OONA, MAX AND THE WYWWYWI PRINCIPLE: GENERALIZED HYPERTEXT AND MODEL MANAGEMENT IN A SYMBOLIC PROGRAMMING ENVIRONMENT

Hemant Bhargava
Michael Bieber
Steven O. Kimbrough
Department of Decision Sciences
University of Pennsylvania

1. INTRODUCTION

Progress in decision support systems comes in a variety of forms. These include increased understanding of the organizational consequences of machine-mediated decision making, improved knowledge regarding how to implement DSSs (decision support systems), advances in techniques for managing and maintaining DSSs, and improvements in DSS technology that result in enhanced functionality or reductions in the cost of building particular DSSs. The progress in DSS reported here falls under the latter category. The research motivation for this work may, briefly, be described as follows.

A DSS is a software tool that facilitates working with data and models. It is widely agreed that building particular DSSs can be done faster and more economically if general-purpose, reusable software is available for performing standard DSS functions (Sprague and Carlson 1982; Kimbrough 1986). Economies arise both from reusing existing software and from working in a familiar, standardized environment. For certain of the DSS functions or components (e.g., business graphics, report writing, database management) design principles and implementation techniques are well advanced and commercial products of high quality are on the market. For other DSS components (e.g., model management systems, hypertext systems) there is broad agreement that they are needed and valuable, yet comparatively little is known about what functionality the components should have or how that functionality should be implemented.

In the case of hypertext, a few systems are on the market but requirements and capabilities for DSS hypertext interfaces are far from being fully worked out and understood. Research on these systems is only beginning. In the case of model management, there has been extensive research but it has not yielded consensus as to exactly what model management is or what features a model management system should have. Research model management systems of limited scope have been implemented. For example, WHIMS (Wharton Interactive Modeling System) manages microanalytic simulation models (Katz and Miller 1986). PLANETS (Production Location Analysis Network System) allows analysts to define a business environment and specific business assumptions. It provides automatic feasibility checks for any potential scenario and performs automatic generation, solution, and interpretation for mathematical models. It is used by General Motors and has resulted in savings in excess of $1 billion (Breitman 1987). Commercial model management systems are, however, non-existent.

We have explored hypertext and model management in the context of DSS and have implemented two prototype systems (Max and Oona) currently in use by the U.S. Coast Guard. The purpose of this paper is to report on and discuss our findings. In section 2, we begin by presenting background material and briefly discussing the central technical ideas (generalized hypertext and model management) for these systems.

2. BACKGROUND

During the next ten years, the U.S. Coast Guard will invest more than a billion dollars in acquiring replacement assets (ships, airplanes, helicopters, etc.) for its present, aging fleet. The cost of operating and maintaining these assets will be several times the acquisition cost. Because of the importance and complexity of the capital asset acquisition problem, the Coast Guard has initiated a project aimed, in part, at systematically developing software for support of acquisition decisions. The goal is to produce a series of specific decision support systems (also called knowledge-based decision support systems, or KSSs) in support of particular acquisition projects, but based on generic and highly reusable system software. The work reported here was done as part of the Coast Guard's KSS project.

Early on in the project, the decision was taken to build a general environment--or shell--for building and running particular KSSs. The concept of a KSS shell is similar in some ways to that of a DSS generator (Sprague and Carlson 1982, p. 11). The idea is that the shell is a powerful, general-purpose software system that facilities the construction of particular DSSs (or KSSs) and provides a variety of features. Because the technology employed (symbolic programming) and the feature set implemented (hypertext, expert systems inference engines, extensive model management facilities) in our KSS shell design

greatly subsumes that of traditional products that go by the name of DSS generators (Express, IFPS, Empire), it seemed appropriate to use the term shell to describe the focus of our efforts. (See Kimbrough [1986] for a related discussion of the idea.)

There are three essential modules in our DSS (or KSS) shells. First, there is a high-level process that makes inferences and provides control over the entire DSS. As currently implemented in both Max and Oona, this process is in the form of a Prolog meta-interpreter (Sterling and Shapiro 1986) and as such resembles an inference engine for a conventional rule-based expert system. Because space is limited and because our current implementations (Max and Oona) are not innovative in this regard, we will not discuss further the high-level control. The second essential module is a model management system. (In the existing prototypes, written in Prolog, data is treated as a special type of model, so the model base management system manages both data and models.) We discuss design concepts and principles for model base management in section 3. Finally, the third essential module is a user interface, or display, (sub)system. Our design and implementations have been much influenced by hypertext concepts, although we have added features not found in other systems. We discuss these in section 4.

3. MODEL MANAGEMENT

Decision Support Systems employ, as do human decision-makers, a variety of models to solve problems and recommend courses of action. A model is simply an approximation of some real-world phenomenon, or a set of assumptions (e.g., architects construct models of the buildings they plan to build, management scientists model service departments as queue systems, etc.; here we are concerned with computer-executable models). The model management system must manage the tasks involved in making, evaluating, and explaining the consequences of these assumptions.

Hence a model management system must facilitate expression of knowledge about models, exercising this knowledge, and explaining the results obtained from using these models. It may be argued that such functionality could be included in any DSS by writing procedures to do each of those tasks; that would be to miss the point. The idea of an MMS is to provide this functionality in a general-purpose form (or as general as possible) which can be applied to a wide class of applications or sets of models.

Broadly speaking, we recognize three user roles for an MMS, corresponding to the three activities mentioned in the previous paragraph: model builders, who construct the models and provide the domain knowledge that enables the MMS to perform in that domain; model users and analysts, who exercise these models or solve and analyze actual problems by providing the data and create reports; and decision-makers, who may browse through the reports and make decisions based on these analyses. They are responsible for these decisions and must be able to justify them. Decision-justification, often omitted in models of decision-making, is an important function of a DSS and must be supported by its model management component. In fact, we take the stronger view that the main purpose of a DSS is to assist users in constructing and evaluating arguments for courses of action (Kimbrough 1987).

