Hybrid KM: Integrating Documents, Knowledge Bases, Databases, and the Web

Doug Skuce
Department of Computer Science
University of Ottawa
Ottawa, Canada
doug@csi.uottawa.ca

Abstract

Knowledge is a critical resource but we still do not have many new ideas on how to manage it. Most (online) knowledge is currently kept in conventional documents that are hard to structure, classify, browse, search, and even find. Organizations struggle with masses of such documents in hundreds of formats. Classical AI has largely ignored this real and serious problem, and while information retrieval research has tackled some of the problems, it is totally at odds with how AI tries to deal with knowledge problems. Cooperative work systems such as the Web and Lotus Notes are beginning to tackle that aspect. Database systems can contribute much of the required functionality. Hence we seek to integrate functionality and ideas from these sources.

This paper describes an attack on the knowledge management problem by introducing a new hybrid notion of document cum knowledge base. We also discuss IKARUS, our experimental testbed.

Introduction

In our view, knowledge management (KM) is primarily concerned with managing the kind of information we now put in documents. We seek new ways to perform KM more effectively, i.e. we seek new ways to improve or replace the use of documents in organizations, scientific communities, teaching, or even for individuals. With web-based publishing and the use of search engines, many people are already experiencing significant changes in how they use and think of documents, i.e. how they do KM.

Today organizations are turning to tools like the Web, search engines, and Lotus Notes to help them with their KM problems. But the problems are far from solved. Some of the main ones are:

1. most knowledge is still recorded as unstructured natural language, with all its shortcomings of precision and conciseness
2. documents and in particular, small parts of them such as sentences, are often hard or impossible to find, hard to update cooperatively, and hard to keep coordinated
3. merging of information from various sources is difficult; "replication", as used in Notes, is a very crude approach, usually too coarse-grained
4. lacking any AI components, most systems today offer nothing in the way of inference, semantic checking, or natural language processing
5. documents are too knowledge-poor, i.e. the density of useful knowledge per megabyte (or per minute of reading) is too low, and finding and organizing them too difficult

The emergence of the World Wide Web has caused many people to rethink how they handle their documents, which for most people = knowledge management. But no one has yet (to my knowledge) proposed a suitable new format to significantly improve the KM process. In this paper, I shall propose such a hybrid "knowledge-rich" format.

Information retrieval has recently become of interest in part due to the availability of so-called search engines on the Web. But now most of us appreciate the serious limitations of their current functionality: tons of documents delivered but tons of reading still to do, and no assistance in organizing the mass of information.

Back on the AI side, there has been the notion of a knowledge base (kb) for years in many
varieties. Kbs were intended to serve a wide variety of purposes, but in fact have not yet seen widespread acceptance outside of AI research projects. This may be in part due to the lack of document-like behaviour, and in part due to the difficulty of constructing them (with which we have years of experience: (Skuce 1993; Skuce and Lethbridge 1995)

The traditional document is meant to be accessed mainly linearly, with modest assistance for non-linear access (e.g. indices). To find a specific fact, tables of contents and indices, or even full-text search, leave much to be desired. But two major additions have now made documents far more useful: hypertext and the advent of search engines. Without them the Web would be only a fraction as useful as it now is.

On the AI side, a conventional knowledge base is usually accessed by a specific subject. These are arranged in inheritance ("isa") hierarchies. Information is usually very sparse and condensed; much of the richness of expression of natural language (nl) having been sacrificed. Systems rarely provide any search capability, offering instead kinds of inferencing. AI can offer this and nl sophistication, e.g. syntactic or semantic tagging, and lexical knowledge, e.g. use of resources such as Wordnet to hybrid format.

My approach to these problems then is to conclude that neither documents nor kbs nor databases have the potential to be extended alone and therefore I seek a new kind of format that combines features of all. In this paper I shall describe my ideas on merging these technologies, and an experimental knowledge management system called IKARUS (Intelligent Knowledge Acquisition and Retrieval Universal System), which is the current version of a sequence of KM systems going back ten years. Note that such a system, having strong searching and displaying abilities, ideally integrates the functionality of a Web browser, a word processor, a knowledge base, a database system storing text fragments, and a search engine.

