

2

NAVAL POSTGRADUATE SCHOOL Monterey, California

AD-A273 207



S DTIC
ELECTE
DEC 01 1993
A

THESIS

An Implementation of the REpresentation and MAintenance of
Process Knowledge(REMAP) model in the Knowledge-Based
Software Assistant Concept Demonstration System

by

Frank J. Hughes
and
Steven C. Kendall
September, 1993

Principal Advisor:

Balasubramaniam Ramesh

Approved for public release; distribution is unlimited.

93-29339



82 935

93 11 30 043

REPORT DOCUMENTATION PAGE				
1a Report Security Classification: Unclassified		1b Restrictive Markings		
2a Security Classification Authority		3 Distribution/Availability of Report		
2b Declassification/Downgrading Schedule		Approved for public release; distribution is unlimited.		
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)		
6a Name of Performing Organization Naval Postgraduate School	6b Office Symbol (if applicable) 37	7a Name of Monitoring Organization Naval Postgraduate School		
6c Address (city, state, and ZIP code) Monterey CA 93943-5000		7b Address (city, state, and ZIP code) Monterey CA 93943-5000		
8a Name of Funding/Sponsoring Organization	6b Office Symbol (if applicable)	9 Procurement Instrument Identification Number		
Address (city, state, and ZIP code)		10 Source of Funding Numbers		
		Program Element No	Project No	Task No
		Work Unit Accession No		
11 Title (include security classification) UNCLASSIFIED-AN IMPLEMENTATION OF THE REPRESENTATION AND MAINTENANCE OF PROCESS KNOWLEDGE(REMAP) MODEL IN THE KNOWLEDGE-BASED SOFTWARE ASSISTANT CONCEPT DEMONSTRATION SYSTEM-UNCLASSIFIED				
12 Personal Author(s) Lieutenant Commander Frank J. Hughes, United States Navy, and Captain Steven C. Kendall, United States Marine Corps				
13a Type of Report Master's Thesis	13b Time Covered From To	14 Date of Report (year, month, day) 1993, September, 23	15 Page Count 82	
16 Supplementary Notation The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
17 Cosati Codes		18 Subject Terms (continue on reverse if necessary and identify by block number)		
Field	Group	Subgroup	REMAP, KBSA Concept Demonstration, REFINE, Andersen Consulting, IBIS, Requirements Engineering, Deliberations, Knowledge-based Software Engineering.	
19 Abstract (continue on reverse if necessary and identify by block number) The Representation and Maintenance of Process Knowledge(REMAP) model supports the various stakeholders involved in software design during development and maintenance by capturing the rationale behind design decisions. This process knowledge is invaluable with changing requirements and assumptions. In the context of formal software development, process knowledge about the development of formal specifications from informal requirements will facilitate the understanding and maintenance of such specifications. We have implemented the REMAP model in the United States Air Force Rome Laboratory's KBSA Concept Demonstration system (a formal software development environment) to capture this process knowledge. We provide a graphical browser to facilitate the instantiation, browsing and modification of REMAP model primitives and a mechanism to reason with the knowledge in the Concept Demonstration system.				
20 Distribution/Availability of Abstract __ unclassified/unlimited __ same as report __ DTIC users		21 Abstract Security Classification Unclassified		
22a Name of Responsible Individual Professor Balasubramaniam Ramesh.		22b Telephone (include Area Code) (408)656-2439	22c Office Symbol AS/RA	

Approved for public release; distribution is unlimited.

An Implementation of REpresentation and MAintenance
of Process Knowledge(REMAP) Model
in the Knowledge-Based Software Assistant
Concept Demonstration System

by

Frank J. Hughes
Lieutenant Commander, United States Navy
B.S., United States Naval Academy

Steven C. Kendall
Captain, United States Marine Corps
B.S., Iowa State University

Submitted in partial fulfillment
of the requirements for the degree of

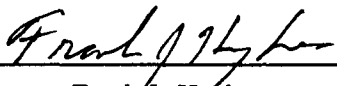
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

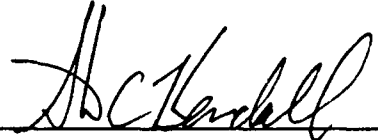
NAVAL POSTGRADUATE SCHOOL

September 1993

Authors:




Frank J. Hughes




Steven C. Kendall

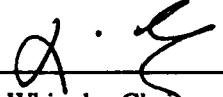
Approved by:



Professor B. Ramesh, Thesis Advisor



Professor Roger Stemp, CO-Advisor



Professor Whipple, Chairman
Department of Administrative Sciences

ABSTRACT

The REpresentation and MAintenance of Process knowledge(REMAP) model supports the various stakeholders involved in software design during development and maintenance by capturing the rationale behind design decisions. This process knowledge is invaluable with changing requirements and assumptions. In the context of formal software development, process knowledge about the development of formal specifications from informal requirements will facilitate the understanding and maintenance of such specifications. We have implemented the REMAP model in the United States Air Force Rome Laboratory's KBSA Concept Demonstration system (a formal software development environment) to capture this process knowledge. We provide a graphical browser to facilitate the instantiation, browsing and modification of REMAP model primitives and a mechanism to reason with the knowledge in the Concept Demonstration system.