The above discussion leads to the following two metaphorical statements about model management systems: an MMS is like an operating system for models, i.e., it handles models in general, as an operating system handles files, irrespective of their content or application; and an MMS is like an expert system for modeling: it attempts, through intelligent-seeming behavior, to fill the position of an human expert in modeling. This contrasts somewhat with the view, taken in previous literature, in which a model management system has been likened to a data base management system ("models, like data, need to be stored, retrieved, shared" [Elam, Henderson and Miller 1980; Liang 1986]), and described as "a structured milieu for storing, manipulating, and retrieving models" (Dolk and Konsynski 1984). The analogy with DBMSs, though not inappropriate, does not sufficiently describe the functionality of a model management system. We attempt to develop a list of features by looking at the steps in a modeling life cycle and the user roles that an MMS should support. We will develop these using the following example.

Example 1(A)

We consider here a part of the Vessel Acquisition process in the U.S. Coast Guard. The process includes measuring various attributes of different vessels and evaluating costs over their estimated life. Assume that we are interested in a set of Vessel characteristics, Life Cycle Cost, Designer, and Builder of the Vessel. Also assume that the total Life Cycle Cost (LCC) is computed using a simple mathematical model that sums various costs, in this case, Acquisition Cost (AC), Maintenance Cost (MC), and Personnel Cost (PC), i.e., LCC = AC + MC + PC.

Each of these components is computed using a mathematical model. For example, the Maintenance Cost over the vessel's life is computed by adding the discounted values for Maintenance Costs, for each year in the vessel's life. The maintenance cost for year j, MC_j, is calculated by inflating a Base Cost by an inflation factor and then discounting it to get the present value. Hence, if b is the Base Cost, i is the percent annual increase in maintenance cost, r is the discount rate, and N is the vessel's life span, we have

$$MC = \sum_{j=1}^{N} MC_j / (1+(r/100))^j$$

Letting pv(T,R,A) be a function that returns the present value of a cash flow A, that occurs T periods from now, with a discount rate R, we have

$$MC = \sum_{j=1}^{N} pv(j,r,MC_j), \text{ where } MC_j = b*/(1+(i/100))^j$$

$$MC = \sum_{j=1}^{N} pv(j,r, b*(1+(i/100))^j).$$

Further assume that the discount rate r varies over the life span, and that the rate for each year is computed using a model M1 that uses forecasts by OMB and by RAND Corporation. For the moment we will ignore other details of this example. We now examine what an MMS should do in terms of the three user roles discussed earlier, using the above example for illustration.

(a) **Model builders**: A model builder would construct the various models to support the Vessel evaluation and comparison process. In the above description, we decomposed the entire problem into smaller problems. For example, the total life cycle cost is computed as a simple summation of its components, though there are more complicated models underlying each component. The MMS must similarly support building of complex models from simpler ones, either by composing models or modifying them. This is important since the sub-models may have independent existence and multiple use. In order for the builder to know about existing models (e.g., present value model), the MMS should be able to describe these models in terms of their essential characteristics (e.g., the parameters and formula for the present value model). The modeler should be able to express mathematical relationships in a natural mathematical form without having to program the computation.

(b) **Model users**: Model users or analysts would be interested in analyzing or evaluating particular vessels by providing values for different attributes of each vessel. The MMS should identify and report on the kind of data required for this evaluation and at a level of detail that is at the user's discretion (For a vessel, we need its life cycle cost and other characteristics; for the life cycle cost, we may need to know the acquisition cost, maintenance cost, and personnel cost; and so on.) It should also facilitate the declaration or acquisition of this data in a flexible form. For example, for vessel 1, the life cycle cost could simply be a multiple of the life cycle cost of vessel 2, and the cost for vessel 2 may be an expected value of some estimated costs, whereas for vessel 3, life cycle cost may be computed at a further level of detail, i.e., by computing the various components. Again, the MMS should be able to explain various models and components of models to the users.

(c) **Decision makers**: Finally, assume certain reports have been created that contain outputs from models, based on which decisions have to be made. The user may want to know how certain results were computed, what models were used, where the data was obtained, etc. Most of these queries would be ad hoc; hence, the MMS must have a general capability to provide this information based on its knowledge about the models. For example, an interaction between the user and the system may be to the following effect:

User: Why was vessel 1 selected?

System: It had the minimum Life Cycle Cost ($1000 m).

User: How was this number computed?

System: Life cycle cost = Acquisition cost + Maintenance cost + Personnel cost; for vessel 1, Acquisition cost = $200 m, Maintenance cost = $300 m, Personnel cost = $500 m. Hence Life cycle cost = $1000 m.

User: What is Maintenance cost and how is that computed?

Below, we briefly describe a set of features that need to be present in a model management system. In sections 5 and 6, we describe how these are addressed in two implementations, Oona and Max. This discussion refers to data models, mathematical models, and deductive models. A data model is simply a vector of the type $[M_1,M_2,...M_n]$, where the M_is are themselves arbitrary models. A mathematical model M is a symbol that assigns a value V to a tuple $T = (t_1, t_2,...t_n)$ according to some rule. A deductive model, in the present context, is like a set of IF-THEN condition pairs and stipulates a set of conclusions corresponding to a set of premises being satisfied.

Feature Categories of a Model Management System

(a) Construct new models, and integrate with existing ones: this involves adding models, as well as knowledge about the models, to the model base (e.g., to add a knapsack model to a mathematical programming knowledge base, one would need to set up the general form of the model, declare information about solution algorithms and programs, and relate it to other integer programming paradigms). A new

model may be created from scratch or be set up by combining or modifying other existing models.

(b) Identify and formulate problems into known models: formulate a model to determine the optimal product mix (Bradley, Hax and Magnanti 1977) as a linear program by defining the decision variables, objective function and constraints.

(c) Describe models, parameters, other related models: the system must be able to describe what it knows (e.g., before using a product mix model, a user may want a description about it, how to interpret its results, related concepts like the simplex method, or where to look for aditional documentation).

(d) Identify data requirements: the system should figure out what data is required to execute a model (e.g., for the product mix model, it would need a list of raw materials with availabilities, final products with unit profits, and activity coefficients).

(e) Data extraction: very often the data to run a model would be available elsewhere in the database and could be obtained automatically by a set of projections from different relations in the database. The above two are important steps in integrating models with data, or more precisely, integrating mathematical models with data models, while retaining the independence between them. They have received some attention in research on expert database systems (Kershberg 1986; Jarke and Vassiliou 1984), but not much in modeling systems.

