small demo during class, as well as a five page manual online. During this trial we observed that:

- the majority of students had little difficulty constructing their 'is-a' hierarchies. This may be due to the limited scope of the project. User experience in building larger knowledge bases (beyond 50 concepts) shows that the complexity of this task escalates rapidly. The ultimate example of this is CYC, where teams of researchers have been involved for years in the task of constructing a vast knowledge base.

- Multiple inheritance did seem to pose a larger problem than the simple 'is-a' relation. Some students (perhaps 20%) required some additional counseling and examples before making appropriate choices in multiple inheritance.

- Students experienced little difficulty creating appropriate statements. However several did not understand the concept of facets, including: how to structure the facet and which type of information should be included in a facet.

- Students found the user interface intuitive and easy to use. Everyone seemed able to enter both subjects, statements and facets with little difficulty.

In addition to the observations of student's behaviour, two significant problems were encountered with the system. The first of these was the inability of the server to disconnect applets, after a user had quit the browser. This resulted in an increasing number of users, so that the server would be registering six users while there were only two online. This caused the next user who logged on to get a small window prompting him for a user name and password. This window would continue to appear even after a user had clicked 'OK' several times and they were unable to continue working. We now know that the dbAnyWhere server sometimes refused connections to users. A series of tests was conducted to try to determine the reasons for these failures. We also contacted Symantec to ascertain the source of the problem. Among the observations made were:

- The number of users did not decline consistently when applets were closed. Symantec indicated this was a bug that may be fixed in March 1998.

- When virtual memory fell below 5 megs, the problem also appeared.

- Also, the SQL Anywhere dbms corrupted several of the databases making them unreadable , even at the dbms level. When this occurred, the message also appeared.

The second major problem was a conceptual error on my part. Users were allowed to enter new subjects throughout the tree. However, several entered existing subject names as children in the same branch as an ancestor. This 'circularity' error was resolved by the 'add parent link' feature.

5.2 Approaches to Knowledge Base Construction

There are two approaches to building knowledge bases.

1) start from a set of documents

2) start from scratch

The first case is easiest, since a resource already exists which can be searched for knowledge base elements such as concepts, statements and facets.

5.2.1 Building a Knowledge Base Using Text as a Resource

Some of the steps involved in constructing a knowledge base from a set of documents will be presented. Also, a tool called the Text Analyzer, currently being developed in the LAKE lab, which can assist in this process will be described.

5.2.1.1 Identify Concepts

Identifying the key concepts of a knowledge domain is one of the most difficult tasks for building large knowledge bases. The first resource for accomplishing this are subject experts. Through sessions both individually and within a group the key concepts, their definitions and the relations between them can be sketched. Documentation can also serve as an important adjunct. It can assist in starting the process, by providing the first definitions and concepts. However, several important problems exist with documentation, among them:

1) Incomplete, poorly written or maintained documentation

2) Lack of tools to search at a fine grained level

Companies faced with the first problem would probably not be concerned with building knowledge bases in any case. The lack of quality documentation has proven in past experiences to be an almost insuperable barrier. Fortunately, the current sponsors of the LAKE lab's research , NORTEL ,have implemented a company wide standard of controlled English called 'Nortel Standard English' (NSE). This will enable the use of 'knowledge extraction' tools to assist in the construction of knowledge bases. One of these tools is the Text Analyzer. It can assist in both the extraction of concepts and statements from documents. The Text Analyzer was developed and is currently being refined by Judy Kavanagh, a researcher in the University of Ottawa's LAKE lab. Further information about it can be found in Judy's thesis, which was a precursor to some of the work detailed below. It can be found at www.site.uottawa.ca/~kavanagh/Thesis/thesis/Abstract.html. The text analyzer forms the

first half (the knowledge base builder being the other portion) of the DocKMan system. The two tools are meant to ultimately work together, the text analyzer serving as a 'knowledge extraction tool' and the knowledge base builder as a 'knowledge organization' tool. Over the next five pages, the functionality of the Text Analyzer will be briefly explained. Then, an example of how a person could use both tools to construct a portion of a knowledge base will be explained.

5.2.1.2 Identify Relation (part-of, is-a, question)

Another essential step is to identify the relations between the concepts. Many bodies of knowledge could profit from several relations. For example a kb with the topic 'java' could have an 'is-a' hierarchy describing all the key concepts of the domain such as class or object. As well, it could have a question hierarchy, organizing and classifying common questions.

5.3 Text Analyzer

The text analyzer has several features which allow users to construct portions of a knowledge base from texts. It operates by first preprocessing text through a syntactic parser called ENGCG (English Constraint Grammar), a product of Lingsoft. [www.lingsoft.fi]. The text is split into sentences and ENGCG identifies the parts of speech within it.

The results are then placed in a database table. This enables analysis of the relations between words as well as the patterns within sentences. Thus, some of the

components of a knowledge base including concepts, predicates, statements and facets can be identified.

