A Knowledge-Based Electronic Information and Documentation System

Robert L. Young, Ph.D.
SciComp Inc.
5806 Mesa Drive
Austin, TX 78731-3742
1-512-451-1050, ext. 204
ryoung@scicomp.com

Elaine Kant, Ph.D.
SciComp Inc.
5806 Mesa Drive
Austin, TX 78731-3742
1-512-451-1050, ext. 201
kant@scicomp.com

Larry A. Akers, Ph.D.
SciComp Inc.
5806 Mesa Drive
Austin, TX 78731-3742
1-512-451-1050, ext. 205
akers@scicomp.com

ABSTRACT

We describe the capabilities of a knowledge-based system to automatically generate a collection of electronic notebooks containing various forms of online documentation and reports. This system is a subsystem of a larger knowledge-based system called SciNapse. SciNapse's raison d'etre is to transform high-level simulation problem specifications into executable numerical programs. The electronic notebooks are generated from the same domain knowledge bases that the system uses to perform its primary tasks. These online notebooks are of two different kinds: reference materials and reports. Reference materials are generated from the latest version of the knowledge base, which includes the classes that drive the system, and a network of objects representing meta-information about the system. The reference materials document the system's capabilities and help users understand what the system can do. Reports are generated from the instances created by a run of the system. They document the transformations the input specification underwent in becoming code, and are intended to help a user understand what the system has done.

We have found that our approach to producing documents has both advantages and disadvantages when compared with more traditional approaches to documentation. The advantages are that we can minimize the manual effort that is involved in writing documentation about the system, while at the same time maximizing the accuracy of the documentation that is produced.

The main disadvantage has been the lack of truly appropriate authoring tools built to work in our environment. When we began, we expected the task of creating such authoring tools to be much easier than it has turned out to be. Later in this paper, we explore some of the factors that have caused this to be the case.

Keywords

Knowledge-based systems, intelligent interfaces.

1. SYSTEM OVERVIEW

We present a knowledge-based documentation and information system that is part of a larger knowledge-based system called SciNapse. SciNapse's purpose is to transform high-level simulation problem specifications into executable numerical programs. The information system draws on four distinct sources of knowledge to produce reference materials and reports. They are the knowledge embedded in object classes, which drives SciNapse to perform its principal tasks; instances created by the system while performing its tasks; a semantic network model of the system created specifically for the information system; and an organized collection of real SciNapse examples that are connected to the model.

The documentation system helps automate document production and facilitates producing different kinds of documents from the same information. We support both automated and manual document authoring. The documents are viewed in windows that display "trees of data," the visibility of which is controlled by the reader. Hyperlinks to other document locations are included in the displays when appropriate. SciNapse defines classes, attributes, facets (attributes of attributes), and rules for inheritance and instantiation. The object system and the knowledge base encoded in it are implemented completely in Mathematica®. It was not our goal to develop yet another object-oriented knowledge representation for the sake of doing it, but rather to bring the frequently cited advantages of these techniques [1] to the unique environment provided by Mathematica. SciNapse makes extensive use of mathematical transformations and mathematical programming knowledge, both areas in which Mathematica excels. Mathematica also has some interesting tools for user interaction.

SciNapse [2] is intended to "take the programming out of numerical modeling." The use of computer models in design and analysis is rapidly increasing. High-level specifications that turn into efficiently executable numerical simulation models should be valuable in many application areas. The class of problems we have focused on typically involves the solution of partial differential equations (PDEs). The ability to describe these models in mathematical or application-specific terms, rather than in programming terms, makes the modelers more efficient. Their design work is not burdened with implementation-related complications, and because code synthesis is automatic, they have more time to do the "what if" comparisons that computer modeling enables but frequently omits due to time and scheduling constraints. These modelers need a language to specify their problems. The language must be mathematically precise, reflective of application concepts, and extensible.

Our basic premises have been validated by the successful development of SciFinance™, a system built by specializing and augmenting SciNapse's knowledge[3]. SciFinance aids financial analysts with the problem of valuing derivatives. SciFinance has a significant amount of additional knowledge of this domain, but utilizes the foundation provided by SciNapse to produce C programs to efficiently implement the models. The financial knowledge is less than ten percent of the total system.