(f) Verify model validity: it may be possible to perform certain automatic checks for model validity (e.g., there is probably an error if a new model being set up adds the outputs of two models, one of which output is a character string).

(g) Verify data validity: the system should recognize an error if, in the present value model, the value for the interest rate has dimension feet.

(h) Solve/execute models: the system would need to determine an appropriate program or solution technique to solve a model, instantiate the model, and execute it. Execution includes rewriting a model in an equivalent form, re-expressing it after certain substitutions, computing the output as a number, computing an analytic solution, or drawing certain conclusions.

(i) Explain model results/outputs: this involves an interpretation of the results, how the results were computed, parameters and their values, source and reliabililty of data, and other factors required to justify a decision made using these results.

(j) Analyze changes: this includes computing the effects of changes in parameter values, either numerically or analytically (e.g., the rate of change by taking first derivatives).

(k) Manage the model base: the system needs to manage the storage, retrieval, access (perhaps concurrent and shared) and updating of models, and integrity and consistency of the model base.

This list is not meant to be definitive. Various researchers have proposed different feature sets (Liang 1986; Geoffrion 1987; Applegate, Konsynski and Nunamaker 1986), and it is not yet absolutely clear exactly what features must be supported in modeling systems. The above model management activities correspond to different phases in a modeling life cycle and are meant to serve the three categories of users mentioned earlier. For example, the end-users in categories (b) and (c) would benefit most from activities (b), (c), (d), (e) and (i), whereas model base developers would benefit from a high-level implementation of (a), (f) and (h), as well as a facility to build in the knowledge required for the other activities.

Model Management Implementation

Given the above set of features for a model management system, we identify four specific issues as being central to the implementation of such a system: model representation, model base organization, a modeling language to express knowledge about models to derive this representation, and rules to process and manipulate this representation.

The issue of model representation has been addressed from the viewpoints of database systems, artificial intelligence, and decision support systems (Bonczek, Holsapple and Whinston 1982; Konsynski 1980; Blanning 1986). Three main techniques have been used for model representation: models as subroutines, models as data, and models as statements in a modeling language (Sprague and Carlson 1982; Dolk and Konsynski 1984; Blanning 1986). Our implementation approach is to represent models symbolically, as symbols that represent various concepts related to a model. Knowledge about models is represented as statements in a modeling language. Due to the lack of distinction between programs and data in a symbolic language, model-independent manipulation rules can directly manipulate this knowledge ("Thought consists of the manipulation of sentences in the language of thought" [Pollock 1986, p. 163]), and we can view models both as statements and as data. Our philosophical approach to model representation is that models are (complex) individuals, about which predication may be applied. The model individuals are named (using symbols) in the model declaration language. Properties of these individuals may be declared or inferred using the

declarations in the model declaration language (and inference engines for inferred properties). These declarations may be logically quite complex, including such features and quantification across sentence operators if need be. Given this logical point of view, Prolog is a natural choice for a programming language to implement the model management system, but it is hardly necessary. Any symbolic computing environment will do.

4. THE DISPLAY SYSTEM

Meeting the following requirements is, we believe, largely the responsibility of the user interface (or display) module for a generalized DSS shell.

1. Users (and system builders) should have access to any system entity (e.g., data, models, meta-knowledge about data and models) consistent with security and integrity requirements.

2. Access to system entities should be facilitated by context. For example, a component of a model would be a system entity. Access to this component might be had noncontextually by allowing the user to request a standard report on the component at any time. Context-based access could occur, by having the user display the model, point to the model component, pull down a window menu of available options, and choose the option that generates the standard report on the component currently pointed to. Besides facilitating such context-dependent access to information, the system should provide material assistance to a user in traversing (navigating through) various contexts and returning to the desired point in the system's context space.

Together, these two requirements generate what we call the WYWWYWI ("what you want, when you want it," pronounced "weeweewee") principle for DSS interface design. Implementing them, and supporting communications to the model management system, is the job of the user interface. It is difficult to overemphasize the importance of a good interface.

> Understanding model behavior may depend less on the actual models than on how the models are packaged....The user interface should facilitate and not hinder the editing process....Developing insight on model behavior is ultimately a process of discovery, of finding trends, surprising behaviors and comparing the behavior of the model to what is expected or observed. [Jones 1987]

It is well established that people can only cognitively handle small amounts of information at one time (Miller 1956). When faced with complex problems, people will decompose them into increasingly smaller subproblems until they are of a manageable size (Shneiderman 1987). Different people will choose to break problems down in different ways and by different cognitive methods. A DSS should help us to understand and work in a complex domain and should aid us in the decomposition of a large problem into manageable chunks. In implementing this we must ask: Can we provide an interface that allows a user to regulate the amount of information visible at any given time while retaining access to all other knowledge concerning the problem domain? Can we provide an interface that gives a user the ability to format both the viewing environment and the knowledge displayed within it? Generalized hypertext (an extension of traditional hypertext) enables the implementation of WYWWYWI for an affirmative answer to these concerns.

Hypertext

Traditional hypertext is the concept of linking related information entities or *nodes* and facilitating the rapid computerized transfer between them. Any computer program that supports this informational structure can be called a hypertext system. For example, in a hypertext encyclopedia article, a user could specify a reference and the system could automatically transfer him/her to the article referred to in the reference. A mechanized version of hypertext was first proposed in Bush's Memex system (Bush 1945). The first computerized hypertext system was implemented by Engelbart more than twenty years ago (Engelbart and English 1968). Heretofore, constrained by the availability of abundant and inexpensive computer processing and memory, hypertext has only recently enjoyed widespread implementation and use. (For a discussion of hypertext design issues see (Akscyn, McCracken and Yoder [1987]. For a detailed survey of the field see Conklin [1987].)

Hypertext encourages (but does not restrict the user to) the display of information in small portions. Nodes should ideally represent a single concept and the links should implicitly or explicitly convey the relationship between two or more nodes. Hypertext systems enable people to communicate in a non-sequential, multi-path fashion. Information is not restricted to a single thread of text with one start and one end point as in a normal text document and users are free to follow links and explore information nodes in any order they wish. Thus a central distinction of a hypertext system is that it is specifically designed to allow users to *browse* through volumes of information (Marchionini and Shneiderman 1988). The user and builder should be able easily to check what further information is available at any time, and then transfer to that new knowledge. Of course, it is important to implement this in a manner that will not overwhelm the user of the system.