5.3.1 Concept Identification

Three text analyzer features assist in concept identification. They include:

1. Noun frequency: By identifying those nouns which occur frequently in the text, some of the key concepts can be identified. For example, in the group of Java texts:

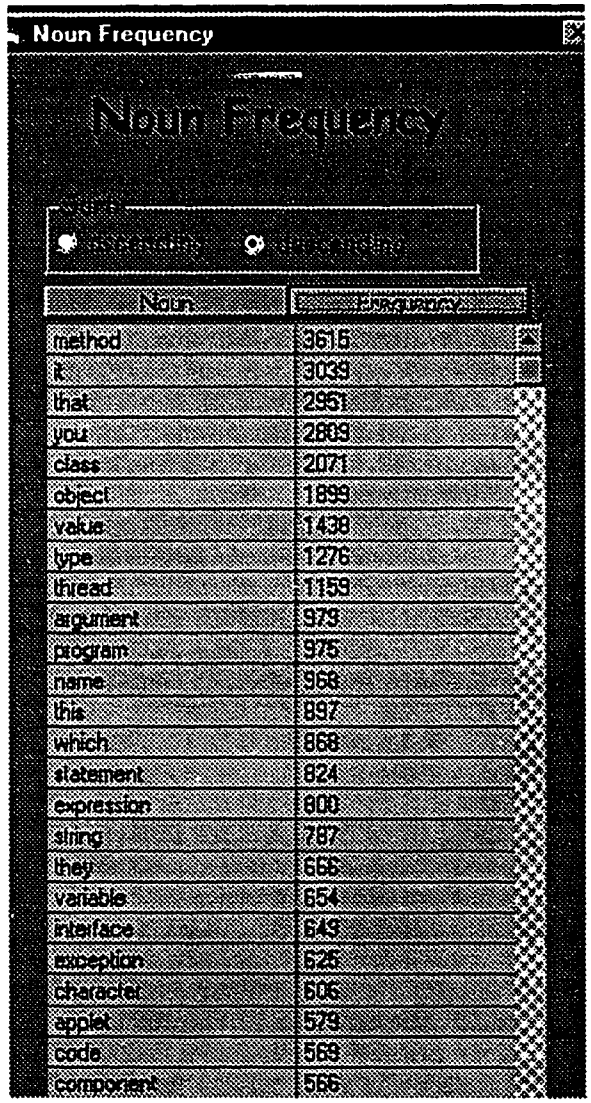


Figure 5.1

2. Compound nouns: These are concepts formed by several individual words such as 'native method' or 'layout manager'. Other elements of a concept hierarchy missed by the first tool can be found.

3. Concordance: The concordance tool searches for a word and splits sentences in relation to that word.

In a search for 'applet' the following results were obtained:

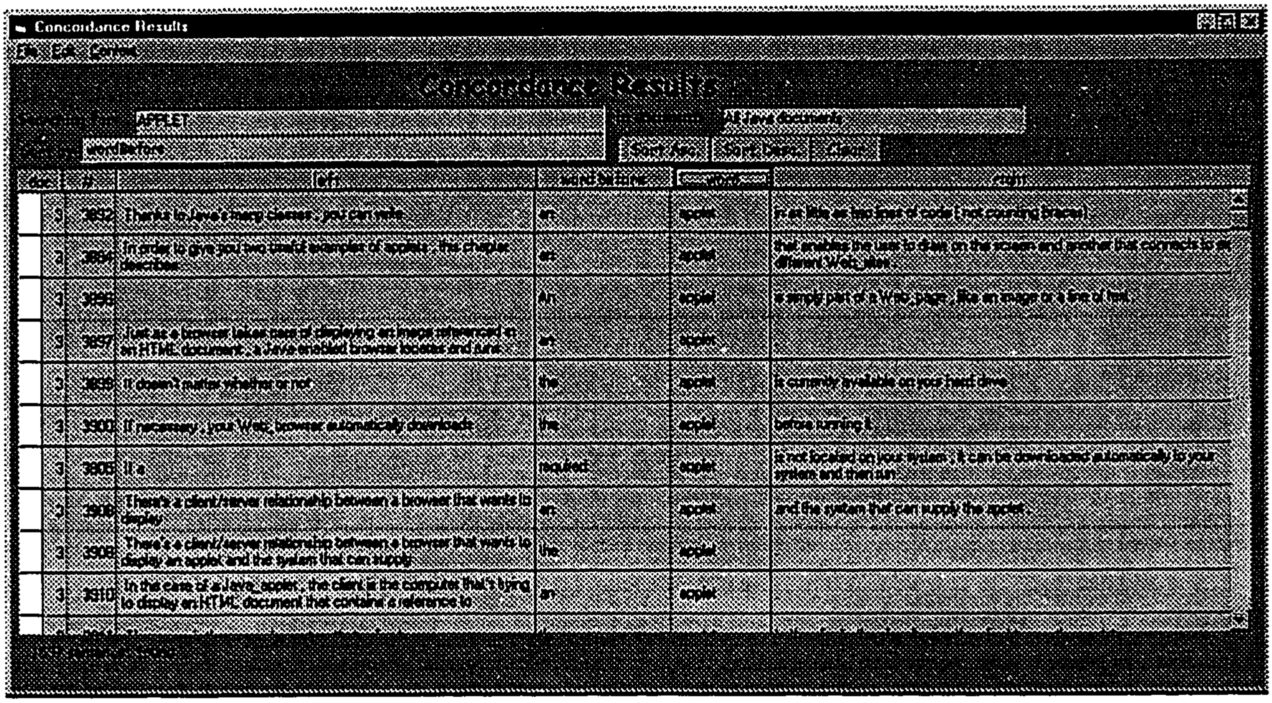


Figure 5.2

The user can then sort on 'word Before' to identify subconcepts , for example 'kinds of error' or 'file stream'. Also, adjectives which would indicate facets can be spotted. A sort on the right of the word could reveal patterns such as 'is a' or 'part of' indicating relationships within the hierarchy.

