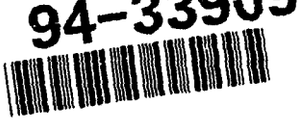
(11)

Model Formality in Human/Computer Collaboration

Brian Harp and
Bob Neches
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

March 1994
ISI/RR-94-383



REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1994	Research Report

4. TITLE AND SUBTITLE
Model Formality in Human/Computer Collaboration

5. FUNDING NUMBERS
MDA-972-90-C-0060

6. AUTHOR(S)
Brian Harp and Robert Neches
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER
RR-383

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
ARPA/SISTO
3701 N. Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Presented at the AAAI Fall Symposium series, October 1993

12A. DISTRIBUTION/AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

This paper describes a programming methodology called *model-based programming* which supports development of systems with complex activities accessing large columes of data. This methodology utilizes multiple levels of data model formality providing some powerful features for systems that are built using this methodology. For example, unlike most traditional systems, information can be programmed into an application during all phases of the system lifecycle from development through daily use. Second, the way information is entered will dictate how much reasoning the system will perform on it. This provides options for the user when they enter information; they can enter information quickly and get less system management, or they can enter information more formally and the system will take more responsibility.

One important component of this programming environment provides a way for users to enter information that was not anticipated at development time - an important capability of a system is going to grow with it's increasingly expert users. This component called TINT (The Intelligent Note Taker) is explained in detail and issues of human/computer collaboration are discussed in the context of adding unanticipated information to complex systems.

14. SUBJECT TERMS

15. NUMBER OF PAGES
6

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Model Formality in Human/Computer Collaboration

Brian Harp

USC/Information Sciences Institute
Suite 1001, 4676 Admiralty Way
Marina del Rey, CA 90292
harp@isi.edu

Robert Neches

USC/Information Sciences Institute
Suite 1001, 4676 Admiralty Way
Marina del Rey, CA 90292
neches@isi.edu

Abstract

This paper describes a programming methodology called *model-based programming* which supports development of systems with complex activities accessing large volumes of data. This methodology utilizes multiple levels of data model formality providing some powerful features for systems that are built using this methodology. For example, unlike most traditional systems, information can be programmed into an application during all phases of the system lifecycle - from development through daily use. Second, the way information is entered will dictate how much reasoning the system will perform on it. This provides options for the user when they enter information; they can enter information quickly and get less system management, or they can enter information more formally and the system will take more responsibility.

One important component of this programming environment provides a way for users to enter information that was not anticipated at development time - an important capability if a system is going to grow with it's increasingly expert users. This component called TINT (The Intelligent Note Taker) is explained in detail and issues of human/computer collaboration are discussed in the context of adding unanticipated information to complex systems.

Introduction

Information workers (e.g., designers, logistical analysts, physicians) require advanced tools and significantly more on-line support to help them achieve productivity gains in the future. The models and the data in these domains are very complex and require a wide range of representations and reasoning components. These systems will typically require human intervention and because of the complexity in the domains, require a high level of interaction between the system and users. This paper discusses some of the issues in these types of systems, describes a paradigm for addressing them, and then focuses on TINT (The Intelligent Note Taker) [3], a tool that has provided significant support for human/computer interaction in the applications we have developed. TINT reduces inflexibility of information systems by providing ways to enter unanticipated information, while giving it enough structure to operate with partial understanding and thereby assist the user in managing it.

Two of the key issues in providing better support for information workers are:

- The need to reduce the inflexibility of current information systems, by providing ways they can operate with partial understanding of information that was unanticipated during construction of their data/knowledge bases
- The need to help users navigate their way through systems when they don't know the functionality available to them, or don't understand the organization and content of the information sources in the system.

In the sections following, we will first describe a programming methodology, called *model-based programming*, that provides a substrate for addressing these issues by providing an underlying knowledge representation and high-level programming capabilities. In discussing this approach, we will introduce a notion of degrees of formality in representation that will permeate the rest of the discussion.

Next, we will briefly describe a layer of software components based on the model-driven programming paradigm that define our framework for building collaborative environments. These tools support functions in task domains involving management of large amounts of information and multiple, long-term activities [6]. TINT is a key tool in that framework. It reduces brittleness and softens demands upon users' knowledge of the system by providing a bridge between formal knowledge and informal knowledge.