DTIC QUALITY INSPECTED 5

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I. INTRODUCTION 1

 A. GENERAL 1

 B. BACKGROUND 1

 C. THESIS OBJECTIVE 3

 D. RESEARCH QUESTIONS 4

 E. SCOPE 4

 F. DEFINITIONS AND ABBREVIATIONS 5

 G. ORGANIZATION OF STUDY 5

II. KBSA CONCEPT DEMONSTRATION 6

 A. GENERAL 6

 B. HISTORY OF KBSA DEVELOPMENT 7

 1. KESTREL INSTITUTE 7

 2. SANDERS ASSOCIATES 9

 3. HONEYWELL SYSTEMS 9

 4. UNIVERSITY OF SOUTHERN CALIFORNIA INFORMATION
 SCIENCES INSTITUTE (ISI) 9

 5. SOFTWARE OPTIONS 10

 6. ANDERSEN CONSULTING 10

 C. KNOWLEDGE-BASED SOFTWARE ENGINEERING USING
 REFINE 11

 D. CONCEPT DEMONSTRATION 12

III.	REMAP	13
A.	GENERAL	13
B.	IBIS MODEL	13
C.	REMAP MODEL	15
IV.	IMPLEMENTATION	16
A.	GENERAL	16
B.	REMAP/CD SYSTEM DESCRIPTION	17
1.	HARDWARE	17
2.	SOFTWARE	17
C.	IMPLEMENTATION TUTORIAL	17
1.	BEGINNING A NEW SESSION	17
2.	LINK CERTAINTY	19
3.	NODE CERTAINTY	19
4.	ADDITIONAL MENU OPTIONS	20
5.	CAPTURING DELIBERATIONS DATA	20
D.	IMPLEMENTATION	20
1.	APPROACH	20
2.	DEFINING VARIABLES	22
3.	REMAP DISPLAY	24
4.	ADDING ATTRIBUTES TO ATTRIBUTES (FACETS)	28
5.	INTEGRATION	31
VI.	COMMENTS AND FUTURE INITIATIVES	34
A.	COMMENTS	34
B.	FUTURE INITIATIVES	34

1.	EASE OF USE	34
a.	INPUT	35
b.	EASE OF UNDERSTANDING	35
c.	EASE OF INFORMATION RETRIEVAL	36
2.	SECURITY	37
APPENDIX A	38
A.	NPS-KB-MODULE-LIBRARY.RE	38
APPENDIX B	40
A.	SAMPLE-LOAD.LISP	40
APPENDIX C	42
A.	HYPERTEXT-PATCH.RE	42
APPENDIX D	46
A.	GRAPH-SN-MODEL-PATCH.RE	46
APPENDIX E	53
A.	HLEC-PATCH.RE	53
APPENDIX F	55
A.	VIEWABLE-SLOTS-PATCH.RE	55
APPENDIX G	63
A.	FACETS.RE	63

APPENDIX H	66
A. GRAPH-IBIS-MODEL-PATCH.RE	66
LIST OF REFERENCES	71
DISTRIBUTION LIST	73

ACKNOWLEDGEMENT

We wish to acknowledge the tremendous efforts put forth by Mr. Mike Debellis of Andersen Consulting's Center for Strategic Technology Research. His assistance and technical expertise proved invaluable in the success of our implementation.

I. INTRODUCTION

A. GENERAL

Software expenditures have expanded exponentially over the previous decade. Software maintenance has accounted for the majority of these costs. There is growing recognition that, an important way of curtailing the high cost of system maintenance is to capture the rationale used to create requirements and designs, and maintain this information throughout the software life-cycle[Ref. 1]. This thesis implements a model for representing design rationale in the context of software development based on formal specifications.

B. BACKGROUND

The Department of Defense (DoD) spends nearly \$10 billion per year on software. Numerous Studies have estimated that maintenance costs can be 70 to 80 percent of the total life-cycle costs. With declining defense dollars, it is important to develop systems that are easier to maintain and implemented economically.[Ref. 2]

Recent research has recognized that capturing and using rationale for the requirements and designs throughout the

development life-cycle will help alleviate the high cost of maintenance.[Ref. 3] Requirements and design rationale can be used in the later stages of the system life-cycle to effect changes and to facilitate reuse of components. This data can also be used in the DoD environment as a component of requirements traceability. Design rationale is often the outcome of deliberations by members of the design team. The focus of capturing design deliberations is to associate the decision making process to a particular specification or design component. This information will create a historical database where the user can identify how the requirements have evolved and provide the knowledge necessary to identify repercussions of changing requirements on the design.

Previous research has produced several models for capturing design rationale knowledge. A model, called Representation and Maintenance of Process knowledge (REMAP), "recognizes the importance of capturing process knowledge to reason about the consequences of changing conditions and requirements in system design and maintenance." [Ref. 4] This model includes IBIS, a model that has been widely used for representing deliberations. Our research aims at developing an implementation of REMAP in an effort to capture valuable design rationale knowledge.

Within DoD, the Air Force's Rome Laboratory (RL) is investigating the creation of a Knowledge-Based Software Assistant (KBSA) through research conducted in both academia and the corporate world. On behalf of RL, Andersen Consulting has synthesized the results of over a decade of research in this area, into a product called KBSA Concept Demonstration system. This tool uses formal specifications to create software products. A major concern with the use of formal specification based software development is the difficulty users face in arriving at specifications. Further, with this paradigm, the process of maintenance is done at the level of specifications. Therefore, it is essential to capture the essence of the process of arriving at formal specifications, so that they can be easily understood and maintained. This process of defining formal specifications from the initial set of informal requirements can be thought of as a deliberation. In this thesis, the REMAP model will be used to represent the deliberations.

C. THESIS OBJECTIVE

This thesis examines and implements an extension of the DoD-sponsored, KBSA Concept Demonstration system using the REMAP deliberations capture model.

D. RESEARCH QUESTIONS

What is the importance of capturing requirements design deliberations, in the context of software development based on formal specifications?

How is the REMAP model implemented into the Concept Demonstration system for Knowledge-Based Software Engineering?

E. SCOPE

The scope of this thesis is limited to a brief overview of design rationale capture during requirements engineering, and software development based on formal specifications. A more detailed description of two specific examples, REMAP and Concept Demo, will be provided. This thesis will concentrate on the process of designing and implementing the REMAP model by extending Concept Demo. A graphical browser instantiates REMAP nodes and links consistent with the Concept Demonstration format to capture the design rationale into the knowledge-base.