5.3.2 Predicate Identification

The primary tool at this time for predicate identification is 'verb frequency'. Combined with some of the statement identification tools it can be a guide to which predicates should appear in a knowledge base. For example if a frequently occurring predicate were not found in the kb after entering many statements, it would indicate several important ones have been missed.

5.3.3 Statement Identification

Two options, 'verbs following noun' and 'subject/verb/object' facilitate the discovery of statements. The first prompts the user for a word, and shows all occurrences of it with a related verb.

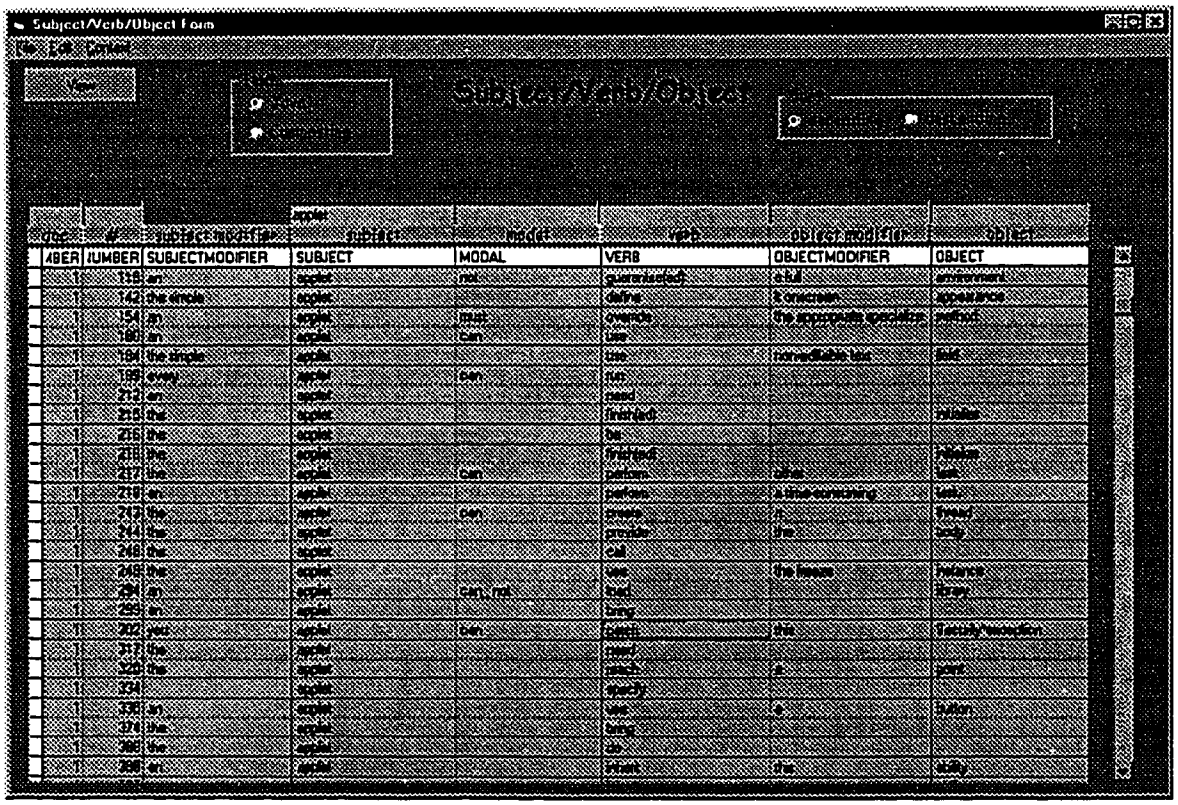


Figure 5.3

In this example we can sort on a verb and see many potential statements . Also, the frequency of occurrence of a phrase can be an indicator of statement importance.

5.3.4 Facet Identification

In addition to its ability to identify statements, the subject-verb-object (SVO) tool can assist in facet discovery. For example, the 'modal' field can indicate the 'modality' of a statement e.g. can, should or must this statement be true. Also, the object modifiers and subject modifiers can often become facets.

5.3.5 Question Answering Tool

The question answering tool offers the user a variety of search options, each of which is expressed in 'question form'. Figure 5.4 below, shows the seven question types currently available.

The user enters a noun in the blank space, except in the 'How do I' case, where a verb is entered followed by a noun. These questions are answered via searches for semantic patterns within sentences. These patterns have been discovered by linguists working with the LAKE group. For example, the first definition question includes patterns such as 'is-a', 'is an', within several words of the search term. Such questions can be of great assistance in knowledge base construction. The sentences culled from the question answering system can respectively find:

- definition statements for concepts
- super and sub concepts (questions 1 and 3)

- part-of hierarchies (question 4 and 5)