2. SYSTEM IMPLEMENTATION

SciNapse, SciFinance, and the information system are implemented fully using Mathematica. The Mathematica system [4] has two separate parts. There is an evaluator/interpreter called the kernel, which implements a very large set of functions (more than 1,000) for dealing with general-purpose programming applied to the domain of mathematics. A sophisticated collection of rule rewriting functions is included. The other component is the support for Mathematica notebooks, which are the windows that provide communication between a user and the kernel. Mathematica notebooks provide the browsing vehicle for the interactive views of our information system.

The notebooks provide a built-in set of WYSIWYG document writing capabilities. The author has control over whether text is treated as text or evaluated by the kernel. There is a documented ASCII external representation of notebooks and a set of kernel functions for manipulating them. The ASCII representation of a notebook is something like HTML or XML [5], but the document grammar is not public.

In addition to normal text, there are three other categories of entities that can be represented in notebooks. They are conventional mathematical notation, a hierarchical grouping construct called a cell, and active areas with the ability to invoke an arbitrary function in response to clicking. All these entities are important to the information system.

3. A PROBLEM AND OUR APPROACH

System documentation has the responsibility of presenting a view that helps system users understand a system, and hence understand how to use the system. For complex systems, the conceptual view that "connects" with users best is rarely obvious. It is not surprising that the earliest versions of system documentation are frequently only marginally usable. This inadequacy is in addition to the problems of accuracy and completeness of the documentation. This situation is usually addressed by completely manual (brute force) preparation of documentation and the acceptance that the documentation will always, inevitably, be incorrect.

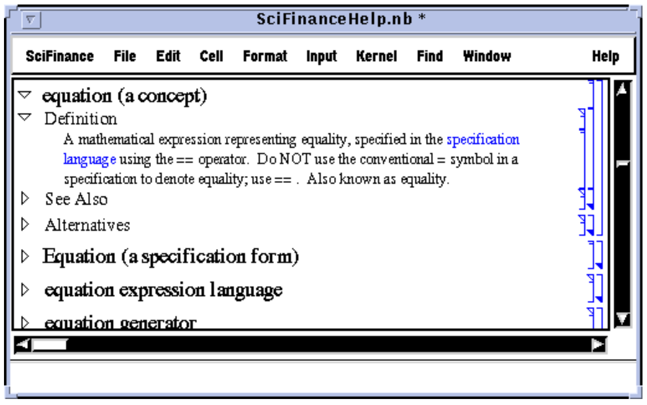
We have conducted an experiment in making more of the documentation production automatic. We built a "semantic network" of knowledge about the system, using the same knowledge representation tools that are used to capture the domain knowledge. We needed a model of the system to organize the information we would need to make available. It needed to be a real computational model, not a metaphorical one. The model is a subsystem of SciNapse, as is all the other knowledge the system uses. And we wanted it to result in a net saving of human effort in producing user documentation over the life of the system. That is, building it and keeping it current should take less manual effort than organizing and maintaining the same information without the model.

The nodes of the network correspond to the basic vocabulary needed to discuss the system. This includes some general mathematical concepts and SciNapse-specific concepts and terms. Only those mathematical concepts that were absolutely central to SciNapse were introduced as general concepts. The SciNapse-specific terms, on the other hand, include all the elements of the modeling specification language and the modeling concepts they implement. The current system has approximately 75 concepts and 350 SciNapse-specific terms. The concepts include truly mathematical terms like "equation" and SciNapse terms like "Coordinate Free Level."

The nodes of the network represent concepts. Each node of the network has six possible relationship attributes: (1) a short definition, (2) a fuller description, (3) a syntax description, (4) examples, (5) alternative concepts, and (6) other arbitrarily related concepts.

We felt it was necessary to have more than one, totally non-specific relationship between concepts. On the other hand, we did not want to refine these relationships to be overly specific. We felt that being overly specific would contribute nothing to our goals. The relationships we have identified, especially given that there is a category called "other," have proven adequate. We will give some examples to make it more concrete.