**A Hybrid Document-Kb Format**

I have not yet found a catchy name for this notion; how about dkb (document-knowledge base) for now? In the following, by kb I mean a conventional frame-like knowledge structure with some inferencing ability, and by dkb I mean the new hybrid format.

By document, I mean a conventional natural language document, possibly on the Web.

The problem is to integrate the most useful functionality of the three contributing technologies, and, incidentally, make the whole thing Web-accessible (so there are really four technologies). We should distinguish the storage format from the viewing format: at the moment I am most concerned with how the user will view a dkb. A dkb should be viewable in various display formats or modes and it should be possible to enter information in various interfaces, just as tables and figures are mixed with text in documents. For some purposes, a dkb might be viewed mostly like a conventional Web document, while others would be viewed more like a kb or a database, or anything in between. It should be possible to make a dkb appear like a document if desired (i.e. with sections, paragraphs, sentences).

In a document, the unit of knowledge is clearly the sentence, or at least, the clause. In a kb, it is what most frame systems call a slot, and in a database it is the record. We call our basic unit statements; they correspond to sentences in a document or slots in a kb or records in a database.

A dkb is a collection of statements of various types, like a database is a collection of tables of records. Each statement is about a subject. The inter-statement format is not fixed as it is in a document but is created at the user's request from a variety of possibilities. How statements are stored is secondary, but for the moment, we are experimenting with using a conventional DBMS, since a significant number of the operations we desire are already supported by a DBMS. The statements are (in our present storage model) records whose fields are parts or attributes of statements: e.g. subject, verb, direct object, date of entry, certainty factor, etc; we call these parts facets. We seek a conceptual and storage format that will support a variety of viewing modes: any desirable mode should be possible. By viewing mode, we mean a choice by the reader as to how the information is to be displayed, e.g. to look like a conventional paragraph, or a frame structure, or a database table, to name three possibilities.
Three structuring principles of a dkb are inherited from its three progenitors:

1. (from documents) statements must have a linear relation and belong to nested parts corresponding to paragraphs, sections, etc. Statements can be formatted to look like a document, and word processing-like operations are available (e.g. insert sentence).

2. (from kbs) statements are organized in various inheritance hierarchies and one usually accesses a frame at a time, corresponding to a subject of interest. Statements should have parts analogous to facets of a slot. Inferencing such as inheritance can be provided on demand.

3. (from databases) tabular structures must be supported, both to present statements in tabular views and to present other non-statement tabular material. The common database operations should be supported.

At the moment (January 1997) IKARUS does not yet support all three modes, only mainly first two. This is because we have only recently realized the importance of the database format, and are currently converting IKARUS to run on top of a DBMS (see below). Ideally of course, rather than graft it on top of a general-purpose DBMS, a specialized permanent store should be designed, probably using a Java-based persistent store.

The Role of Hierarchies

The only hierarchy in a conventional document is its part-whole structure, reflected in the table of contents. We call such a hierarchy of subjects a topic hierarchy (subjects, the main index for statements, can be in other kinds of hierarchy, hence the two terms; topics are subjects in a topic hierarchy). An IKARUS dkb, which we originally thought of more in terms of traditional (kb-style) "isa" hierarchies, now offers three kinds of hierarchies for its subjects. The two most common are the traditional "isa" or "type" hierarchy and the "topic" hierarchy. In a topic hierarchy, a sub-element need not (and usually doesn't) imply its super, as in "engine is grouped under car repair", but in an isa hierarchy, it must. Topic hierarchies correspond to tables of contents, subjects to an index. To support sequential viewing, the subjects in a hierarchy may have an ordering imposed as they do in a document, having the meaning: "you should read about this before this", a principle that extends to the statement level. The user may then choose to view a dkb more like a document, and approach it by order of topics, or more like a kb, and approach it by the "isa" hierarchy, assuming the dkb had an explicit type hierarchy as well as the topic hierarchy. (We consider that at least one of the two is essential.) If desired, statements can be sequenced, so they may be presented in a text-like order and read thus ("show me a paragraph about birds"). The third kind of common hierarchy is "part of", which can be used when you want to describe a recursive containment structure, e.g. software modules containing others, or parts of an airplane.