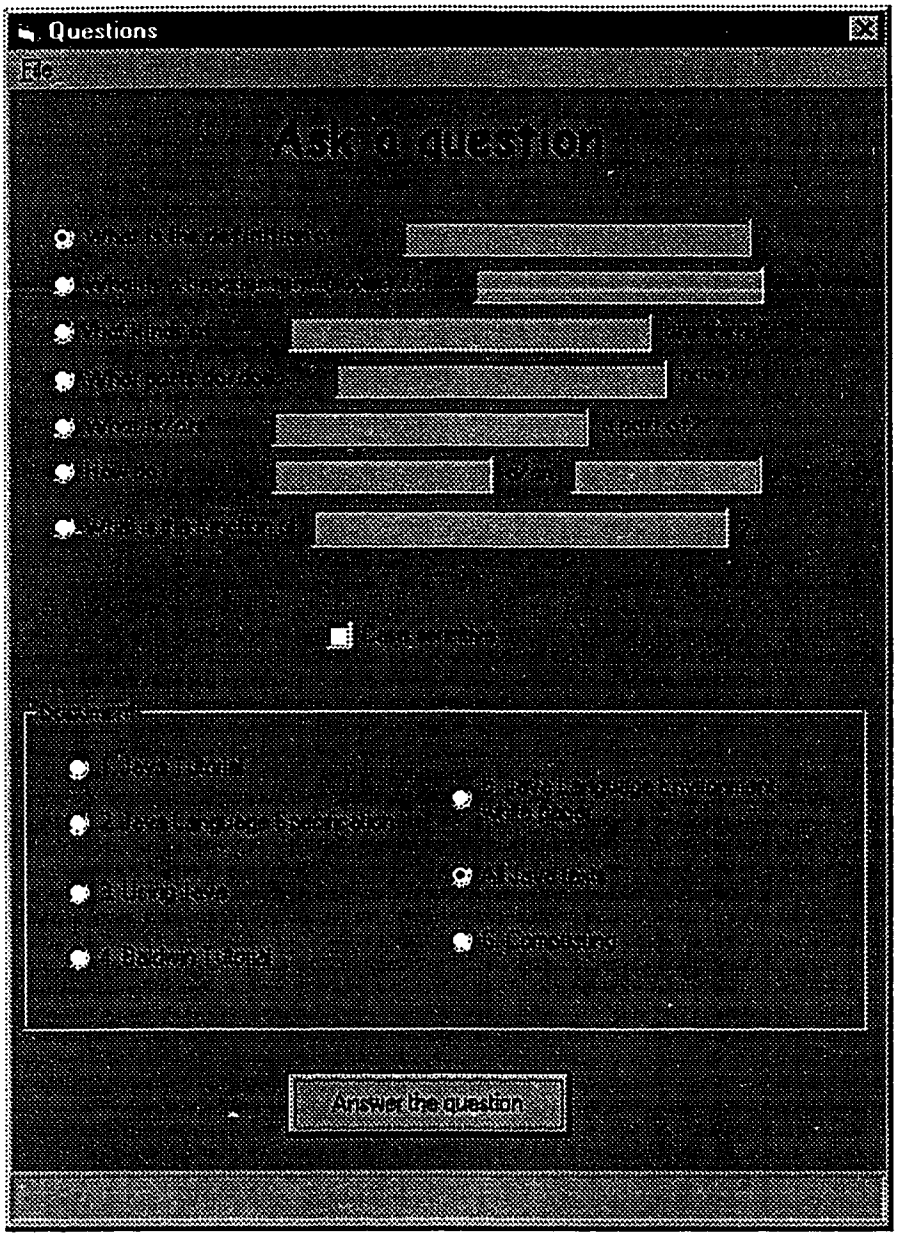


Figure 5.4

Having briefly examined the functionality of the Text Analyzer, its use will be demonstrated in a brief example.

5.4 Java Knowledge Base

In a trial of DocKMan, four megabytes of various documents about Java were downloaded from the Web. This included a complete Java text with a table of contents. It formed an initial subject hierarchy. Subjects were added and deleted as necessary to 'clean' it. Once this was completed, a linguist began using some of the text analyzer's functions to create the knowledge base.

In an interview with a linguist who used the system to identify concepts, subtypes, functions (or statements) and parts, we obtained some insight into how the tool might be used to build knowledge bases. Two of the text analysis tools surveyed seemed of greatest value, the question answering tool and the concordance. Definitions for various concepts were identified via both the 'What is' feature and the concordance. The second element of the question analysis tool also proved useful in 'explaining the concept' and could provide some statements.

In comparing text analyzer results with those of a human analysis of the text, the general impression was that the definitions and subconcepts were properly identified. However, other results were less positive, with many functions being missed. This may be a result of the current state of development of the tool, rather than its potential in assisting knowledge base construction.

5.5 Knowledge Base Construction Walkthrough

During the design of one of DocKMan's predecessors CODE4, an investigation was made into the activities involved in knowledge organization.[34] Among them were

Categorizing

This activity tries to determine the organization of category elements. An example is the choice of a hierarchy type ('is-a', 'question', 'part-of') and the subsequent decisions on ordering of the elements within the hierarchy.

Naming

Determining the correct name for a concept and being careful to distinguish between synonyms forms a second key activity.

5.5.1 Knowledge Visualization and Extraction

Finding specific facts quickly and 'in the context' of a particular concept is one of the knowledge base builder's advantages. It also allows the viewer to see the overall structure of knowledge at a glance. A demonstration of how a combination of the knowledge base builder and the Text Analyzer can assist in these activities will be presented. Also, a portion of the Java knowledge base will be explored to demonstrate its usefulness in knowledge discovery.