Figure 1 (below) is an abbreviated snapshot from one of the generated documents showing the presentation of several concepts, including two concepts called "equation." The first, labeled "equation (a concept)" corresponds to the mathematical idea, an equation. The second entry, "Equation (a specification form)" describes the SciNapse specification language construct, "Equation." Not surprisingly, it is a construct used to describe mathematical equations. When working out definitions, we realized early on that it was natural, perhaps inevitable, that the same words would occur both as concepts and as SciNapse language elements, as is the case with equation. When this happens, the super-classes of the two terms will differ at some level and will indicate what kind of entities the sub-class is. We use the appended names (as shown in Figure 1) when this case occurs.


SciFinanceHelp.nb *
SciFinance File Edit Cell Format Input Kernel Find Window Help

equation (a concept)

Definition

A mathematical expression representing equality, specified in the specification language using the == operator. Do NOT use the conventional = symbol in a specification to denote equality; use == . Also known as equality.

See Also

Alternatives

Equation (a specification form)

equation expression language

equation generator

Figure 1: The SciFinance Help Notebook

Examples are important in teaching people how to use SciNapse. In addition to small example fragments (the values of the examples attribute), we build an entire network of nodes for examples, each of which has the same six possible attributes to describe its example. Each example node corresponds to an external example file.

4. PRODUCING AND INTERACTING WITH DOCUMENTS

We will discuss four kinds of reference documents. Two are automatically produced, and two are written primarily by humans. The SciFinance reference document and examples collection are automatically produced. The quick reference guide and specifications turned into active displays are primarily written by humans.

Let's look more carefully at some of the reference documents and how they are produced. The master SciFinance help document is produced by selecting all the appropriate nodes from the network. The nodes are then sorted alphabetically. Then each is rendered as a Mathematica cell or a tree of nested cells, and written into a notebook document.

A Mathematica notebook cell is a rectangular space in a notebook, which may contain displayable content of any sort. Cells may be nested hierarchically. They can provide a tree of display elements. Cells provide an important form of view control. Each cell can be open or closed. When a cell is closed, only its first line is visible. Embedded cells in a closed cell are invisible, regardless of whether they are open or closed. To provide an outline view of the material, we construct most of our documents using nested cells containing the appropriate text as the first line of the cell. Each cell can be easily opened or closed by the user. The open/closed state of contained cells is unchanged when a containing cell is toggled.

Let's return to the notebook shown in Figure 1 and discuss the state of its cells. The **"**Equation (a specification form)" is closed, as are the other cells in this fragment. However, all the sub-cells of this latter cell are closed, except for the first one containing the word "Definition." You can toggle either by clicking one of the triangular icons on the left-hand side of a cell, or by double-clicking the brackets on the right-hand side of the cell. This marking of cells and the triangular icon is optionally invisible. The brackets show both cell boundaries and cell nesting. In a situation where the user does not need to see this information, it is can be omitted.

5. GIVING THE VIEWER MORE CONTROL

Hyperlinks are one way to separate detailed information on a topic from the references to the topic. However, using the HTML browser scheme has some consequences. First, the user has most of the responsibility for remembering the context surrounding the hyperlink. When someone is trying to learn information, both context and details are relevant. Second, moving back through the stack of pages visited is clumsy. Third, the only hyperlinks available are those explicitly provided by the author. The viewer has no way to add them. Fourth, the time to produce a page in response to clicking on a hyperlink may not be constant.

The nested cell scheme can be used to address some of these issues. When a body of material can be organized hierarchically, it can be mapped to tree of cells. A reader can leave cells closed to suppress the details while opening any that contain details of interest. Any subset of the cells that the reader finds useful may be opened. The rest of the tree skeleton provides the context.

Another aspect of Mathematica notebook interactivity is the ability to make an area of a notebook cell active, and to provide a function to be used with that area. When the viewer clicks on the area, the function is invoked. Words corresponding to concepts are made active in the notebook described above. This correspondence is indicated by color and by underlining. (The appearance of the notebooks is not accurately reproduced in some of the figures in this paper. However, Figure 1 should be close.) In this case, clicking an active word invokes a hyperlink jump action. The notebook moves the concept into view. This can trigger several different actions. If the target is in the same notebook, the target is scrolled into view. If it is in a closed cell, the cell is opened so the target really becomes visible. If the reference is to another notebook, the notebook is opened as a separate window and the target is brought into view. There is a stack of locations, and the reader can move backwards as with an HTML browser.