Generalized Hypertext

Generalized hypertext will give both the builders and users of applications and models access to hypertext features in all functions and parts of the DSS shell and its applications. It is the mechanism for implementing the WYWWYWI concept in which the user is able to regulate the amount and format of information visible at any given time at all levels of the DSS process, including DSS construction, analysis and control.

Nodes and Buttons

The concept and implementation of nodes varies from one hypertext system to the next. In traditional systems, a node is the origin or destination of a link. Sometimes it will be a point (e.g., a word); more often it is one or more paragraphs of text. Many systems fix the size of a destination node (Conklin 1987). We have generalized the concept of the node to that of a generalized hypertext *button*. A button is simply any entity known to the system. Any button may be linked to any other button. This means that any system entity (DSS application, model, datum, document, key word, link, etc.) can be linked to any other entity. Alternatively, a button could be logically connected only to the knowledge reasoned about it. As part of our research, we are developing a formal framework for generalized hypertext buttons.

Linking

Traditionally, hypertext systems have been intended to deal with the presentation of ideas. Builders of hypertext nodes are generally authors expressing their thoughts or research in a nonlinear fashion, representative of the structure of their ideas. Users are readers who initially may follow the links established by authors, but are able to create additional links and make on-the-spot annotations at will.

In most hypertext systems, links are (usually) explicitly established by an author or user, although occasionally they are implicitly generated by a system action. Generally two types of explicit links are supported, *organizational links* and *referential links*.

Organizational links exist in systems supporting a structured node hierarchy similar to a semantic net. Textnet (Trigg 1983), for example, has *toc* (table of content) nodes that contain an outline of the underlying text structure for a given relation. Organizational links connect the parent toc node with its corresponding children, either lower level toc nodes or *chunk* nodes containing the actual text.

Referential links are links employed by the author or user to establish a connection between two arbitrary nodes. Many systems allow these referential links to be *typed*. For example, Textnet, a system for authoring and critiquing documents, has 54 link types such as "refutation," "explanation," and "generalization." These link types are important in establishing a structure for the node network and can guide a user in visualizing the relationship between a series of nodes.

In a traditional hypertext system, implicit links can be generated by the system when it performs a keyword search. Normally, the user browses the information base by explicitly traversing established links among nodes. However, when a user searches for an arbitrary keyword, the system will often create internal links to move the user from his present position to a node representing the search outcome (Conklin 1987).

Links may have associated weights or other characteristics to direct the linking process or they may have procedures attached that are invoked when the link is traversed. Alternatively, they may simply contain or represent pointers between nodes.

One of the main differences between text found in a DSS and that in a more standard hypertext system is the type of information represented. Traditionally, hypertext systems represent ideas recorded by authors. Most links tend to be explicit because the system is generally unable to comprehend the meaning of text and determine relationships between common ideas or topics. Some hypertext researchers are currently addressing this issue of content analysis (Hammwöhner and Thiel 1987) and others are looking into other ways to establish relationships between nodes through the structure of node hierarchies, typed nodes and typed links automatically (Trigg 1983; Conklin 1987). Because its elements (e.g., data and models) can be thought of as entities whose names and components are recognizable expressions (i.e., nodes), a DSS lends itself to the automation of hypertext linking. An intelligent knowledge-based system can use these expressions automatically to establish direct and indirect connections among the various entities known to it. One of our contributions has been to greatly extend the domain of implicit, system-generated links. In both Max and Oona, most of the links among nodes are automatically generated by the system. We believe that a general theory of system-generated linking is possible, but discussion of that is beyond the ambitions of this paper.

Disorientation

An important issue in the design of any hypertext system is "managing the hyperspace." The representation and execution of buttons and links will affect how a user navigates through the system knowledge, application and document base. It is easy to become disoriented while navigating the myriad of links and nodes in an environment that encourages one to transfer among a plethora of dis-

tinct pieces of information (Bernstein 1987). We are exploring methods of enabling the user to maintain his or her orientation at all times (e.g., by providing a computed "Where am I?" function), but at present we rely on a system log that can be recalled and used to move to a previously-occupied node. At the very least, we hope to add a graphical view of the current system status.

Meta-Level Control

To allow for a truly flexible system with applications beyond the intentions of the shell designers, all aspects of the system should be customizable. The user should be able to create new button types and append to or replace any shell or application routine. Alternatively, the user should be able to insert a "demon" (a procedure which monitors the system, performing an action only when an activating condition takes place) (Gevarter 1987). Such changes, moreover, should be capable of being made under program control (rather than merely by editing). Our existing prototypes lack any significant form of meta-level control, as do all present hypertext systems of which we are aware.

5. OONA

Oona is a prototype DSS employing the shell, the model management, and (to a degree) the hypertext concepts discussed above. Oona is written in Prolog and runs in the VAX VMS environment, using VT240 and above (mouseless graphics) terminals. The model (and data) management system is central to the shell and includes two key elements. The first is a series of conventions for declaring models (and data) and for declaring properties of models (and data), including such information as the type of model (e.g., external FORTRAN; internal, written in the system's modeling language; interactive, requiring user inputs - a unidimensional utility function elicitation program is included) as well as other information about the model (e.g., source, reliability, location of documentation). These conventions may aptly be described as a model declaration language. The second key element to the model management system is an evaluator. This evaluator is able to match models to appropriate data sets. In the event that the model is expressed in the system's modeling language, the evaluator is able to evaluate the model and produce the result. If the model is, an external FORTRAN model, the evaluator is able to direct appropriate calls to the operating system and to report back the results of running the model. If the model is written in Prolog, but not in the system's modeling language, the evaluator is able to call the appropriate Prolog routine and to report back. Finally, the evaluator keeps track (in conjunction with other elements of the system) of model results that are already known so that requests for model outputs will not needlessly result in model execution.