Due to their AI origins, IKARUS statements also have kb-like i.e. frame-like behaviour and attributes: e.g. a subject may inherit statements or parts of statements (called facets) from some higher subject. In addition, facets can contain pointers to other elements of a dkb or other dkbs, e.g. from a statement to other statements, permitting structures such as conditionals.

Viewing Structures

A critical aspect of a dkb is its wide variety of viewing formats, somewhat like a conventional database offers many ways to view data, or a word processor offers several views. We have evolved the following approach to viewing choices. First, one chooses whether to view statements by subject, like a kb frame, or by section, like in a conventional document, where sections are nested from the whole document, chapters, etc, down to the sentence level. Next, one chooses how to order them. Third, on top of these primary viewing options, we can add "masking" i.e. hiding or showing only those that meet some condition, date of entry.

In addition, one may want statements which contain only pointers to other statements or material such as conventional documents, or sentences or paragraphs extracted from such, or conventional web anchors. But our goal is to gradually move away from any reliance on conventional documents, i.e. if starting a new dkb from scratch, one should put as much knowledge as possible directly into structured dkb statements (see below) rather than write it more in the form of conventional sentences. But people find this hard to do, and most knowledge for the foreseeable future will still be in document
format. A middle position would be to have an expert prepare a set of key subjects as a dbk using structured statements, and have others contribute ordinary sentences or paragraphs.

Here are some example display requests:

1. I want to see statements entered in 1996 about attribute of vehicles as a database-like table.

2. I want to see all statements about printing in Unix in which the destination facet specifies a network device (see below for the idea of a facet.)

3. I want to see all statements about planets whose verb means "motion".

4. I want to read sequentially all statements about Java with the facet level: beginner. Format as a Web document with a hierarchical (book-like) structure.

5. I want to see all statements containing the string "actor" sorted by their verbs.

The hardest problems are those involving word meanings, i.e. that would require some kind of lexicon (or existing kb) to deal with requests such as 2, 3 or 5.

**IKARUS Knowledge Structures**

In IKARUS, a statement is thought of (and at the moment represented by) a database record (we are currently experimenting with MSQL 2 and Access). A dbk then is currently being implemented as a database, a set of tables, and though we realise this is not an optimal storage solution; it permits us to experiment for now. A typical statement structure would have the following facets (columns in a table of statements), which we group into three types:

- **linguistic facets** (for statements that mimic natural language sentences)
  - subject: all statements have a subject, which is what they are about
  - verb
  - direct object

The above format serves to store simple statements having just a subject-verb-direct object, which is what most kbs do. For more natural statements, we may add facets containing quantifiers, modifiers, verb complements, tense, etc.

**(annotation facets**

<table>
<thead>
<tr>
<th>date of entry</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>structural facets</td>
<td></td>
</tr>
<tr>
<td>statement no</td>
<td>a unique index</td>
</tr>
<tr>
<td>next statement pointer (for viewing sequentially)</td>
<td></td>
</tr>
<tr>
<td>part of points to section this sentence is part of</td>
<td></td>
</tr>
</tbody>
</table>

A facet having no syntax or semantics rules we term informal. If it could be at least checked automatically for syntax, or lexical content, or a human can understand it with no possible ambiguity, we call it semi-formal. One that could be automatically translated into something like KIF and axiomatized (for example), we term formal.

A dbk is designed for a particular purpose, analogous to a database design. It is constructed of a number of tables, like a database, both to provide various formats for statements and others for the hierarchies, lexical information, and the document structuring information. We do not discuss these.

**Knowledge base Linking and Merging**

One of our current research interests is on how to link a dbk to others. The idea would be to permit a network (a "subWeb") of dbks to refer to each other so that a user can access as many of them as he/she needs to extend what is available locally, just as the Web does for documents. To do this IKARUS permits storing what we are calling "remote subject pointers" as facet contents, so that you see what look like anchors pointing to subjects that are stored in remote dbks when you access a subject in your local kb. By clicking on one, you cause another window open showing the remote subject entry, like jumping to a new Web page. But what is harder (we are currently working on this) is to intelligently merge the two into one coherent display. The problems arise when the remote dbk does not have the same facet structure, or differs in terminology.