It was decided to try to extend the 'thread' concept in the Java knowledge base. It already existed under the grouper† 'operating system'. It is also a concept in Java. The first question that arose was where to place it within the Java hierarchy, e.g. What are its superconcepts? Using the question answering system's second question . The results for the question answering system's second question, (find ' supers ') where 'thread' is entered as the noun are found below:

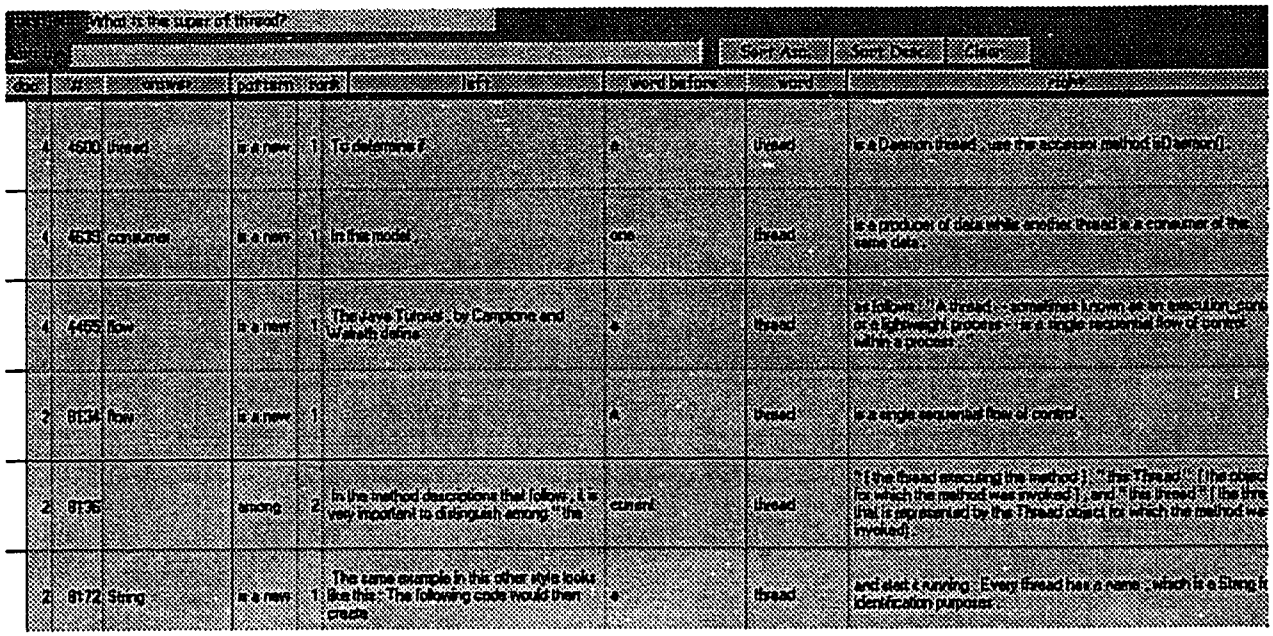


Figure 5.5

The responses vary from 'noise' such as sentence 4639 which talks about producers

† A grouper is a set of related topics (e.g., about-cars could contain car dealer, car).

and consumers of data, to sentence 4455 or 8134 which define thread as a 'lightweight process' and a 'single sequential flow of control', implying the superconcepts process or sequential flow of control.

The analysis of the sentences is of course assisted by the fact that the reviewer has knowledge of Java and has an idea of what 'process' means. However, process itself is not found in the knowledge base, so its superconcepts must first be found. A similar question analysis search is undertaken for process, but the results are meager, the only sentence returned being "Think about it this way, each program is assigned a particular person to carry out a group of tasks called a process". While this gives me a good example, it doesn't tell me what the super concept is. This is where a good computer dictionary in the body of texts could prove useful. The surrounding sentences are then examined to try to find the answer, to no avail. Finally, another text is examined and it's discovered that a "sequential process is a program in execution"[35] . Therefore a process is a program in an executing state. Knowing this one can search again in the hierarchy to find program. The new subjects 'executing program' and 'process' can then be introduced, followed by a link between it and the existing subject 'thread', using the methods seen in Chapter 3.

Having now introduced thread into the hierarchy, we are ready to add some statements. The verb following noun tool is invoked to begin searching for sentences with 'thread'. A portion of the output is found in Figure 5.6.

A substantial number of results are returned when thread is placed in the subject field. In order to find the most important statements, verb frequency is used as a guide.

Generally, those verbs with higher frequency will indicate more important predicates and statements. However, frequency can not be used as an absolute guide and a knowledgeable user's judgment is required. With this caveat in mind, two verbs were selected, 'create' and 'use'. One sentence will be shown from each, along with the resulting statements and facets.