Control of the system is via a high-level meta-interpreter that results in a cascading model graph architecture (Kimbrough, et al. 1986). The user interface is menu- and (primarily) command-driven. Once a model is declared and known to the system, generic commands are available for displaying the model in symbolic form and for further (recursive) exploring of the constituent symbols in the displayed model. To illustrate, three models are integrated via a multiattribute utility function. There are two performance models (SAR, performance on search and rescue, and ELT, performance on enforcement of laws and treaties), both written in FORTRAN. The model declaration language associates symbols with these routines. The third model is a cost model (CASHWHARS) and is written in the system's modeling language. At the top level, a user could see the utility model expressing the overall value of a given ship:

```
    u(ship1) =
k(1)*u(1,sar(ship1)) + k(2)*u(2,elt(ship1)) + k(3)*u(3,cost(ship1))
```

At this point, generic commands are available for exploring ship1, k(2), u(3,*), or any symbol declared in the model declaration language and known to the system. These generic commands work on *any* such symbol, and so are reused from one particular DSS to another. To emphasize the point, the generic commands effectively produce automatic linking of declared objects (and their parts) in the model base, which objects become (virtual) hypertext nodes. (For an earlier implementation of some of the user interface ideas in Oona, see Chesapeake Decision Sciences [1986]) As in any DSS, users may interactively run models and retrieve data; they may alter input data and, to a degree, the models themselves, and see the consequences of changed assumptions. In addition, with Oona users may interactively explore the logical relationships among model elements and may ask, what the source of gasoline price was and where this parameter is used in the models known to the system.

6. MAX

Max (MacIntosh KSS) is a general-purpose, reusable environment for creating and working with Knowledge-based Decision Support Systems (KSS). It is a tool to help KSS builders and users and is implemented in Prolog on the MacIntosh. Max manages the common tasks in problem-solving, model management, and user-machine interaction that are independent of any particular application, freeing the builder to focus on the application knowledge. It consists of three main modules: a user interface module, a model (and data) management module, and a control module. A specific KSS is obtained by combining Max with a specific knowledge base written in the modeling language of Max.

Model Management in Max

Model Representation

In Max, models are represented as statements in a modeling language that represent various concepts related

to a model. A conceptual model is defined as a collection of symbols and relations between symbols, or sets of symbols, on which a variety of actions can be performed. An action is a process that manipulates symbols and symbol-relation sets. A symbol is different from its name; it stands for concepts and information about it. For example, the symbol ∞ stands (among other things), for concepts such as $\lim(x->\infty, e^x/x!) = \infty$. A symbol can be a simple object like Cost or Name or a compound object like Linear Program.

Each symbol has an associated symbol structure. In Max, the basic building blocks or primitives for symbol structures are atoms, lists, and tuples. An atom is a unit: a number, alphabet, or word. A list has an arbitrary number of objects of the same type and the order is irrelevant (i.e., is treated as irrelevant by the system). A tuple has a fixed ordering of a given number of objects of arbitrary types. Thus, a symbol has one of the following structures: a) atomic, b) list of the form $[\mu]$, where μ is a symbol structure, or c) tuple of the form $(\mu_1, \mu_2, ...\mu_n$, where μ_i is a symbol structure for i = 1,... n ($\geq$1). The notation L#i is used to refer to the i^{th} element of a symbol L of structure list. Similarly T@X is the value corresponding to element X in tuple T. Note that the definition is recursive, since the μ and μ_is, in b) and c) respectively, may be non-atomic, and is sufficient in the sense that any object of any arbitrary structure as defined above can be classified as above.

Model Base Organization

The entire model base is organized as an attributed acyclic graph of definitional dependencies. There is independence between the general structure of a model and the detailed data for instances of it. For example, the various instances of a model to determine the optimal product mix would share properties of the general product mix model. Thus, models are partitioned into classes, where all instances of a class share certain common properties. The classes are themselves organized into an is_a generalization hierarchy to form a taxonomy of models that permits further inheritance of properties and also relates a model to other models. A model is definitionally dependent on its attributes and can be viewed as an aggregation of these attributes, which can themselves be non-primitive models. Such organization principles are often used in knowledge representation (Jones 1985; Brodie, Mylopoulos and Schmidt 1983) and modeling (Geoffrion 1987; Mannino, Greenberg and Hong 1988) since they encourage modularity and stepwise specification as well as aggregating models from smaller ones, and support both top-down and bottom-up design.

The model base is viewed as a graph with models as nodes, and typed arcs corresponding to the kind of relation between the participating nodes. In addition to the above three types of relations, other relations may be defined and these correspond to new arc types in the graph. An example is the relation "solves" for a pair (A_1, A_2), where A_1 is a technique to solve problems in the class A_2. This organization is the knowledge representation link to hypertext and facilitates easy and meaningful navigation in the knowledge base. For example, the set of objects reachable from an object A under a transitive relationship R is the transitive closure of A under R and can be computed recursively.

Modeling Language

We earlier emphasized the need for a symbolic representation for models. The knowledge to be represented is normally available from domain experts and users of the system. A modeling language is a vehicle to express this knowledge and, hence, must provide features to declare the information necessary to perform the functionality listed above (in **Features of a Model Management System**). Secondly, a specific DSS requires the creation of an application knowledge base by users in categories b) and c); hence, the language should be usable by domain experts and end-users who may be non-programmers. It has also been argued that the language should be computer-executable as well as suitable for communication (Geoffrion 1987; Fourer 1983; Meeraus 1983) and this essentially clinches the case for algebraic notation and a symbolic modeling language, which is better suited for user-machine communication, modification, and documentation.

In Max, this is achieved via a Model Declaration Language (MoDeL) that consists of several statements to declare various properties about models and is meant to be used primarily by model base developers. These statements include a brief textual definition of the concept, declaration of definitional dependencies, classification of the object, any mathematical equations, dimensions, assumptions, and relations to other objects. Mathematical equations are stated in an algebraic form that is close to a natural mathematical representation. The language is declarative so that order of the statements is irrelevant. It also includes special structures to represent frequently occurring data types in modeling real-world problems. For instance, in various classes of models, there is a need to represent data objects which have a variable number of values of the same type (e.g., series of cash flows in a financial model). The notation listof(X) is used to represent a variable number of occurrences of type X and corresponds to a vector having elements of type X. (A similar strategy is used in Oona, except for model hierarchies.)