**Linguistic Issues; Lexicons**

If people are to share knowledge in the manner we envision, they must agree on how words are
to be used. Indeed, as I have argued for many years (Skuce 1995a,b), the sine qua non of knowledge sharing is shared ontologies, which are basically standardized sets of terms. Some of the problems are the same for conventional databases: metadata must correspond properly. In our dkbs, we identify four levels of terms on which agreement must be made for linking to make sense:

1. The most important names are for the subjects: if my "automobile" is to be linked to your "car" at least I must believe that your car is a true synonym, and you must believe it too if you are linking back to my dkb. Each subject must point to the same definition: e.g. you must make your car definition the same as mine! Dkbs can be linked with only this level of agreement, but the other levels are desirable too. Every subject should have some kind of definition, even if informal; the more formal the better. IKARUS supports sense numbers on subject names, permitting using a term with multiple senses, which we have found to be a necessity.

2. We usually use a three part semi-formal statement structure as follows: subject - statement-identifier (like a slot name) statement-contents (the slot values, that may have facets) The statement identifiers are often nouns or verbs.

3. The next level of terminology that would need agreement are the facet names and semantics. As a small example, a subject quantifier facet could be specified with values "all", "most", "some", "few" and "none". (This illustrates what we call semi-formality.) Here we might hope for agreement with less difficulty (than on thousands of subjects) on perhaps a set of 20 to 50 facets, akin to conceptual relations in conceptual graphs, i.e. that are widely accepted and that have reasonably clear semantics.

4. Finally we have all the remaining terms. In a large dkb, these can easily outnumber the others. We do not consider these at the moment, but can suggest that they should at least have definitions in a shared lexicon.

A dkb should be thought of as a super lexicon: it identifies a set of terms, i.e. their use, meaning, mappings to other terms or dkbs, etc. The problems of sharing dkbs have in common with all knowledge sharing efforts the ontology problem, which we view as essentially the problem of establishing standardized lexicons. Indeed, a dkb is an ideal medium for creating shared ontologies.

Building Dkbs: Two Approaches

Next we discuss how one might build a dkb. There are two approaches: a) starting from a set of existing documents, probably the most common case; b) starting from scratch, i.e. no documents available, or very few. We have experimented with both. The former case is much easier of course, because the dkb will act as an addition to an already useful resource, and one does not have to worry so much about completeness: one can pull salient facts out of the document and create equivalent dkb statements that are clearer, shorter, and easier to find. Or you can add totally new statements, with whatever level of formality is appropriate.

Document-based Dkb Construction

To work from existing documents, you first need a collection of them and some tools to access them. We have developed several such tools. As an example, we have downloaded about 4 Mb of documents of various types and sizes about Java from the Web using the WIC tool developed in our labs by Zayour (Zayour 1997) including a complete text on Java (java.sun.com/books/Series/Tutorial/index.html) with a comprehensive table of contents as a starting point. This text is over 800 pages as a conventional book. WIC automatically downloads the documents and indexes them using Glimpse (Manber 1994), a very flexible local search engine. WIC permits queries based on paragraphs, i.e. a query for a set of terms returns a ranked stream of paragraphs containing them. This facility alone we find very helpful in finding material in a document base. Next, we import automatically the table of contents from the text, adding or deleting any subjects we wish, creating a topic hierarchy in IKARUS. This provides a navigational "backbone". From these, we can extract all sentences having a subject occurring in it, and create a preliminary dkb indexed by these subjects and selected other components in the statement, such as the verb. We use a syntactic tagger for this.

By clicking on a subject, we have the following options for access the dkb so far created directly from documents:
1. view a section of the document (e.g. where the subject is introduced)

2. view many paragraphs about that subject, usually adding some more terms to restrict them. They are sequenced by relevance.

3. view all sentences in a concordance-like display (we currently use a separate tool developed by Kavanagh (Kavanagh 1995) for this but it will soon be subsumed by the DBMS functionality.)