So to give the viewer more control, we provide two different forms of information organization and natural ways to navigate through both. The hierarchy of cells is used for all information about a concept. We write these notebooks with all the top-level cells closed, providing a natural table of contents from the text itself. Phrases that are defined elsewhere are marked distinctly. If one of these is contained in an open visible cell, it is visible. Clicking on it will jump you to the definition of that term. The open/closed cell paradigm is very simple and is used in the same way everywhere we employ it. The hyperlinks and page stack paradigm we use is essentially the HTML browser paradigm, so users will normally already be familiar with it.

6. MAKING EXAMPLES AVAILABLE

Example specification files are processed by the information system to connect specification files with the documentation about the specification language. Figure 2 shows a portion of an examples notebook. BlackScholes1D is the name of this example. It corresponds to an external file that could be processed by SciFinance, named BlackScholes1D.eqn.

```
BlackScholes1D
Definition
Equation Generator for the 1D Black-Scholes
  Equation with greeks  BlackScholes1D.eqn
Examples
  BlackScholes1D[Keywords->{Vega,Rho}]
  BlackScholes1D[D0->0]
  BlackScholes1D[Keywords->{Vega,Rho}, D0->0]
Syntax
  BlackScholes1D[] |
  BlackScholes1D[Keywords->
                 {<subset of Vega,Rho}>},
                 <var>->value]
Description
  The equation generator for the
  1D Black-Scholes Equation
```

Figure 2: A Fragment from the SciFinance Examples Help Notebook

An examples notebook is built much the same way as the previously described reference notebook. Whereas concept and terminology nodes are defined manually by the knowledge entry process, the example nodes are determined automatically by searching a list of external directories. For every file found meeting some naming criteria, a node is built. These actual example files are parsed. There are comments at the beginning of each of the files providing the descriptive information that becomes the values of the attributes for the node that is built. This very late binding of values seems appropriate for examples. Example files can be inserted and deleted at any time, and their comments can be edited at any time prior to running the examples collector. A reader of the examples notebook can do the same type of interactions discussed above: either opening/closing cells or following a hyperlink. Here, however, following a hyperlink takes the reader to the full display of the actual example specification file, rather than to a concept definition.

Information can be “encoded” by the names and organization of the directories containing the examples. Meaningful directory names help identify the nature of the examples. Under each directory is listed the actual examples found in that directory and sub-directory relationships are captured as a tree of names. Figure 3 (below) shows this part of the notebook for a typical SciFinance system. The terms have domain significance. If the cell corresponding to any one of them is opened, the list of examples for that category is visible. Each line is a cell containing a list of the names of the examples found in that directory.

```
Examples\
  correlation\
  other\
  pathdep\
    asian\
    barrier\
    lookback\
    vanilla\
```

Figure 3: The Hierarchy of SciFinance Example Directories

7. DEVELOPER- STRUCTURED DOCUMENTS

The reference guides discussed above are generated totally by gathering and ordering nodes from the knowledge network. All of the information comes from the network. These documents have simple structures. The content is collected, ordered, and rendered in hierarchical cells. We felt the need for documents with more complex structure, mostly written by humans. We support this by allowing writing and processing of fairly arbitrary Mathematica notebooks. Recall that Mathematica notebooks provide “WYSIWYG” writing capabilities.

The “Quick Reference Guide” is in this category. This kind of document can have any text its author wants. Since documents of this type will be using terms found in the knowledge network, we wanted it to be easy to create hyperlinks from references to nodes of the network. Therefore, we used access to the knowledge network to construct the hyperlinks automatically.

In a live cell, all terms exactly matching a network node are automatically converted into hyperlinks by a preprocessor. This relieves the author from explicitly identifying references to nodes. The author may choose a cell type that does not support auto-linking and the discussion will remain uncluttered by link markings.

We use the same conventions to indicate parts of the display as hyperlinks to the concept node of the same name, as defined in the Reference Notebook.

We use very similar preprocessing to automatically produce “active” displays of specification files from real specifications. Each word in the specification that corresponds to a term in the language is converted into a hyperlink jumping from its occurrence in the specification file display to the help notebook explanation of the term.

These techniques would be applicable to documents of any kind. We have not yet applied them to long documents needing a professional appearance because of the lack of some needed features in the Mathematica notebook system.