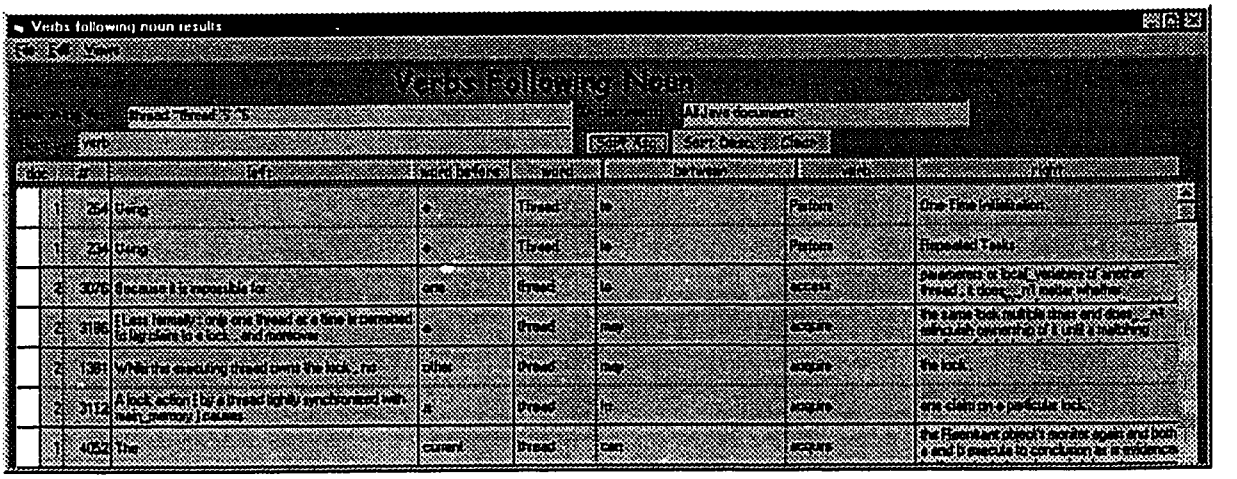


Figure 5.6

The first sentence deals with thread groups. It states: "Thread groups provide a way to manage threads and to impose security boundaries: For example, a *thread may always create* a new thread within its own thread group but creating a thread in another thread group requires the approval of the security manager as does the creation of a new thread group." This sentence illustrates some of the problems faced in knowledge base construction, even with the assistance of text analysis tools.

Several statements are evident from the text including:

- Thread groups manage threads
- Thread groups impose security boundaries

- Thread creates new thread groups. Facet = requires: security manager permission
- Thread creates new threads in thread group.
- Thread creates new thread outside thread group. Facet = requires: security manager permission.

This was a difficult sentence to break into statements and facet subdivisions. While statement identification proved easy, identifying the correct facets led to some debate. For example, the final two statements could have been one e.g. Thread creates new thread, with 'location facets' 'in thread group: does not require security manager permission' and 'outside thread group: does require security manager permission'. Also, a new subject appears, 'thread group'. The statements related to 'thread' should be fully inheriting since there may be subtypes of thread that share the same characteristics. This sentence also demonstrates the usefulness for knowledge base construction of controlled English documents , such as those written in Nortel Standard English(NSE).

The second sentence: "A thread may use the value of a variable or assign it a new value" is more straightforward. The statements are

- Thread uses value of a variable
- Thread assigns a new variable value

Both statements have the facet: Modality : sometimes. The facet is easy to spot in this case due to the division of the sentence into different parts of speech as seen below:

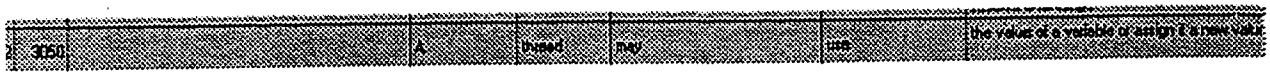


Figure 5.7

Having very briefly explored a method for building knowledge bases, we will now use Dockman as a knowledge exploration tool. The subtree below 'object' will be browsed. Several subject's statements and facets will be explored.