Example 1(B)

We will use some of the models from Example 1(A) to illustrate the model declaration language in Max. For the sake of brevity, we will omit details of other parts and

present only one level of detail. We use the abbreviations LCC, AC, MC, and PC for Life Cycle Cost, Acquisition Cost, Maintenance Cost, and Personnel Cost respectively. Assume we have predefined models to compute present value (pv) of a cash flow, and expected value of a series of probabilistic outcomes.

pv: [amount, interest rate, time]
pv(a,r,t) = a/((1+r/100)^t).
expected value: [listof(Outcomes), listof(Probabilities)].
expected value(X,P) = Σ(i,1,length(X),X#i * P#i).

Model Declaration Language Statements

Vessel: [Vessel Characteristics, LCC, designer, ship builder]; "A vessel is a marine vehicle owned by the U.S. Coast Guard."

Vessel Characteristics: [Displacement, Speed, Length, Propulsion]

LCC: [AC, MC, PC]; "The life cycle costs for a vessel are determined by adding the present values of various cost categories over the vessel's estimated life."

LCC(AC, MC, PC) = AC + MC + PC.

LCC is_a Discounted Cost.

MC: [Base Cost (b), Inflation factor (f), No of years (N); "The maintenance cost for a vessel is...."

MC(BC,If,N,D) = Σ(i,1,N, pv(b*(1 + f/100)^i,'M1'#i,i).

MC is_a Discounted Cost.

The second category of users who create knowledge are analysts who are mainly concerned with instantiating and executing models. Hence, an important feature of the modeling language is how it deals with data declaration for model parameters. In MoDeL, the data values for various parameters may be entered directly by the user, be retrieved by a database reference, or be the output of another model. For example, a value for a parameter "interest rate" may be a number (e.g., 7.50), the output of a deductive model (e.g., interest rate = 7.0 if Condition1, 8.0 if Condition2) or the output of a mathematical model (e.g., expected value([(8.5, 0.4), (8.0, 0.3), (7.5, 0.3)])). This allows the user a high level of flexibility and encourages non-redundancy through use of logical references.

Example 1(C)

Suppose that a user wants to evaluate a certain vessel (say, Vessel#4) that is a candidatate for acquisition. The system will then determine from statements about the model Vessel the kind of information required to create an instance of it. We will illustrate a part of this process. Suppose the system is prompting the user for a value for Maintenance Cost (Figure 1). The user can specify the value in a number of ways:

i) Enter a number, e.g., 1000, or a non-numerical value, e.g., "mc."

ii) Select to specify the value at a further level of detail; the system then responds with prompts for the Base Cost, Inflation factor and the number of years over which the cost is to be computed (Figure 2).

iii) Specify the cost by reference to some other known value, e.g., 2*(Vessel1@LCC@MC).

iv) Select some other model to determine the value. For example, if it estimated that the cost is 1500 with probability 0.6 and 1000 with probability 0.4, the user may enter "expected value([1500,1000], [0.6,0.4])".

v) enter an expression which is some combination of the above, e.g., 2*Vessel1@LCC@MC + expectedvalue ([1600,Vessel3@LCC@MC],[0.6,0.4]).

At various stages the user may also obtain help or information about the models in question, e.g., the user may want to know about maintenance cost.

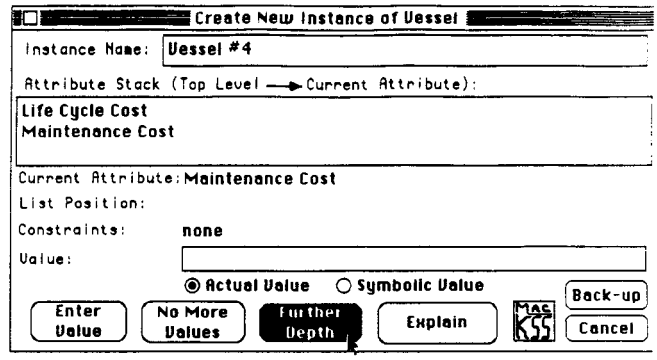

Figure 1. Further Depth Gives You Maintenance Cost's Components (Figure 2)

For existing instances, users may query about or modify data values, or they may ask for explanations for computed values. Data may also be transfered into reports or documents. For computed values, the system displays the result but maintains knowledge about how the value was computed so that an explanation may later be generated. Hence, if some of the parameter values happen to be changed after the report was created, the report will automatically show the new output. Figures 3 and 5 show

the result of a command to explain the Maintenance cost of Vessel#4 which appears in a document (Figure 4). This explanation is computed dynamically by a general rule to explain data values. The user may also ask for explanations at a deeper level of detail; e.g., how the Base Cost was computed.

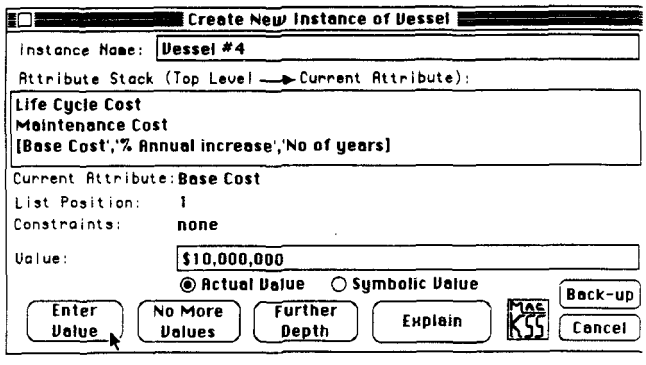

Figure 2. Entering a Value for Base Cost, a Component of Maintenance Cost

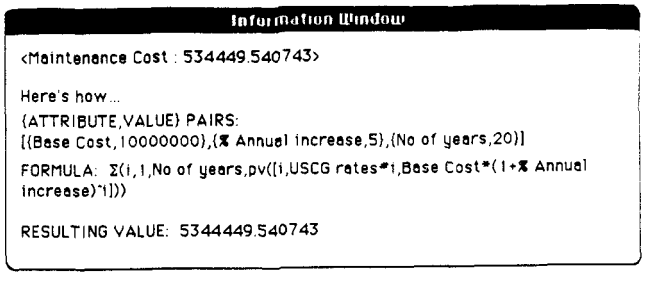

Figure 3. Explanation for Maintenance Cost

Model Manipulation Rules

The symbolic representation for models is derived from statements in the model declaration language. These statements are also treated as data and can be manipulated to perform the features described in Section 3. Actions are processes to manipulate symbolic expressions based on the representation of the model. The manipulation rules are based on model structure, rather than models themselves, and are therefore general-purpose and domain-independent. In spite of the generalizations, we need a mechanism to handle exceptions. This is done by letting the specific rules, if any, for any action, override the general rules for that action. In the current implementation, there are rules to support features (c), (d), (e), (h), and (i), and partially support features (a), (g), (j) and (k), and we illustrate some of these in Example 1(C).