There is another point worth noting. Our reference documents have been at the ends of the spectrum of automatic versus manual structuring. This need not be the case. Documents that combine approaches are certainly possible. One useful technique would be to use more sophisticated pattern matching to decide when text is converted into a hyperlink and what that hyperlink points to. More sophisticated algorithms for the basic document structure would also be easy to implement, given the tools we have in place.

8. KNOWLEDGE AUTHORING

For the most part, human authors have populated the network of information we have been discussing. They specify what nodes should exist, and the values for their attributes. The authors must mark a term in the text as a reference to some other node. Unlike the processing of the Quick Reference Guide we knew that in the knowledge network the mixture of text using words that could refer to concepts and those that are actually meant to do so would only be handled reliably with explicit indications of a concept reference.

The representations of the network take full advantage of taxonomic inheritance and computed methods. Using methods we have defined some aids for the knowledge authors. Two interesting ones are for the values of “Alternatives” and “See Also” attributes.

The Alternatives for a concept can, in some cases, be defined automatically. This is the case when the entire set of concepts corresponds to a set of classes that are descendents of a particular class, perhaps filtered in some way. Computing alternatives values in this way has several advantages. Since these are the classes that actually drive SciNapse, they will certainly be a correct representation of the system’s implementation. Specifying how to compute them is less work than enumerating them manually. For any node in the set, its Alternatives are the set with the node itself removed.

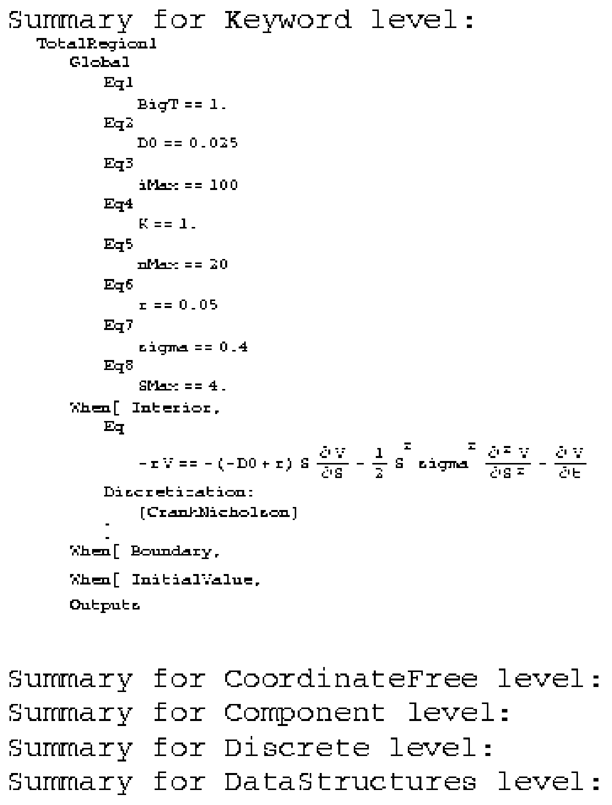
Another aid exists, one for the See Also. This value is computed as an explicit, manually entered list conjoined with all the concepts referenced in any other node’s attributes. That is, we allow a knowledge author to enter an arbitrary set of concepts and call them the See Also set. Then, we add to that set any concept mentioned in specifying any of the node’s other attributes. This mechanism is always used to compute the value of a See Also, although either or both lists may be empty.

Furthermore, an author may construct a generic documentation method for a class of objects which will compose a documentation string from a template of substrings and from attributes that are common to all members of the class, but whose attribute values may distinguish one class member from the others.

9. LEVEL SUMMARIES

SciNapse solves the problem of converting a very high-level problem specification into a running program by refining the problem through a series of levels. It starts at the Keyword Level, a very abstract level of problem representation. It terminates when the Code Level is finished. A Level Summary notebook is produced for each SciNapse run to document the progress of the synthesis refining the problem.