F. DEFINITIONS AND ABBREVIATIONS

KBSA	Knowledge-Based Software Assistant
REMAP	Representation and Maintenance of Process knowledge
IBIS	Issue Based Information Systems method
Concept Demo	KBSA Concept Demonstration System
REMAP/CD	Thesis implementation of a Concept Demo extension including REMAP

G. ORGANIZATION OF STUDY

Beyond this introduction the thesis consists of four major chapters. Chapter II contains an introduction to the KBSA Concept Demonstration system. A brief history and current status will explain what KBSA is intended to accomplish and where this software development paradigm based on formal specifications fits in the Department of Defense (DoD) information technology structure. Chapter III discusses design rationale deliberation capture and examines in detail the various elements of the REMAP model used to facilitate capture in this project. Chapter IV focuses on the specific steps necessary to develop, integrate, and use the REMAP extension implemented in Concept Demo. Finally, Chapter V details conclusions and recommendations.

II. KBSA CONCEPT DEMONSTRATION

A. GENERAL

In 1982, U.S. Air Force's Rome Air Development Center (RADC), currently designated Rome Laboratory, began envisioning a new approach to solving "the software problem". In 1983, RADC published "Report on a Knowledge-Based Software Assistant" [Ref. 5] delineating possible artificial intelligence (AI) uses in integrating a systems approach to the complete software life-cycle. The Knowledge-Based Software Assistant (KBSA) offered a potential solution to issues relating to software productivity, quality and life cycle costs. [Ref. 6]

The Knowledge-Based Software Assistant is a formally-structured knowledge-based, software design, development, and maintenance tool that encompasses the entire software life-cycle. This new paradigm combines formalization with automation to achieve four distinguishing features. These are:

- incremental, formal, and executable specifications;
- formal implementation where verification and validation arise from the implementation development process;
- enforced project management policy maintaining consistent relationships between various software objects; and,
- high level system development and maintenance accomplished at the requirements and specification levels.

Implementation of these novel concepts and features would occur in an evolutionary, three-phase contractual approach.

Phase I involved dividing the concept into modules or "facets". Individual organizations concentrated on specific facets (e.g., Project Management Assistant, Requirements Assistant) while maintaining a consistent formalism for later integration.

Phase II began combination of some appropriate facets (e.g., Requirements/Specification Assistant) and the creation of a demonstration system (Concept Demo) to assist in implementation and acceptance of this new paradigm. [Ref. 7]

Phase III will involve the Advanced Development Model (ADM) which further integrates the preceding facets to provide a more usable product. [Ref. 8]

This chapter will provide a history of Phase I and II. Knowledge-Based System Engineering (KBSE) using the REFINE programming language will be discussed in the context of KBSA, and finally, an expanded discussion of Concept Demo will provide a system overview and the rationale for choosing it to implement REMAP.

B. HISTORY OF KBSA DEVELOPMENT

The following is a brief history of KBSA development, presented by research organization in roughly the order in which each began work on a facet.

1. KESTREL INSTITUTE

The Kestrel Institute began designing and constructing a Project Management Assistant (PMA) prototype in 1984. [Ref. 9][Ref. 10] Kestrel incorporated three types of capabilities into the design. (1) Project definition, (2) project monitoring and (3) an effective user

interface allow PMA to provide knowledge-based support through project communication, coordination, and task management. [Ref. 11] PMA is a step beyond traditional project management tools. In addition to typical utilities found in these tools, (e.g., Pert and Gantt Charts, test cases and results, milestones) PMA understands implicit relationships between products, and, using powerful temporal relationships, provides a mechanism for formally expressing and enforcing project policies. The initial PMA was completed in 1986, an expanded version was completed in 1990. [Ref. 12]

Development of Performance Assistant (PA) began in 1985. PA was designed to assist with performance optimization at many levels in the software development cycle. Kestrel Institute concentrated on two primary areas to permit high-level programming in a wide variety of languages to produce efficient code within the bounds of the KBSA paradigm. (1) Control optimization established efficient and formal transformations effecting specification to code conversion, and (2) data optimization, including data structure selection, created efficient object types for copying, storage, retrieval, etc. [Ref. 13][Ref. 14]

The Development Assistant (DA) effort began in 1988. Sharing and expanding many capabilities from Performance Assistant, the DA uses PA's efficient optimization techniques and integrated these more closely into the KBSA paradigm. This key facet supports the construction of the application domain to include system specifications. Formal specification transformations provide detailed code. High-level decisions can be quickly and systematically implemented and considered. [Ref. 15]

2. SANDERS ASSOCIATES

In 1985, Sanders Associates began work on the Knowledge-Based Requirements Assistant (KBRA). KBRA provided the important functionality of accepting informal requirement definitions with incompleteness and inconsistency, and building and maintaining these in a consistent internal representation. This representation could be manipulated to provide multiple views, formats and tools for an iterative requirement definition phase. [Ref. 16]

3. HONEYWELL SYSTEMS

In 1986, Honeywell Systems and Research Center began work on KBSA Framework. Framework took a full scale approach with two goals in mind, (1) integrating a KBSA demonstration and (2) building upon the demonstration to logically and consistently interconnect the various KBSA facets. Efforts in this area include (1) determining minimum functionality to combine all facets, (2) developing a common interface for facets to mesh, and for users to interact with the system, (3) support for programming-in-the-large concepts, and (4) providing functionality for a distributed environment. The Honeywell effort provided a common language, the Common LISP Object System (CLOS), The KBSA User Interface Environment (KUIE), and a preliminary Configuration and Change Management (CCM) model for KBSA.