After introducing our approach and describing TINT, the next two sections will discuss relative strengths and weaknesses of humans and computers in the context of the two key issues mentioned above. We will then discuss a set of issues associated with managing interactions between the user and system. Finally we will briefly discuss the knowledge required for a system to be a collaborator and the architectural requirements of a flexible information system using TINT.

Model-Driven Programming

Model-driven programming seeks to specify behavior in a fashion that reduces the burden of explicitly handling control. Our current implementation of this paradigm does so by providing a modeling system consisting of a core modeling language and satellite languages integrated

together into a central domain model [8]. The core model defines the terminology of the application while the satellite models provide tool-specific terminology for the application. An interface component is capable of presenting all of the models in the representation system so the user has a seamless view of the application.

A number of different languages are used within our model-driven environment and they vary in their level of expressiveness. It is generally believed that the more a modeling language formally represents the semantic content of an application the more the system can infer and "understand" about the data. The models produced from these kinds of languages are called *formal* models [4]. Logic based languages such as Prolog and LOOM [5] are examples of a formal modeling language.

On the other hand, some types of information in an application may be in a purely syntactic form (e.g., strings, numbers) with no domain specific representation. We call this type of model an *informal* model. There are a number of reasons why the data may be informal:

- the developer may choose not to model it formally because the extra detail is not required to meet the goals of the application
- the data may not have been known at development time making it impossible to build the model to represent the data. Thus, when the data is entered by the user it has to be captured in a semantic-free representation.

Semantic reasoning cannot be performed using only informal information because the information is not entered in a form that can be used by semantic reasoners.. However, integration of formal and informal representations is possible by storing informal information in formal objects. (E.g., an attribute of an object may contain a value of a string). If a formal model is used in an application and a user wants to assert some information that doesn't fit the model, systems typically give no assistance in handling it for the user. In applications where this is the case, users typically have to record this informal information externally in a paper system or utilize a mechanism that is independent from the formal model (e.g., a text editor to record and save text to files).

To help close the gap between formal and informal models, our modeling system contains a third representation, a *semi-formal* model. Semi-formal models [3,4] use a relatively formal representation, but there are specially designated attributes that contain informal information. Special inference mechanisms operate on the semi-formal representation which take these special attributes into account.

The structured part of the semi-formal representations serves two purposes in a modeling environment:

- it represents domain specific information that can be anticipated and modeled formally
- it provides a bridge between the informal information and formal representations of an application.

Integrated User Support Environments

We have been building on top of this model-driven programming architecture to provide a framework for creating integrated user-support environments[6]. We define an integrated user-support environment to be a set of software tools (e.g., scheduling systems, databases, browsing aids, notes, interface components, and expert systems) which the user can utilize to perform the activities of that domain. The key components of our framework are:

- HUMANOID [9], a user interface management system which facilitates evolutionary development by allowing users to readily create customized displays.
- SCENARIOS/AGENDAS [7], a language and planning system for scheduling and controlling the activities of multiple agents (humans and software processes) dealing with multiple,,extended, interleaved problem solving tasks.
- DAM [2], a data assimilation and monitoring module that regularly inspects databases to detect changes of interest to the system. Our approach is particularly concerned with retrofitting to pre-existing databases.
- BACKBORD [10] a knowledge base / database browser that helps create queries and other specifications.
- TINT [3], a system that utilizes a semi-formal model called Notes to capture and manipulate unanticipated information in applications.

TINT

Because TINT addresses some of the most important issues in human/computer interaction, it will be the primary focus of discussion in the rest of this paper. TINT, utilizes a semi-formal representation called Notes which contains generic note types and attributes and provides inference mechanisms and a user interface for managing semi-formal information. The Notes representation utilizes a persistent, frame-based representation and provides an Application Programmers Interface (API) used to link informal and formal information together.

A general model of note types is provided with the TINT system; developers can simply specialize this set of basic note types for an application. The note taxonomy contains representations for general issues. For example, problems and their solutions are represented by the classes *anomaly note* and *resolution note*. Anomalies capture observed conditions in a data source that require further investigation. Resolution notes capture the response to an anomaly note and typically describe the details of the response or how it was accomplished, thereby indicating that investigation is complete. Anomalies and resolutions are notes, because in many applications, we find that users want to make assertions about anomalies that haven't been specifically designed into the system. These two general note types are used by the data monitoring module (DAM) mentioned above.