Figure 4 (below) is part of one of these notebooks. It uses the same kind of hierarchically grouped cells to provide the tree of information in a Level Summary. The summary for each level starts with a title and sub-divides into named geometric regions. Eventually, equation names and the equations they name are grouped. In Figure 4 we go from Summary for Keyword Level to TotalRegion1 to Global to Eq1 to the equation that is named Eq1. As discussed before, the open or closed status of the cells is easily controlled by the reader. In the Level Summaries, there are also active areas of the cells. Each active area corresponds to an object. In fact, each one corresponds to an object instance, which was produced as part of the process of transforming the original specification into the target program.



```
Summary for Keyword level:
  TotalRegion1
    Global
      Eq1
        BigT == 1.
      Eq2
        D0 == 0.025
      Eq3
        iMax == 100
      Eq4
        K == 1.
      Eq5
        nMax == 20
      Eq6
        r == 0.05
      Eq7
        sigma == 0.4
      Eq8
        SMax == 4.
    When[ Interior,
      Eq
```

$$-r V == -(-D0 + r) S \frac{\partial V}{\partial S} - \frac{1}{2} S^{2} sigma^{2} \frac{\partial^{2} V}{\partial S^{2}} - \frac{\partial V}{\partial t}$$

```
      Discretization:
        [CrankNicholson]
      .
      .
    When[ Boundary,
    When[ InitialValue,
    Outputs

Summary for CoordinateFree level:
Summary for Component level:
Summary for Discrete level:
Summary for DataStructures level:
```

Figure 4: Part of a SciFinance Level Summary

The mathematical notation most appropriate for equations at different levels changes. We honor these mathematical conventions by using Mathematica's ability to display very authentic mathematical notation.

10. BROWSING OBJECTS

As mentioned above, the names of objects appearing in the Level Summaries are active. Clicking on one will cause the execution of a function that creates and opens a notebook containing a description of that object. The description includes the names of the parent class and children objects, if any. It also includes some subset of the attributes with their values, and includes a subset of the facets with their values for each attribute. We use the same technique of hierarchically grouping cells to capture the relationships among the pieces of text producing the display. The reader has the control to suppress details by closing cells exactly as we discussed above.

Whenever the name of an object is displayed, it is active. When clicked, it behaves just like the clicking of an object name at the Level Summary. A function is executed that creates and opens a notebook containing a description of that object. A list of objects currently displayed in notebooks is maintained so that a second click anywhere on the same object name does not create a new notebook. It just de-iconifies the existing window and brings it to the top of other windows on the screen.

Attribute and facet values are displayed using the same mathematical notation described for the Level Summaries. SciNapse represents the same equation at different levels by using different object/attribute combinations for each. Hence, the Level Summary hides these bookkeeping details and makes it easier for the user to follow the progress of equations, with their transformations, from level to level. When looking at raw objects, more implicit, contextual information is needed to understand what they mean and how they fit into the solution of the original problem.

We provide three different views for objects, aimed at different kinds of users. First, is a view for SciNapse users. Second is a view for SciNapse developers. Third is a view for someone who needs specific information on the structure of the object with respect to the object taxonomy.

The user view presents attributes of an object ordered by the level with which they are associated. This level is the one during which the attribute acquired a value. Furthermore, only attributes marked as having "math" values are displayed. This is a meta-marking that the knowledge base authors use to mark appropriate attributes. This filters out attributes that would be of interest to developers, but meaningless to a "naïve" user.

The developer view presents attributes of an object ordered by level, just like the user view, but does not filter out any attributes. Each attribute is displayed with its value. SciNapse uses facets of attributes for a variety of bookkeeping tasks. The facets that hold this bookkeeping information are filtered from the developer view in the interest of reducing clutter. But, if other facets for an attribute are found, they are displayed with their values. The cell displaying an attribute is created closed. The facets will not be visible unless the user wants to see them and explicitly opens the cell.

The object structure view organizes and filters the attributes and facets in a way different from either of the other views. It divides attributes into those that have local values and those that inherit values from some ancestor. The level of an attribute is ignored for ordering. The cells for attributes of either kind are sub-cells of a label, either local attributes or inherited. Either entire group may be toggled open or closed. As with the developer view, the attribute cells are created closed, with only their values visible. However, for this view, all facets with local values are shown. No filtering of the bookkeeping facets occurs. This view is intended to readily answer a different set of questions than the other views.

It should be noted that these three object views are produced by editor sibling classes that share methods and the attributes that control the views. It would be very easy to change any of them, or to create new views with other characteristics.