[Ref. 17][Ref. 18]

4. UNIVERSITY OF SOUTHERN CALIFORNIA INFORMATION SCIENCES INSTITUTE (ISI)

ISI began its effort to develop a KBSA Specification Assistant in 1985. ISI's goal was to facilitate the development of formal executable specifications. They use two methods, (1) a top down method in which the system specification is incrementally elaborated, and (2) an

evolutionary method that iteratively changes or transforms the specification as development of the system continues. In 1988, ISI and Lockheed Sanders began merging the Sanders Knowledge-Based Requirements Assistant and the ISI Specification Assistant into a system called ARIES. [Ref. 19]

5. SOFTWARE OPTIONS

Transaction Graphs were developed, in 1988 by Software Options, to formally define a graphical syntax that could be used to specify and enforce the coordination of the many KBSA activities that would be used consistently throughout the system. [Ref. 20]

6. ANDERSEN CONSULTING

The creation of a complete demonstration system involving concepts of KBSA commenced in 1988 and was termed KBSA Concept Demonstration (Concept Demo). The Concept Demo is a system that combines the technologies previously described. This Refine-based demonstration system will be described in detail in the following section.

The figure on the following page provides the KBSA Research Program Overview.

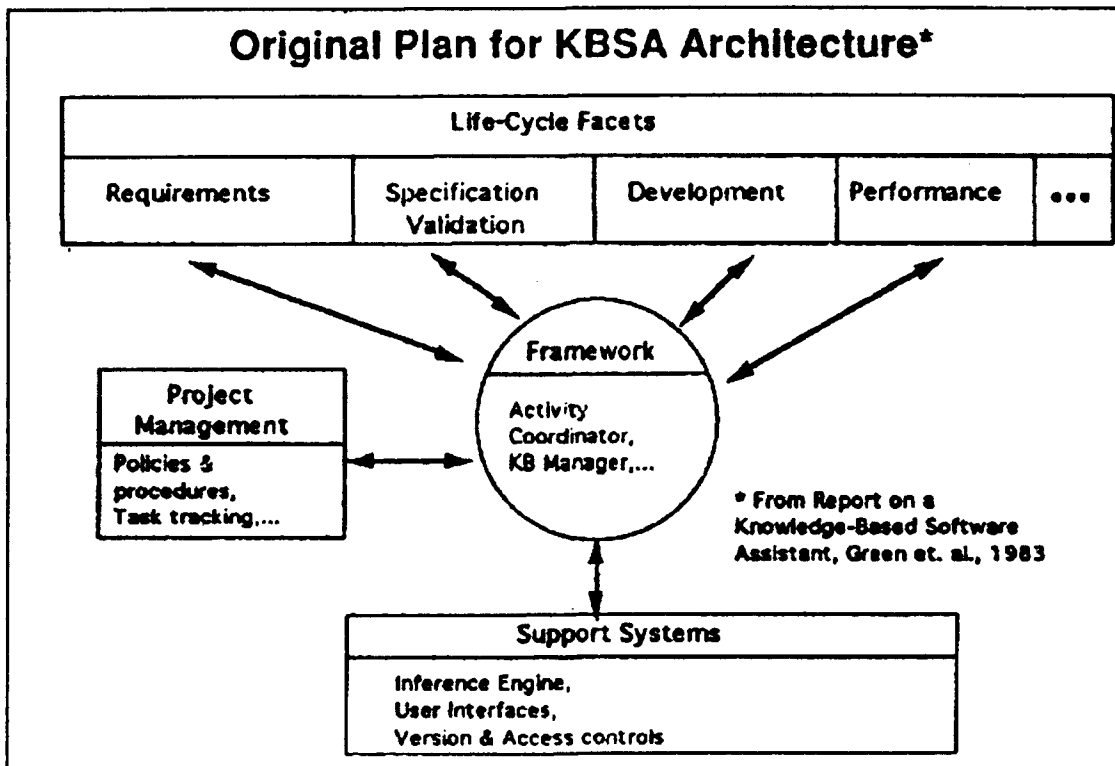


Figure 1

C. KNOWLEDGE-BASED SOFTWARE ENGINEERING USING REFINE¹

Knowledge-based software engineering is a software development approach that analyzes program characteristics and application domain so that information gained can be formalized and added to a knowledge-based compiler, thus permitting knowledge to be automated. This approach is valuable because it fosters automatic reuse of knowledge. [Ref. 21]

One of the first knowledge-based commercial software development environments, and the first to provide powerful transformational programming features, is Reasoning Systems' REFINETM. REFINETM's principal features are (1) the REFINETM Specification Language, (2) the REFINETM Specification

¹ REFINE is a registered trademark and codemark of Reasoning Systems, Inc., Palo Alto, CA.

Compiler, and (3) the REFINE™ knowledge base management system. [Ref. 22] These features combined with REFINE™'s extendibility and flexibility provided an ideally compatible environment for incorporation of the KBSA paradigm into a concept demonstration system.

D. CONCEPT DEMONSTRATION

Andersen Consulting created Concept Demo with an important goal of communicating the KBSA approach to software development. Concept Demo displayed the full range of KBSA functionality, from capturing informal requirements to generation of efficient code, with emphasis on those functions that differentiate KBSA from conventional CASE tools. Using a REFINE™ software development environment, Andersen Consulting integrated facets from previous KBSA projects where feasible, and re-implemented equivalent functionality when necessary to provide consistency throughout the demonstration system. [Ref. 23] KBSA Concept Demo provided an extendible, formalized development system capable of generating and maintaining efficient, low-level code created through a transformation process initiated at the requirements definition phase.

VI. REMAP

A. GENERAL

Chapter II described the KBSA Concept Demonstration System's functionality during the system life-cycle from informal design requirements capture to generation and maintenance of efficient code. Research has shown the value of capturing not only design requirements, but also, the design history, intermediate artifacts, and the deliberation process relating to the development of design solution.[Ref. 24] Design information reuse for similar systems, and system maintenance are facilitated by capture and use of this process knowledge.