In addition to types, notes have attributes. Some of the attributes are restricted to other formal or semi-formal objects, while some contain strings (informal data). In general, the attributes with formal restrictions serve as a context for the note so the user can relate the semantic information they read in strings with the semantic information in the formal model. For example, notes have a value slot called *referent*, where the values are semi-formal or formal objects. Referents can be used in many ways, but typically notes are created about some particular

objects in the system and these objects are placed in the referent slot.

TINT is designed to support a small group of developers/users who must work as a team in an information system environment. In this context, TINT provides many of the desirable features suggested in the seminal paper on the seven issues of Hypertext [1]. For example, TINT supports traditional hypertext functionality in addition to advanced features such as retrieval, hypertext links to other representations, and browsing. However, TINT currently does not support some functionality of extremely distributed systems like Lotus notes, World Wide Web (WWW) or Gopher such as network communications for a large network of users.

Now that we have described some of the technical aspects of TINT, we will discuss a set of issues in human/computer interaction and how TINT addresses these issues. We will show how TINT assists humans in several large application systems, and how it can be used to collect information that would not otherwise be collected if a strictly formal or informal representation were used.

Sharing Responsibility Between Users and Computers

Responsibility must be shared between humans and computers in these complex domains. If computers are thought of as just another agent working in a particular domain, then humans should be assigned tasks in which they are most competent and software should be designed to perform tasks that the computer can do well.

In the paragraphs that follow we discuss three issues that benefit from human/computer collaboration. These three issues are: Understanding and modifying formal models, retrieving information, and flexible human interfaces. We discuss how the TINT system (and other peripheral software) supports the human in performing these activities and discuss the best division of labor between human and computer.

Understanding and Modifying Formal Models

Every system must operate on a model, either an implicit model or an explicit one. Many systems today use a single, formal language (e.g., Prolog, Relational DBs) to model all of the domain data. These modeling languages provide a relatively rigid representation that models known information very well. Application developers build applications by developing the requirements of the system and then constructing the models and associated functionality [2]. These systems are built, thoroughly tested and users then start using them for their work.

As is the case in most complex domains, users experience new situations periodically, and over time they become more and more expert in the domain. One would like the system to reflect this growth as well. However, in many cases the user is not familiar enough with the model or the modeling language to make these changes to the system. Unfortunately, the developer cannot be available for the lifetime of the system either, so over time the users become more expert (and must remember more) and the system becomes obsolete. TINT provides a way to reduce this type of system obsolescence. TINT allows the user to enter information easily and quickly without requiring changes to the core model. Notes that capture new information can be created and linked to the relevant features in the core representation. These notes capture the user's knowledge and provide a way to make the system more supportive without making modifications to the core models.

EASES [11] is an application that illustrates the use of TINT in augmenting its system's knowledge. EASES assisted item managers in managing spare parts for weapon system repair in the military. In one case, item managers observed that a set of parts were failing more frequently than average. The users investigated the problem and found that heat-sensitive parts on aircraft flying in the South Pacific failed much more rapidly. They discovered that pilots fired their afterburners more while flying over the uninhabited parts of the South Pacific and this caused the heat sensitive parts to fail faster.

Since this situation was not anticipated, EASES' core knowledge base only represented some of the information in this example. Part/subpart relationships were represented along with failure rates and some other logistics attributes. However, developers could not predict such specifics as South Pacific regions and pilots' flying habits would have an impact on aircraft failure rates. Failure-conducing factors, such as heat-sensitivity , were also not part of the initial model. Using Tint, an EASES user can enter this information easily as a note and link it to the appropriate parts stored in the core knowledge base. Of course, the system would not detect this problem if it arose in the future, but the note does document the occurrence of this problem for future reference. Furthermore, the system can be programmed to ask for help when it is doing predictions and it recognizes the presence of notes that might impact its results. This type of data capture in a traditional system is difficult to perform, if it is possible at all.

Not only does this semi-formal layer allow annotation of a knowledge base, but there are some additional benefits:

- Application notes are no longer logged on paper but are entered into the computer for storage.
- The developer can analyze the notes based on the model and extend the formal model accordingly. (This analysis and model extension is currently a manual process.)
- Because the notes use a semi-formal representation, some of the capabilities of formal representations can be used with notes such as retrieval, classification and some inference.