To create real dkb statements, manual work is needed. This is justified if enough subsequent users will access the material, and it is of sufficient quality. We also envision a mode where dkb statements are created incrementally by actual users who find material in the text (or create it themselves) thus slowly building up a repository somewhat like a faq list.

**Direct Dkb Construction**

As an example of creating a dkb directly, suppose we are designing a piece of software. Using a dkb system like IKARUS, we might start with the following hierarchies:

1. a topic hierarchy, for all subjects (major terms)
2. a class hierarchy
3. an isa hierarchy for all subjects

Note that case or IDE tools would probably have a class hierarchy, with a fixed data format in which to record information. They would also have a separate topic hierarchy as a help system (the two are often poorly coordinated.) IKARUS can reproduce the function of these structures, plus many more. Next, we design specific statement structures for various purposes. Figure 1 shows an example of a dkb entry for the subject “array” (in Java). Facts such as these are very difficult to mine in conventional sources.

IKARUS, being Web-based, permits a group of designers to record and share every small fact related to the design, from the meaning of terms (an important kind of knowledge too often left to chance) to detailed specifications or user documentation. The more formal the structures are, the more we can write auxiliary software that can scan the dkb entries looking for possible errors. Not only will this use of a dkb improve the design process by improving the communication amongst designers and programmers, it will greatly ease the production of documentation which is too often left as an afterthought.

**Concluding Remarks**

The ideas and work presented here are very preliminary, but they represent conclusions derived from many years of work with the CODE system, and more than a year using its successor, IKARUS, but without a DBMS as persistent store (we have used dbm in IKARUS until Jan 1997). We have also worked for about five years on techniques for extracting information from texts. We have tried therefore to summarize our current ideas about how one could better store and present such information. Our techniques may work best on technical information; we have no experience with nontechnical material.

The best way to demonstrate the utility of a true dkb would be to create one that many people found indispensable. While we would like to do this, we lack the human resources. We plan however to assemble a partial dkb for Java over the summer of 1997. It should contain one or two thousand statements on about one hundred subjects.

The most obvious use for a dkb, from the work we have done, would be as a reference or teaching resource, i.e. largely replacing manuals, texts, and online help. But another more challenging use would be as a medium for knowledge dissemination, i.e. scholarly publishing. Instead of writing conventional articles, particularly in experimental sciences, one would add to a distributed dkb. One could instantly attach one’s new ideas in appropriate places in a global dkb. It would be enormously easier to find information, and to coordinate it (much inadvertent duplication would be avoided.) It would represent a step even greater than the move to Web-based publishing that is now happening. This an avenue I hope to pursue.

**Acknowledgements**

This research has been supported by Mitel Corporation and the Natural Science and Engineering Research Council of Canada.

**References**

Kavanagh, J. 1995 The Text Analyser: A Tool for Extracting Knowledge from Text. MSc Thesis, Dept. of Computer Science,
array

General facts

is a: object ; indicates inheritance in IKARUS, i.e. you may choose to see statements from object
subclass of: Object ; the superclass in Java
definition: An array is a special kind of object that can contain a number of elements of the same type indexed by integers. (written by DS)
comment: Arrays are unusual. There is no visible class Array, and hence they cannot be subclassed. They do have the full behavior of an Object.

Is-verb facts ; a common type of statement
an array is: referred to by reference; automatically garbage collected, etc inh from: object

How To (keywords bolded)
create=allocate: <element type> <var> example : byte ArrayOfBytes
create=declare: new <element type> [<integer>] where: integer is the length example: new byte [10]
destroy: automatic upon becoming unreferenceable
subclass: not possible
copy: use clone() ; etc for every other verb that applies to arrays

Accessible Properties ; properties that are explicitly stored
length=size: use: length

Other Attributes ; properties that are not explicitly stored
element type: any kind of Object comment: all elements must be of same type. The element type is not an accessible property.

Methods ; a list of method descriptions for objects in general inherited from object could appear here

Figure 1. The following is a slightly enhanced view of a typical IKARUS display. The viewing format is a cross between a document and a kb style. Anyone who doubts the usefulness of this display is invited to try and locate all this information any other way. (It took the author several hours.). The information is not complete. Semicolons precede comments.