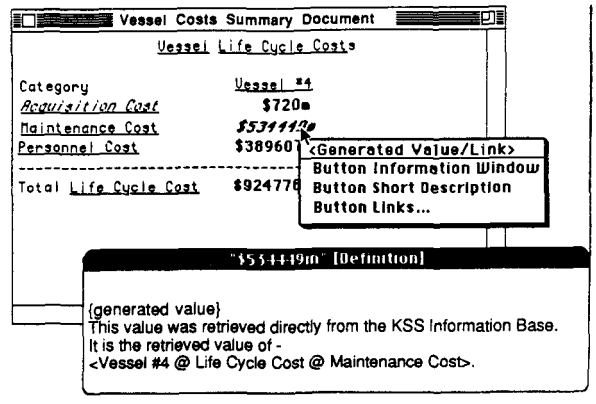

Figure 4. Selecting $534449 (the Maintenance Cost for Vessel #4) causes a "pop-up menu" to appear with valid operations for the selection. Note that key words are underlined in the document, generated values are boldface and links are italicized.

Figure 5. Choosing "Button Short Description" calls up the [Definition] window shown here.

The rule-based action processing subsystem is the inference engine of the modeling system. It captures user commands, determines and obtains a set of information to execute the command, and triggers off rules for model manipulation with this information as arguments or data. For example, if a user asks for an explanation for the number 534449, the system determines that it is the Maintenance Cost for Vessel#4 and triggers the rule explain(X), with X = Vessel#4@Life Cycle Cost@Maintenance Cost. The rules manipulate higher-order concepts (such as is_a, listof, Σ). For example, for any relation R, R(Y), X is_a Y → R(X); or if R is transitive, R(A,B), R(B,C) → R(A,C).

As in most DSSs, there are sets of rules to process various actions on models: retrieve data, transform data, query and modify, describe concepts, explain data values, analyze changes, create new class instances, etc. For example, to execute any model, the system determines the data required and obtains it from the database or from the user, transforms the data for the appropriate model, and executes the model. The rule to describe a model generates an initial description about a model (which is not a pre-defined "help-text"), and provides the user with a set of related concepts which the user may choose to pursue for further information.

Maxi: Implementation of a Hypertext User Interface

Maxi is the user interface subsystem of Max. It is bimodal in that it can both access the knowledge base directly and control a user workspace, called a document, into which text and model results may be placed. Docu-

ments are integrated workspaces combining such objects as standard text, business graphics, and spreadsheets. These objects can be stored, queried and linked just as any other system entity. The discretionary display of information (WYWWYWI) is implemented in several ways. First, Maxi will recognize and process several types of generalized hypertext *buttons*:

(1) **Keywords**: character strings registered in the knowledge base. (These include the names of all system entities: models, data, documents, links and other genralized hypertext buttons.)

(2) **Generated Values**: retrievable results from the knowledge base. (These include both data that my be looked up and model results that must be calculated.)

(3) **Link Elements**: both implicit system-generated and explicit user-specified logical connections between any two system entities

(4) **Exchanges**: document sections that can be hidden from view or replaced by an alternative document section.

(5) **User-defined**: users may specify new button types and provide instructions for how the system should react to their selection.

Max provides several meta-level functions operating on these buttons:

(1) **Location**: Maxi will determine which button(s) on the computer screen have been selected and will offer a list of options available. (See Figure 4 for an example.)

(2) **Descriptions**: Maxi can generate a description of any registered entity, telling what it is, how it was created, or by whom it was entered, and what is directly related to it. In doing so, it makes use of whatever model managment system is installed.

(3) **Explanations**: For generated values, Maxi can determine which model, function and data instance were used to create it. (See Figure 5 for an example.)

(4) **Linking**: Two link paths logging the actions of the user are maintained at all times. A user can, at any time, initiate a knowledge search (or *information detour*) following explicit user-specified or implicit system-generated links. The user can return to any point along the link paths. For example, while executing a present value model, the user could request information on interest rates. This would generate a description including related keywords. The user could then request information on any of these keywords, and so on to the limits of the declared knowledge within the system.

(5) **Adding Knowledge**:

(a) A user can permanently record comments about any system entity at any time, thus adding to the organization's knowledge base.

(b) A user can add new data instances of models (*scenarios*) and new documents at any time.

(c) The KSS builder will be able to add new models at any time.

(6) **Suspending Operations**: Whenever an operation passes control to the user (for example, while waiting for user input in a model execution), the user will be able to stop working on this operation and start any other (including halting the session). Later the link paths can be used to return to where the work left off.

Customizing the Interface

Maxi allows the user to customize most aspects of the interface. For example, one may have several different style modes for any given document. All text and business graphic buttons may be highlighted or just certain button types displayed; there may be a formal presentation mode, and a casual revision mode. One may swap between modes by specifying a mode index. In the future, the user will be able to activate and deactivate links with the same mode index, which will also be used to install one's choice of exchange button sections. Finally, the user may choose among several methods of highlighting. (See Figure 4 for some examples.)

7. CONCLUSION

Generalized hypertext and model management for DSS are well-established concepts. That they are here to stay is obvious. What forms they will take--what features they will have, how they will be implemented--is yet far from clear. The main results to date of our research and reflection on the problem of designing these DSS subsystems are reported above. Two prototype systems, Max and Oona, have been implemented for the U.S. Coast Guard. Initial reaction from users has been enthusiastic. Research and development will continue and will be the subject of future reports.