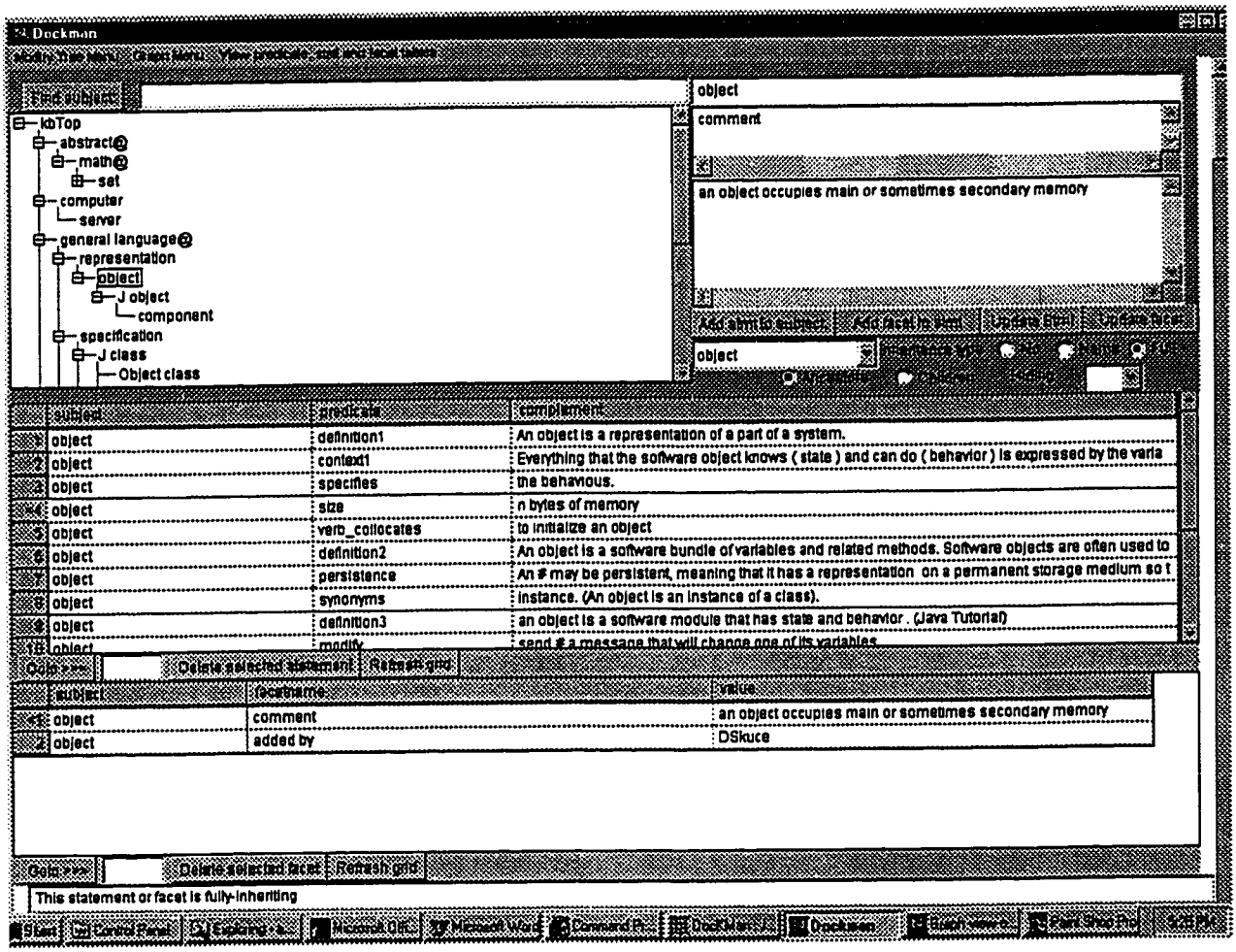


Figure 5.8

The @ symbol following a concept defines a 'grouper'. This means all children are related topics rather than 'is-a' relations. At a glance, the user can see some important relations between subjects. For example, divisions such as object is a representation are apparent.

The user can then browse both statements and facets while seeing the subject in the context of a hierarchy. For example, the user may want to know something about 'object'. Clicking on it shows several important facts about object, summarized from denser texts or from an expert's knowledge.

In this Chapter, users' experiences and impressions of DocKMan's knowledge base builder were detailed. Also, a tool, the Text Analyzer was briefly explained and its value as an aid in knowledge base construction explored. Finally, we walked through a small example of knowledge base construction and exploration.

6 Chapter 6 – Summary, Discussion and Future Work

This chapter summarizes the contributions of the thesis and presents ideas for future work on the knowledge base builder.

6.1 Summary

The main goal of the thesis was the development of a tool which would allow the easy structuring of information into knowledge bases. Chapter 1 began with an examination of some of the problems in information retrieval and knowledge management. DocKMan's knowledge base manager was proposed as a solution to some of these.

In Chapter 2, research related to this work was surveyed, including 'classic artificial intelligence' knowledge base systems such as CYC and Ontolingua, to more recent and less formal 'knowledge management systems' such as Code4, Know-it-all and DocKMan.

Chapter 3 introduced the knowledge base builder and the basic concepts behind it, its user interface and operations. Chapter 4 discussed the implementation of the knowledge base builder, the technology chosen for accessing the database from a browser environment as well as a guide to the classes that manage the knowledge base builder. In Chapter 5, we presented the experiences of some of DocKMan's users. As well, a tool which can operate in conjunction with the knowledge base builder, the text analyzer was presented. Finally, an example of the construction of a portion of a knowledge base using

both tools in tandem was presented. In this chapter, Chapter 6, we also present some future work to be done on the knowledge base builder.

6.2 Discussion

After designing and implementing the knowledge base builder, we have come to the following conclusions on both conceptual and implementation issues:

Conceptual

- The knowledge base builder allows users to organize knowledge in 'frame-based' structures.
- The combination of well written texts in controlled English (e.g. Nortel Standard English) combined with tools such as the text analyzer form an avenue of research into better means of extracting knowledge than exist at present
- The output from this tool can be evaluated by a user and easily placed in a knowledge base with the builder tool.
- The hierarchical organization of information seems to speed users ability to both retrieve information and better understand its context.

Implementation

- With over 25 people having used the system, few difficulties were experienced with the user interface. Users were able to build knowledge bases with the assistance of a small manual and a short demo.

- While users found the interface intuitive, several problems with interactions between the various layers of software (middleware and DBMS) made DocKMan prone to breakdown.

6.3 Future Work

1. Support for different inheritance types

   Currently we have No, Predicate-only and Full inheritance. These inheritance types are primarily for the 'is-a' hierarchy. Other hierarchies (e.g. 'part-of') may require other inheritance types.

2. Support for dimensions and groupers.

   A dimension of an object is a division based on a particular characteristic. For example, the concept car could be divided by colour (green car, blue car, yellow car) or by manufacturer (Ford, Chrysler, GM).

   A grouper forms the top of a topic hierarchy e.g. a set of related topics. For example: (car related topic -> car dealer, car) where car related topic would be the grouper for this hierarchy. A valid statement for it might be 'concerns | cars'.

3. Facet-picking menu

   Many facet names, such as source and modality , tend to repeat themselves. This feature would simply present to the user a list of currently used facet

names. This would have the advantage of indicating to new users some of the facets they should consider adding to statements.