The Representation and Maintenance of Process knowledge(REMAP) model was used, in the context of Concept Demo, for providing this functionality. REMAP was developed using an empirical study on problem-solving behavior of individuals and groups engaged in systems development. The model represents deliberations that could occur in any systems development entity. The REMAP specifically investigates the process knowledge attributed to the objects created during the requirements engineering process.[Ref. 25] The REMAP model is based on and expands the Issue Based Information System (IBIS) method as shown in Figure 2.

B. IBIS MODEL

The IBIS is a method to represent argumentation processes.[Ref. 26]

The IBIS model is comprised of three primitives or nodes. These are **ISSUE**, **POSITION** and **ARGUMENT**. These primitives are connected by relationship links as shown in Figure 2.

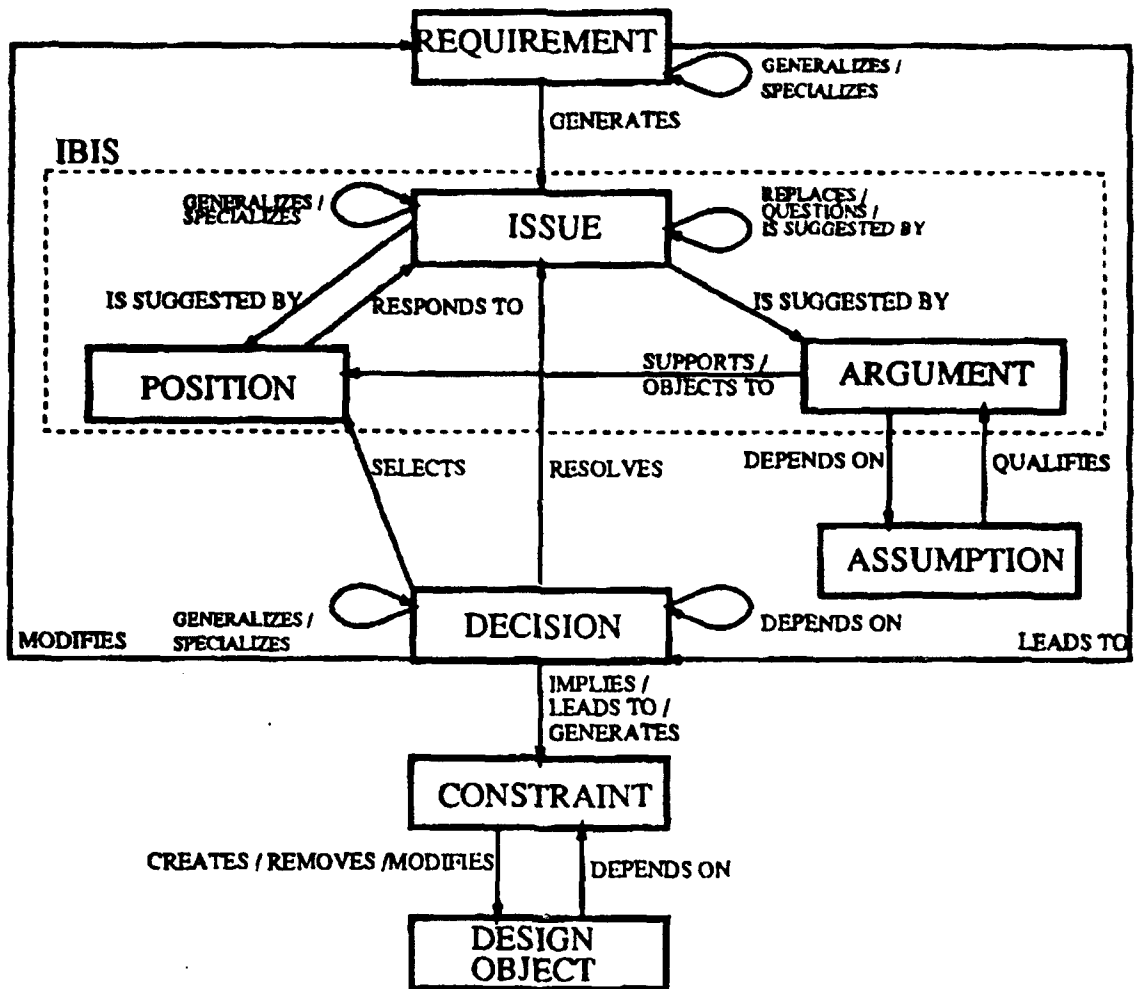


Fig. 1. Conceptual model.
REMAP Model

Figure 2

The IBIS model is contained within the REMAP model and isolated for convenience by the dotted-line box.

Useful information concerning deliberations can effectively be represented using the IBIS method. An **ISSUE** can generalize or specialize another **ISSUE**, and the relationships between nodes (e.g., **POSITIONS** and **ARGUMENTS**) can be preserved in both a positive (supports) and negative (objects to) sense. The REMAP model extends the IBIS model to include additional representations, tailoring it to

the representation of design rationale knowledge in systems development.

C. REMAP MODEL

The REMAP model incorporates primitives to model the iterative nature of real-world requirements engineering. As shown in Figure 2, the additional primitives of REMAP are: **REQUIREMENT, DECISION, ASSUMPTION, CONSTRAINT, and DESIGN OBJECT.** The IBJS model lacks primitives to represent the context in which deliberations occur. For example, the sources of issues and the decisions that are the outcomes of deliberations are not represented. REMAP nodes and their associated links create an extensive model for recording requirements engineering deliberations. The model is designed to represent the iterative nature of requirements definition refinement, and their underlying rationale, by capturing the deliberations between stakeholders (i.e., anyone providing input into the specific decision being deliberated). Additionally, REMAP provides traceability by relating requirements to design objects.