11. SUMMARY AND CONCLUSIONS

Knowledge is central to the functioning of SciNapse. The knowledge representation tools we are using make both knowledge and meta-knowledge readily available. It seemed expedient to use the primary knowledge representation tools to also build a model of SciNapse that helps organize information about the system. Both purely descriptive textual information, identified by its place in the model, and the classes and instances that actually control SciNapse's operation were readily available. The decision to approach system documentation in this way was influenced by the fact that human resources were at a minimum at that point in the project. We also knew that if the cost to document in this way was too high, we could do a one-time conversion to a purely manual system.

Having this network in place, we found many ways to take advantage of the declarative knowledge. This strategy required a knowledge representation system. We built the representation system in Mathematica, but Mathematica did not bring functionality that made the system easier to build.

We have developed a uniform style of organizing all the notebooks. This style uses functionality of Mathematica notebooks. Documents are typically organized hierarchically. That is what a table of contents is intended to convey. We have used the behavior of nested cells in Mathematica notebooks to create documents that are, in some sense, their own tables of contents.

In contrast, hyperlinks in traditional HTML documents provide only one way to move from the occurrence of a reference to an idea, to an elaborated discussion of the idea. All the work producing the correct pagination and hyperlinking has to be done by the human author, in advance. The reader has no choices other than those that the author has built in. We believe that supplementing this style of interaction with the interaction available through a reader's control of cell visibility gives the reader useful ability. It is also important that using this control be instantaneous, as it is for our pre-computed cells. A reader would be inhibited if cell expansion could take large or varying amounts of time. This aspect of our presentation addresses the same controllability issues that the Microsoft Windows Explorer does for a tree of folders. However, that technique addresses only the names of files and folders, not the contents of the files.

12. PROS AND CONS

The main disadvantage we found was the cost of authoring, of both knowledge for the system and documents to be processed by the system. Knowledge authoring would become more efficient with the introduction of standard GUI tools for Mathematica notebooks. These have not yet become available. Even though Mathematica has had a very sophisticated document model underlying the notebooks, writing complete professional looking documents manually in Mathematica is still too time consuming. New versions of Mathematica and some Mathematica-based products and tools will eventually make this job much easier. We seriously underestimated the time before this software would become available.

We can compute using our knowledge representation very easily. It provides inheritance, methods, and the techniques designed to make the computations with knowledge easy and natural. There are a number of powerful functions Mathematica provides for creating and reading notebooks. They address dealing with notebooks syntactically, and help simplify what would otherwise be very tedious code to write. These functions are not specific to the content of the notebooks. Together, these sets of computational abilities create a large space of document production strategies. We have only explored a small part of this space.

Our approach to document creation eliminates the possibility of making certain kinds of errors. As we mentioned earlier, the automatic computation of all or part of a value both reduces the manual work of updating this kind of information and makes it necessarily correct since it is computed from the knowledge base itself. It is also possible to do validity checking of some properties of the knowledge network. It is easy to check that every alleged reference to a node is to a node that actually exists. It is also possible to check that starting from a list of nodes, every node in the network can be reached. These can be thought of respectively as detecting "dangling references" and "garbage." For us, the advantages have outweighed the disadvantages. Since the system is new, a more quantitative measurement of costs for comparison is not yet possible.

13. ACKNOWLEDGEMENTS

SciNapse would not exist without the contributions of Curt Randall and Stanly Steinberg.

This work was supported in part by the National Institute of Standards and Technology, Advanced Technology Program under Cooperative Agreement #70NANB5H1017.

14. REFERENCES

[1] R. Fikes and T. Kehler, "The Role of Frame-Based Representation in Reasoning," CACM, September, 1985, pp. 904-920.

[2] R. A. Akers, E. Kant, C.J. Randall, S. Steinberg and R. L. Young, "SciNapse: A problem-solving environment for partial differential Equations," IEEE Computational Science and Engineering, July-September, 1997, pp.32-42.

[3] J. Gatheral, Y. Epelbaum, J. Han, K. Laud, O. Lubovitsky, E. Kant and C. Randall, "Implementing Options-Pricing Models Using Software Synthesis", Computing in Science & Engineering, November-December, 1999, pp. 54-64.

[4] Stephen Wolfram, Mathematica (3rd ed.), Cambridge University Press, 1996.

[5] Extendable Markup Language: XML. See http://www.xml.org