Retrieval of Textual Information

For a human to work effectively with computers, they need to be able to ask for information and receive it in an understandable form. Retrieval capabilities are especially important because people often want to collect the details of a particular task or object. This is important for auditing purposes and recall of details for application to other tasks. Hypertext systems with only link-following

capabilities are not up to this task and text search using string matching is somewhat hit-or-miss. Using the semi-formal representation of notes, retrieval systems like BACKBORD [10] can be used to find groups of notes with particular features or values. The formal aspects of the note and the query capabilities of its representation language can be utilized to find notes of interest.

Flexible Interfaces

Typical computer interfaces today provide very little flexibility for humans to enter information that was not anticipated during development. There seem to be at least two reasons for this:

- systems are built on relatively static models of the domain so relating random information to the model in a systematic way is difficult and
- it requires significantly more work for a developer to add input capabilities for unanticipated information, so it is usually not added.

The TINT Notes representation has been designed specifically to alleviate the first problem: adding unanticipated information to static models. Users can create note instances through TINT and link them to objects in the existing knowledge bases. TINT currently supports links from notes to notes, notes to LOOM[5] knowledge base objects, notes to Itasca Object Oriented DB objects, and notes to interface objects modeled in the KR language[9].

Our methodology for the second problem is based on the design philosophy of the particular interface system we are using, HUMANOID[9]. HUMANOID provides a development paradigm where a developer starts with a set of default templates and specializes them for the particular application windows. HUMANOID also provides a wide range of capabilities that can be "mixed in" to the current application interface. The current development of TINT will add general note-taking "mixins" for HUMANOID so the developer can add the note-taking facility to a new template. Using this scheme, we can mix in note-taking facilities in virtually all the windows so that users can enter information anywhere they wish. There are currently some issues that must be resolved in the current implementation of this interface. For example, when creating a note about a particular area on the screen it is not always obvious which object or objects are to be the referents of the note. Our current solution assumes the note is to reference an intermediate form in the window.

Assumptions about Communication and Coordination

The division of labor for an application can be partitioned into three categories; computer, human and mixed initiative. By "labor" we mean inference that extends or modifies the domain knowledge. (Processing done by the interface to present information or to parse it into an internal representation doesn't count.)

Mixed initiative is the most interesting because it involves the computer AND the human, so we will focus on it in this section. An example of a mixed initiative activity is data monitoring (DAM) in logistics database applications like EASES and DRAMA [2]. When developers add data monitoring to an application, they first define data patterns relative to the core knowledge base. These patterns describe conditions in the data that should be flagged every time data fitting that pattern occurs. When an occurrence is detected, an anomaly note is created and asserted into the note database. When the anomaly note is asserted, the note is passed to our framework's Scenarios/Agendas mechanism. Scenarios/agendas contains knowledge about activities: which activities can resolve each anomaly, how to divide up the tasks of the activity between the user and the system, and how to present the user tasks. Because it knows which activities can resolve each anomaly when the anomaly note is sent to it, Scenarios/agendas decides on the appropriate activities and initiates them. If it doesn't know of any activities that match that anomaly, it uses a default activity which presents the anomaly to the user and reminds them to do something about it.

The data monitoring aid only creates anomaly notes for patterns that were programmed by the developer. Users, on the other hand, can also create anomaly notes that capture problems in the data source and Scenarios/agendas will manage these anomalies also. The Scenario/agendas mechanism doesn't require that it understand the problem represented in the anomaly note. It simply knows the structure of the anomaly note so it can handle problems that are defined by either the system or the user.

This is a mixed initiative task from a number of aspects. One is that the user or the system is allowed to enter an anomaly. Another is that once the activity has been instantiated, the user has a chance to enter information and perform tasks that resolve the problem. If there are activities that a system can do without assistance from the user, it goes ahead and does them.

Supporting Ways that People Prefer to Work

We feel the functionality of TINT is important to have in information systems because of observations that have been made about how people work. One characteristic we have observed is that if a system is so rigid that it completely controls the dialog (e.g., a line-oriented interface where the system asks users questions), naive users feel intimidated and aggressive or creative users feel stifled. Instead, users prefer systems with the level of flexibility similar to a paper system, where they can write things where they want, rearrange documents, and reorganize information. TINT provides this capability by allowing users to make notes about arbitrary objects at anytime; the system will store and link them appropriately.