In REMAP, a **REQUIREMENT** is iteratively deliberated and refined using the other primitives of the model. **REQUIREMENTS** lead to **DECISIONS** and **DECISIONS** modify **REQUIREMENTS**. Other **REQUIREMENTS** can generalize or specialize the original **REQUIREMENT**. **REQUIREMENTS** generate **ISSUES** which may also be suggested by **ARGUMENTS** and **POSITIONS**. **ASSUMPTIONS** qualify **ARGUMENTS**. **DECISIONS** imply limits or **CONSTRAINTS** on the output **DESIGN OBJECT**.

IV. IMPLEMENTATION

A. GENERAL

The implementation of REMAP within Concept Demo was accomplished in a truly iterative fashion. We experienced a steep learning curve at the onset, while learning the various software packages and environments. We began with a tutorial offered in the Concept Demo manual. This was followed by a short course at Andersen Consulting. We learned the REFINETM wide spectrum language using a computer-aided training course. Familiarization with EMACS, UNIX, and the Sun workstation environment itself was gained using a "catch-as-catch-can" methodology. The Concept Demonstration User's Manual[Ref. 27] proved invaluable in our efforts to understand the system and its code. The RefineTM User's Manual[Ref. 28] documents the capabilities of the REFINETM LISP-based environment.

In this document, the following conventions are used:

NODES	BOLD AND ALL CAPITAL
FILE NAMES	<i>ITALICIZED AND ALL CAPITAL</i>
LINKS	<i>italicized and lower case</i>
CODE	<i>italicized, bold and off-set</i>
FUNCTIONS	<i>BOLD, ITALICIZED AND ALL CAPITALS</i>
MENU ITEMS	"bold and quotes"
MANUALS	<u>Underlined</u>
EMACS COMMANDS	<i><u>CAPITAL, ITALICIZED AND UNDERLINED</u></i>

This chapter will provide a brief system description, followed by a more detailed look at the actual system development process. A REMAP/CD tutorial will follow and provide a brief look at an example of all functionalities.

B. REMAP/CD SYSTEM DESCRIPTION

1. HARDWARE

Due to licensing restrictions, we implemented the REMAP model on a SPARC 2 workstation equipped with 48 megabytes of RAM. This was an important constraint in our work as previous work in the Concept Demo with 32 megabytes of RAM led to the machine locking up.

2. SOFTWARE

We used the UNIX/SUN OS 4.1.1 as our operating system, with its support for OPEN WINDOWS version 2 windowing software. In addition, we used the GNU EMACS 18.55.0 text editor extended with macros to run Common Lisp and REFINE™ environments.

C. IMPLEMENTATION TUTORIAL

1. BEGINNING A NEW SESSION

At this point, we assume that you are able to call up the Concept Demo. Within the EMACS buffer, type M-X RUN-A-REFINE. The system will load the REFINE environment. The version of Concept Demo that we used is *kbsa-cd-fv-pm-1-3* as received from Andersen Consulting. From within the *SCRATCH* buffer in REFINE™, ^X^F to load *SAMPLE-LOAD.LISP* into the buffer and compile using the M-X REFINE-COMPILE-BUFFER command. In the *REFINE* buffer, at the prompt, type (user::load-sample)<RET>. The extensions are now loaded in the order previously described.

From within the Concept Demo program, mouse left on the Knowledge-base button in the command menu. A pull-down menu will appear with several options. Mouse left on the **"Graph-Used-By-Relations"** menu item. This creates a window of the systems current knowledge-based modules. Mouse left on Aircraft-Domain-Spec. Another pull-down menu will appear, from this menu mouse left on **"Graph REMAP Model"**. A blank screen with **REMAP Model For KB-module** at the top should appear. Left mouse anywhere on the screen to reveal the **"Semantic Net Window Options"** menu. Left mouse on **"Add Informal Object"**. The menu that appears will display any of the eight REMAP nodes that you can create. Mouse left on the **REMAP-ISSUE** node. An input box will appear to allow you to name the node if you wish. Delete allows you to clear the box and enter the appropriate name for your **REMAP-ISSUE**. Hit <RET> when complete. Now a box appears to add any appropriate text for this **REMAP-ISSUE**. When complete type ^D. If the node does not appear, mouse right on an open space of the screen and mouse left on **"Zoom Out"**. Mouse left at a point upper left to the node and move the lower right hand corner of the grid to include the completed node within the grid. The blue, rectangular node will now appear on the screen. You may now create more nodes. Each addition of an informal node will require you to name the node and to provide more detailed information about the node in a text box provided. Each pulldown screen query provides a default answer to instantiate the node. For this tutorial, mouse left on any open space and create an **REMAP-ARGUMENT** node. A rectangular, turquoise node appears. Menu choices defined for hypertext nodes is available for the node. For example, right click on the **REMAP-ARGUMENT** and select move. This will allow the user to place the node in a convenient position.

At this point you may add links to the nodes. Mouse left on the **REMAP-ISSUE** node. A pulldown menu will appear, select **"Add Link"**. Another menu will appear that will provide all the possible outgoing links for the node you selected. Choose *issue-is-suggested-by-argument*, and select the node that this **REMAP-ISSUE** should point to by clicking left on **REMAP-ARGUMENT**. A link label, **ISSUE-SUGGESTED-BY-ARGUMENT:1**, is created with a default certainty of 1 established.

2. LINK CERTAINTY

We have implemented an attribute to links called **CERTAINTY**. This attribute gives us a weighting scale from zero to ten for each link. To modify the **CERTAINTY** of a link, mouse left on that link and a menu will appear. Select **"Modify CERTAINTY"** from the list and a new menu will appear to add a value of **ZERO** through **TEN**. Mouse left on the value you desire to assign to the link. The screen will automatically refresh and display your **CERTAINTY** value with the link.