4. Embed hyperlinks in the grid

   In order to achieve full Document based Knowledge Management the 'knowledge base builder' must be more closely linked with documents. For example individual statements or facets could contain hyperlinks in the complement field to an area of a document which explores the topic in further detail. This feature would give DocKMan a greater 'document browsing' capability.

5. View of subjects or statements depending on user permission

   This form of 'fine grained control' is important in cooperative knowledge base construction where the viewing and/or editing of certain facts or subjects is restricted to one group of users.

6. Search on text and knowledge base with integrated result presentation.

   Forms to search both text and the knowledge base, with results presented together to the user.

7. Integration of the text analyzer with the knowledge base builder

   As a first step, the text analyzer should be available on the Web along with the knowledge base builder. As a further step, the text analyzer could create a skeleton knowledge base automatically. Also, it could be used to analyze any

text retrieved from the integrated search facility described in the previous point.

Systems such as DocKMan's knowledge base builder rely heavily on informed user's behaviour (to both navigate through information and place it in the right area of the hierarchy). Therefore, it should be tested extensively by users, with designer present during some of the tests. Future refinements of this initial prototype would be based on their input.

Further Information

The DocKMan users' manual written by the author (and converted to HTML format thanks to John Talbot) as well as the graphviewer manual written by Wratko Hlavina and additional papers and information about DocKMan can be obtained at http:// dockman.site.uottawa.ca. This site is maintained thanks to the hard work of John Talbot who can be reached at jtalbot@site.uottawa.ca.

References

[1] www.gvu.gatech.edu/user-surveys/survey-1998-04

[2] www.gvu.gatech.edu/user_surveys/survey-1998-04/use/q56.htm

[3] www.gvu.gatech.edu/user_surveys/survey-1998-04/use/q6.htm

[4] www.cyc.com/overview.html

[5] www.cyc.com/tech.html The CYCORP CYC Technology: Company Overview @ 1998

[6] www.cyc.com/tech.html

[7] Cyc Ontology Guide: Introduction. Cycorp. 1996. www.cyc.com/cyc-2-1/intro-public.html

[8] Eodem Loco.

[9] Douglas Lenat, CYC: A Large Scale Investment in Knowledge Infrastructure. Communications of the ACM, November 1995, Vol. 38, No.11 pp. 34.

[10] D. Skuce, T. Lethbridge, CODE 4: A unified system for managing conceptual knowledge. International Journal of Human-Computer Studies, 1995 , 42, pp.413.

[11] Ibid. p.417.

[12] Eodem Loco

[13] Ibid p.421.

[14] Eodem Loco

[15] Ibid p.422.

[16] Ibid.p.423.

[17] D. Skuce, Knowledge Management in Software design: a tool and a trial. Software Engineering Journal, September 1995, pp. 185.

[18] Ibid. p. 190.

[19] What is an ontology. Stanford University. 1997. www-ksl-svc.stanford.edu:5915/doc/frame-editor/what-is-an-ontology.html

[20] How to design an ontology. Stanford University. 1997. www-ksl-svc.stanford.edu:5915/doc/guided-tour/how-to-design-an-ontology.html

[21] What is an axiom. Stanford University. 1997. www-ksl-svc.stanford.edu:5915/doc/frame-editor/what-is-an-axiom.html

[22] T Lethbridge, D. Skuce. 1995. CODE 4: A unified system for managing conceptual knowledge. www.csi.uottawa.ca/~tcl/papers/ijhcs95/cod4jau3.html#RTFToC4

[23] www.grasp.com/turning.htm Grasp Information Corp.

[24] www.gvu.gatech.edu/user_surveys/survey-1998-04/graphs/use/q75.htm

[25] A. Glassel, Was Rangamushan a Yahoo? Whois.internic.net/nic-support/nicnews/mar98/enduser.html

[26] R. B. Allen, Two Digital Library Interfaces That Exploit Hierarchical Structure. superbook.bellcore.com/PAPERS/RBA/LIBR/libr.html, June 2 1995, p. 2.

[27] D. Nation, C. Plaisant, G. Marchionini, A. Komlodi. Visualizing websites using a hierarchical table of contents browser: WebTOC. www.uswest.com/web-conferences/proceedings/nation.html p. 2.

[28] P.H. Winston, Artificial Intelligence Third Edition. Addison-Wesley, 1992 . pp. 179-183

[29] Microsoft Advanced Data Connector 1.0 FAQ. www.microsoft.com/adc/faq.html

[30] J. Sutherland. OOPSLA '96 Integrating Java, Objects, Databases and the Web. www.tiac.net/users/jsuth/oopsla96/suth.html

[31] Project Java Activator. Early access release 3. 1998. Sun Microsystems. Java.sun.com/products/activator.

[32] W. Lalonde, J. Pugh. Inside Smalltalk Pretnice-Hall, Englewood Cliffs NJ. 1991. pp.57-83.

[33] T. Sundstedt, Observer and Observable: An introduction to the Observer interface and Observable class using the Model/View/Controller architecture as a guide. www.javaworld.com/javaworld/jw-10-1996/jw-10-howto.html.

[34] T. Lethbridge, Practical Techniques for Organizing and Measuring Knowledge. www.site.uottawa.ca/~tel/thesis/html/thesis.html#RTFT0C15

[35] A. Schiberschutz, J. Peterson, P. Galvon, Operating System Concepts. Addison-Wesley. 1991. p. 90.