Many activities require multiple sessions before they can be completed. This requires the relevant data and information be carried along with the activity until completion. External observations as well as internal ones should be maintained by the computer so that the fullest possible context is maintained. Users sometimes have trouble remembering all the details of a problem when they have been away from the problem for a period of time and they have been very busy. TINT supports users by

allowing them to make notes about the activities they perform and attaching the notes to related objects in the system for context.

Methods for Shifting Responsibility between Humans and Computers

Now that we have discussed the strengths and weaknesses of computer and humans and some of the communication issues between them, we will briefly discuss some methods for shifting responsibility between them.

Three major roles have been identified in information systems: developer/maintainer, user and computer. These are obvious distinctions, but are interesting when considering responsibility and communication. One way to analyze these roles is based on the type of information that each party must understand. Developers must understand almost everything about the application: all semantic information, the languages (syntactic information), and all of the actions and functional components. End users understand all of the semantic information of the domain and relatively little (if any) of the syntactic information. Finally computers understand only what they are told; they understand what they are told about the domain knowledge (semantic information) and all of the syntactic information (since language syntax exists for the computer).

Based on these observations, users will be best supported if they can communicate with the system in terms of the formal model. It provides the model closest to the user's model and performs inference and reasoning at the level closest to that of the user. However, as we previously stated, there will always be information that is not in the formal model. This information has to be managed more by the user and less by the system. For instance, if some information is entered as informal data, the system may only be able to store the information and the user will have to understand it (with respect to the domain) and manage it in the system.

Thus, the way the end user enters information into the system will determine the response and the level of support of the system. If the user wants the system to help them model the information and manage it more, they will have to make sure the information gets into the formal model. If they don't need as much reasoning for some of the data, they can enter that as notes and the system will maintain the notes. In that case, much of the semantic content of the note will only be understood by the user.

The manual analysis of notes created in a system provides a way of moving responsibility from the user to the system. As was mentioned earlier, end users will enter notes into a system as they use it. Developers are able to analyze those notes to understand how the system should be extended. They can then extend the domain model using the formal representation, and can add activities for that data so that the system takes action when new information is entered.

Managing Interactions

Now that we have discussed reasoning and representation issues, we will briefly discuss the interface of TINT with respect to some well known interface paradigms.

Models of Interaction

We will discuss two interaction methods: intermediary vs. model world. The intermediary method of interaction typically presents information in a textual form as opposed to iconic or graphic form. To enter information, users typically type information in from the keyboard. The model world paradigm is basically a graphically presented interface where a user interacts with the computer via direct manipulation techniques.

We believe there should be a mixture of both types of interaction. In the TINT interface, we use the model world paradigm in attaching notes to other objects. However, we use primarily the intermediary paradigm to present and edit a note because users think of notes as textual objects. We also use the intermediary method for some activities because it provides much more information to the user about what can be performed in the system than does a model world.

Whether intermediary or model world paradigm is used, we believe the interface should be based on an explicit model; we call this a model-driven interface. We feel this is the best type of interface (especially for development environments) no matter which type of interaction scheme is chosen. One reason a model is important is that changes to a potentially large group of application windows can be implemented via a single change to the interface model. Another reason is that a developer or end user can talk about the interface structures formally. For instance, notes discussing interface issues can be attached to specific windows (and consequently their underlying objects) because a model exists that represents what is being presented on the screen.

Achieving Natural Communication

The reason we strive to design flexible systems is that they will support more natural communication. There are three criteria that must exist in a system if it is going to support natural communication:

- The formal, semantic models must be used to present information, and those models must be available to the user to annotate.
- Semi-formal representations must be available to augment the formal knowledge that is shared by the user and the system.
- The interface must support formal and semi-formal presentation and manipulation.

Having these three features present in a system means that a user can think in terms of the formal model, add information to either the formal or semi-formal model via the interface, and have query capabilities available for either formal or semi-formal information.

For example, in the EASES domain discussed earlier, the user may examine the features of a specific weapon system component. The part's information is retrieved

from the core knowledge base and presented to the user who can make a note about one of the attributes of that part. Later, when this component is presented again, the note will show up in that presentation because it is linked to the part. Depending on the number of notes and their relationship to other objects, the presentation may be in the same window as the component attributes or in a different window.