3. NODE CERTAINTY

Each node also contains a certainty value that could represent the strength of a **REMAP-ARGUMENT** or the belief of an **REMAP-ASSUMPTION**. The certainty value is color-coded in the **REMAP-ASSUMPTION** node. For demonstration, mouse left on any open space. Select **"Add Informal Object"** and select **REMAP-ASSUMPTION** from the menu that follows. Continue as previously described to complete the node. A yellow **REMAP-ASSUMPTION** diamond is created, the default value being 1.5 for node certainty. To view the certainty value color functionality, mouse left on the yellow **REMAP-ASSUMPTION** node, select **"Modify Node Certainty"** from the menu that appears. Select node certainty of 0.5 and note the yellow **REMAP-ASSUMPTION** node changes to red, representing a node certainty of less than

1.0. This procedure can be repeated, and the selection of a value greater than 2.0 will create a green node.

4. ADDITIONAL MENU OPTIONS

Four additional menu items within the "Options For **REMAP-NODES**" menu were added to demonstrate the ability to call remote functions and/or programs. "Show **REMAP Model**" displays a representation of the generic **REMAP** model within an **XWINDOW** window. "**CUCKOO**" calls an audio program for demonstration purposes. "Show **Multimedia**" makes an external call to a multimedia program that could represent a **REMAP-ARGUMENT**, for example. "Call **EMACS**" allows us to call an external program and with **EMACS** specifically, to create extended text concerning a **REMAP** node.

5. CAPTURING DELIBERATIONS DATA

Recalling from Chapter III, we have certain **REMAP-REQUIREMENTS** to achieve when creating software. From these **REMAP-REQUIREMENTS** certain **REMAP-ISSUES** may arise that could lead to different courses of action. This could further lead to supporting and opposing **REMAP-ARGUMENTS** to the **REMAP-ISSUE**. **REMAP-ASSUMPTIONS** could lead to our **REMAP-ARGUMENTS** for or against. As each new node arises within the deliberation, we create and name the node to reflect the key concept and also provide textual elaboration of that node. Additional nodes and links are created and related in a similar fashion.

D. IMPLEMENTATION

1. APPROACH

Previously, we described how to use the system. We will now discuss the details of our implementation. The initial approach to accomplishing the implementation of **REMAP** within the Concept Demo involved extensive re-use of existing functions. We modified functions that had been used to

partially implement the IBIS model into the Concept Demo. These functions were fairly easy to locate using the EMACS text editor. The M- command proved invaluable allowing the trace of existing functions throughout the system. We were then able to evaluate the use of different functions and determine how they could be reused for our implementation. This method was successful for the initial implementation of a basic graphical browser to instantiate REMAP nodes and links within the Concept Demo. As work progressed and the implementation of new functionality became essential, new code had to be written.

In order to facilitate compilation of files with REMAP code, we created a file called *SAMPLE-LOAD.LISP*. (Appendix A) The EMACS command M-X REFINE-COMPILE-BUFFER is used to compile the buffer containing the file and then the function '(USER::*SAMPLE-LOAD*)' is used on the command line in the *REFINE* buffer. All the files that have been listed into *SAMPLE-LOAD.LISP* are now compiled in the appropriate order. This procedure compiles and loads the last saved version of the listed files. If a file is loaded in an EMACS buffer, and if changes have been saved, when you compile the *SAMPLE-LOAD.LISP* buffer with the M-X REFINE-COMPILE-BUFFER command, you will see a message in the *REFINE* buffer stating that the current version of the file that you have modified has not been compiled, but that the system will recompile the changed file for you and create a fast-loading(.fasl) version. The advantage of using the *SAMPLE-LOAD.LISP* file is that instead of loading each listed file into a buffer and compiling that buffer, only a couple of simple steps are required.

Individual functions within the EMACS editor can be compiled by placing the cursor within any portion of the function you would like to compile and executing the ^C-^C

command. 's method is useful to check the correctness of modifications of work in progress. To compile a file, you must first load that file into a buffer. This is accomplished by using the EMACS command ^X-^F and then entering the name of the file you would like to enter into the buffer. The buffer created is automatically named the same as the file you just entered. Now you are able to compile the entire buffer using the EMACS command M-X REFINE-COMPILE-BUFFER. This method is useful when several functions need compiling within the same buffer.

2. DEFINING VARIABLES

The first step in extending the Concept Demo involved the creation of appropriate variables corresponding to the REMAP model. To avoid confusion with previously set variables, the name of each variable to define a REMAP node is started with "REMAP"(i.e., REMAP-ISSUE). A patch to the original *HYPertext.RE* file written for the Concept Demo called *HYPertext-PATCH.RE* contains these definitions. Each node of the REMAP model is represented a variable within the system. The variables for the REMAP model are defined in the following format:

```
var remap-issue: object-class subtype-of remap-node
```

The super-class *remap-node* is also defined within the same file:

```
var remap-node: object-class subtype-of hypertext-node
```

The variable *hypertext-node* was previously defined within the *HYPertext.RE* file.

For each **REMAP-NODE** an attribute called *node-certainty* is then defined. This definition allows a real value to be assigned which could be used to signify, for instance, the belief in an **ASSUMPTION** node, or the certainty of an **ARGUMENT** node.