8. ACKNOWLEDGEMENTS

The work reported on in this paper was funded in part by the U.S. Coast Guard Research and Development Center, Groton, Connecticut, with Steven O. Kimbrough as Principal Investigator (contract number DTCG39-86-C-80348).

9. REFERENCES

Akscyn, R.; McCracken, D.; and Yoder, E. "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations." *Proceedings of Hypertext-87*, November 1987, pp. 1-20.

Applegate, L. M.; Konsynski, B.; and Nunamaker, J. "Model Management Systems: Design for Decision Support." *Decision Support Systems*, Vol. 2, 1986.

Bernstein, M. *The Bookmark and the Compass: Orientation Tools for Hypertext Users*. Eastgate Systems, Inc., 138 Brighton Avenue Suite 206, Boston, MA 02134, 1987.

Blanning, R. W. "An Entity-Relationship Approach to Model Management." *Decision Support Systems*, Vol. 2, 1986.

Bonczek, R. H.; Holsapple, C. W.; and Whinston, A. B. *Foundations of Decision Support Systems*, New York: Academic Press, 1982.

Bradley, S.; Hax, A.; and Magnanti, T. *Applied Mathematical Programming*. Reading, MA: Addison-Wesley, 1977.

Breitman, R. "PLANETS: A Modeling System for Business Planning." *Interfaces*, Vol. 17, No. 1, January/February 1987.

Brodie, M.; Mylopoulos, J.; and Schmidt, J. (eds.). *On Conceptual Modeling*. New York: Springer-Verlag, 1983.

Bush, V. "As We May Think." *Atlantic Monthly*, Vol, 176, July 1945, pp. 101-108.

Chesapeake Decision Sciences, Inc. *MIMI/MJ: Manager for Interactive Modeling Interfaces*. 1986.

Conklin, J. "A Survey of Hypertext." MCC Technical Report #STP-356-86, Rev. 2, 12/3/87.

Dolk, D., and Konsynsk, B. "Knowledge Representation for Model Management Systems." *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 6, November 1984.

Elam, J.; Henderson, J.; and Miller, L. "Model Management Systems: An Approach to Decision Support in Complex Organizations." *Proceedings of the International Conference on Information Systems*, 1980.

Engelbart, D. C., and English, W. K. "A Research Center for Augmenting Human Intellect." *AFIPS Conference Proceedings*, Vol. 33 Part 1, Washington, D.C., The Thompson Book Company, 1968.

Fourer, R. "Modeling Languages Versus Matrix Generators for Linear Programming." *ACM Transactions of Mathematical Software*, Vol. 9, No. 2, June 1983.

Geoffrion, A. "An Introduction to Structured Modeling." *Management Science*, Vol. 33, No. 5, May 1987.

Gevarter, W. B. "The Nature and Evaluation of Commercial Expert System Building Tools." *Computer*, May 1982, pp. 42-41.

Hammerwöhner, R., and Thiel, U. "Content Oriented Relations Between Text Units: A Structural Model for Hypertexts." *Proceedings of Hypertext-87 Conference*, November 1987, pp. 155-174.

Jarke, M., and Vassiliou, Y. "Coupling Expert Database Systems." In W. Reitman (ed.), *AI Applications for Business*, Norwood, NJ: Ablex, 1984.

Jones, C. V. *Graph-based Models*. Unpublished Ph.D. Dissertation, Cornell University, 1985.

Jones, C. V. "User Interfaces." University of Pennsylvania, Decision Sciences Department Working Paper 87-11-10, 1987.

Katz, N., and Miller, L. "A Model Management System to Support Policy Analysis." *Decision Support Systems*, Vol. 2, No. 1, March 1986.

Kimbrough, S. O. "On Shells for Decision Support Systems." University of Pennsylvania, Department of Decision Sciences Working Paper, 1986.

Kimbrough, S. O. "The Argumentation Theory of Decision Support." University of Pennsylvania, Department of Decision Sciences Working Paper, 1987.

Kimbrough, S. O.; Coe, T.; Pritchett, C.; Roehring, S.; Smith, J. A.; and Sprague, M. "A Decision Support System for Evaluation of Advanced Marine Vehicles." In Jane Fedorowics (ed.), *Transactions of DSS-86, Sixth International Conference on Decision Support Systems*, Washington, D.C., April 21-24, 1986, pp. 15-26.

Konsynski, B. R. "On the Structure of a General Model Management System." *Proceedings of the Fourteenth Hawaii International Conference on System Sciences*, 1980.

Liang, T. P. "Toward the Development of a Knowledge-Based Model Management System." Ph.D. Dissertation, University of Pennsylvania, Department of Decision Sciences Working Paper 86-11-01, 1986.

Mannino, M.; Greenberg, B.; and Hong, S. N. "Knowledge Representation for Model Libraries." In B. R. Konsynski (ed.), *Proceedings of the Twenty-First Hawaii*

International Conference on System Sciences, Vol. III, 1988, pp. 349-355.

Marchionini, G., and Shneiderman, B. "Finding Facts versus Browsing Knowledge in Hypertext Systems." *Computer*, January 1988, pp. 70-80.

Meeraus, A. "An Algebraic Approach to Modeling." *Journal of Economic Dynamics and Control*, Vol. 5, 1983.

Miller, G. A. "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capability for Processing Information." *Psychological Science*, Vol. 63, 1956, pp. 81-97.

Pollock, J. L. *Contemporary Theories of Knowledge*. Totowa, NJ: Rowman & Littlefield Publishers, 1986.

Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Reading, MA: Addison-Wesley Publishing Company, 1987.

Sprague, R. H., Jr., and Carlson, E. D. *Building Effective Decision Support Systems*. Englewood Cliffs, NJ: Prentice-Hall Inc., 1982.

Sterling, L., and Shapiro, E. *The Art of Prolog: Advanced Programming Techniques*. Cambridge, MA: The MIT Press, 1986.

Trigg, R. *A Network-based Approach to Text Handling for the Online Scientific Community*. Unpublished Ph.D. Thesis, University of Maryland, 1983.