Clarifying Assumptions

What knowledge must a system have to be a collaborator? A model of the domain (either distributed or centrally located) shared by both the user and the system is a key feature of a collaborating system. The formal model represents terminology that is semantically understood by both the system and the user. The system can retrieve and present domain data based on the formal model and the user can enter information that can be added to the formal model.

A semi-formal representation is necessary if a system is to be a collaborator because the formal domain models never represents everything that a user wants to store in the system databases. The semi-formal representation must be linked to the formal models so the context of the note is captured.

What are the appropriate architectures for deploying this knowledge? The global architecture that we are promoting (and TINT especially within it) supports a situated perspective as opposed to a planned perspective. By this we mean many of the applications will involve working with information that was not anticipated at design time.

Our programming paradigm involves high-level tools (e.g., TINT, DAM) which are combined by the developer to compose a system. Satellite knowledge bases support the tools and are integrated with the core knowledge base producing a model for an application that contains formal as well as semi-formal representations. The range of representations allows a user to define a flexible system that accepts a wider range of information and the system assists in managing it with the user.

The interface component is separated from the representation and reasoning systems, and is the primary integration component for an application. The interface makes multiple representations transparent to the user so they can work at the level that is natural for the application, not at the representation level. HUMANOID supports an integrated capabilities approach where a developer can add "mixins" to windows in the interface to get the best capabilities for that part of the application. For example, this allows a developer to mix in note-taking capabilities in all the windows so that users can enter notes about information in the window with a standard note interface.

Conclusion

We believe the model-based programming paradigm will support development of more flexible systems, leading to more natural human/computer interaction. We have described a range of models that support capture and retrieval of very different kinds of information. We have shown how TINT fits into this model-based approach by introducing a semi-formal model that bridges between formal and informal representations. We discussed a number of ways that TINT reduces the brittleness of information systems and reduces demands on users' knowledge of the system. For example, users acquiring information about heat sensitive parts on a particular aircraft can enter that information even though the concept of heat sensitive part does not exist in the current, formal representation of the system. Reducing brittleness and increasing flexibility provides more natural communication between system and user, and TINT provides key capabilities in making this a reality.

References

[1] F. Halasz, Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems. Communications of the ACM, 1988, 31(7): 836-852.

[2] B. Harp, P. Aberg, D. Benjamin, R. Neches, P. Szekely. *DRAMA: an application of a Logistics Shell.* In Proceedings of the Annual Conference on AI and Logistics, Williamsburg, VA, March 1992. pp. 146-151.

[3] B. Harp and R. Neches. *NOTECARDS: An Everyday Tool for Aiding in Complex Tasks.* ISI Research Report RS-88-204, March 1988.

[4] H. Kaindl and M. Snaprud. *Hypertext and Structured Object Representation: A Unifying View*, In Proceedings of Hypertext'91, November 1991.

[5] R. MacGregor and M. Burstein. *Using a Description Classifier to Enhance Knowledge Representation.* IEEE Expert, June 1991, pp. 41-46.

[6] R. Neches, et. al., *The Integrated User-Support Environment (IN-USE) Group at USC/ISI*, In Proceedings of InterCHI'93, March 1993.

[7] R. Neches, D. Benjamin, J. Granacki, B. Harp, and P. Szekely. *Scenarios/Agendas: A Reuseable, Customizable Approach to User-System Collaboration in Complex Activities.* ISI Working Paper, 1991.

[8] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. *Enabling Technology for Knowledge Sharing*, AI Magazine, Fall 1991, pp. 38-56.

[9] P. Szekely, P. Luo, and R. Neches. *Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design.* In Proceedings of CHI'92, May 1992, pp. 507-515.

[10] J. Yen, R. Neches, M. DeBellis, P. Szekely, and P. Aberg. *Backbord: An Implementation of Specification by Reformulation.* In J.S. Sullivan and S.W. Tyler (Eds.), Intelligent User Interfaces, ACM Press, 1991, pp 421-444.

[11] R. Ohlander, L. Friedman, R. Gurfield, R. Neches. *Intelligent Logistics Support Tools*, In Proceedings of Symposium on AI Applications for Military Logistics, March 1987.