The current implementation limits values for node certainty to zero through 3.0. This can easily be modified to fit the user's needs through *MODIFY-NODE-CERTAINTY* in *GRAPH-SN-MODEL-PATCH.RE*. The following code defines the variable *node-certainty* and includes a rule that computes the *node-certainty* for a **REMAP-POSITION** based on the average of the *node-certaintys* of **REMAP-ARGUMENTS** that support it.

```
var node-certainty: map(remap-node, real)
  computed using
  remap-position (p) & ~empty (argument-supports-
    position (p)) => node-certainty (p) =
  reduce (+,image (node-certainty, argument-supports-
    position(p))) / size (argument-supports-
    position(p)),
  true => node-certainty(@@) = 1.5
```

Our next step within this same file is to define the links between the **REMAP-NODES**. As an aid in understanding the **REMAP** model, we included the name of the link and the nodes that the link connects in the variable name. An example of a link definition looks like this:

```
var issue-suggested-by-position: map(remap-issue, set
  (remap-position))
  computed-using issue-suggested-by-position(@@) = {}
```


The variable *issue-suggested-by-position* is a map of a **REMAP-ISSUE** onto a set of **REMAP-POSITION**. The (@@) is the default function used as a wildcard in patterns to fill out mandatory parts of the syntax of an object class.[Ref. 29]

When the variables are created, we needed a method to export these variables into the system 'CD package. This was done utilizing a **REFINETM** form. The following form exports **REMAP-NODE**, **REMAP-ISSUE**, and *issue-suggested-by-position* and other links and nodes mentioned in it.

form export-hypertext-patch

```
export ({'remap-node, 'remap-issue, 'issue-suggested-  
by-position,... },  
'cd);
```

3. REMAP DISPLAY

The next logical step was to incorporate the **REMAP** model components in menus so that we could create nodes and their links on a graphical browser. The file *GRAPH-SN-MODEL-PATCH.RE* contains a majority of the code for menu creation. The first item we concern ourselves with is the creation of informal objects(**REMAP-NODES**), within the system. Informal objects are objects that have a informal component such as text. This is done in a function called **ADD-INFORMAL-OBJECT**.

As can be seen below, this function checks whether the objects we intend to modify is a **REMAP-NODE**:

```
function add-informal-object (dw: diagram-window) =
  let (informal-obj-class-names: set(symbol) =
      image(name, class-subclasses
            (find-object-class('remap-node), true)))
  let (ht-class-name: symbol =
      single-menu("Choose the type of Informal Object",
                  informal-obj-class-names,
                  '::display-function, 'string-capitalize,
                  '::moving?, false, '::abort-value, *abort-value*))
  user-create-instance(find-object-class
                        (ht-class-name),
                        diagram-window-spec-module(dw))
```

Next we provided a method to initialize an instance of the type of node selected. The following initializes a **REMAP-ISSUE**:

```
Method user-initialize-new-instance (obj: remap-issue,
                                      kbm: kb-module) =
  edit-hyperstring(obj);
  add-informal-functional-requirement-hlec(obj, kbm)
```

It is relatively straight forward to add an informal object. Any **REMAP** node can be selected from the menu to be instantiated as an informal object. The next subsection will describe how the nodes appear on screen.

Now that a node called **REMAP-ISSUE** has been created, a menu that displays those options that are available for use with that new node is available. There are three components

that allow display of those options. First, a method called **MOUSE-LEFT-METHOD** has been specialized for each **REMAP-NODE**. This enables the user to click on the node and invoke a menu. An example for the **REMAP-ISSUE** looks like this:

```
method MOUSE-LEFT-METHOD
  (obj: cd::remap-issue,p,w)=
  activate-cd-menu(*menu-for-issues*,
    'mouse-left,obj,p,w)
```

This method calls a variable called **menu-for-issue**, so we precede this method with the variable statement shown below:

```
var *menu-for-issue* :menu-object =
  make-menu-object("Options for Issue Node",
    Combine-items([*sn-objects-menu-items*,
      *informal-objects-menu-items*,
      *issue-menu-items*]),
    'cd::semantic-net-window-options)
```

This variable combines several menu items as seen above. The first two items in this list are generic Concept Demo variables. The third item (**issue-menu-items**) adds those options that we considered pertinent to the **REMAP-ISSUE**. Below is a sample of the menu items we created for the **REMAP-ISSUE** node of the model. These will be more fully explained in the tutorial section.

```

var *remap-issue-menu-items*: any-type =
  <<"Call EMACS", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/local/bin/emacs"))>,
  <<"Show multi-media", (lambda (b,o,p,w)
    excl::run-shell-command
      (string(format(false,
"/usr/openwin/bin/pageview/files/is1/remap/figures/~a"))>
  <<"Cuckoo", (lambda, (b,o,p,w)
    excl::run-shell-command
      ("/usr/demo/SOUND/cuckoo.clock"))>>

```

In the file *VIEWABLE-SLOTS-PATCH.RE*, a variable called **viewable-slots-for-hypertext** is defined. This provides the legal slot definitions for all hypertext nodes (such as **REMAP-NODES**) or complains to the system when an undefined attribute is found. We have extended the definition of this variable to include REMAP specific attributes, such as *issue-replaces-issue*, as follows:

```

var cd-ui::*viewable-slots-for-hypertext*
  :set(re::binding) =

  {find-attribute-or-complain('issue-replaces-issue),
   find-attribute-or-complain('issue-questions-issue)}

```

The general construct *re::binding* used here, refers to REFINE'sTM uniform treatment of constructs that introduce named entities, such as variables, constants and types. [Ref. 30]

The *VIEWABLE-SLOTS-PATCH.RE* file also identifies the specific viewable slots for node menu items (i.e., **REMAP-ISSUE**). The format for this definition is the same as above. For instance, all the possible links from an **REMAP-ISSUE** node

are grouped into the **viewable-slots-for-remap-issue** variable.

In order to utilize the functions already established in *VIEWABLE-SLOTS.RE*, we need to add our definition of viewable-slots for each specific node-primitive. This is accomplished in a variable definition called *viewable-slots*. The definition looks like this:

```
var viewable-slots: map(re::universe, set(re::binding))
    computed-using
    remap-issue(x) => viewable-slots(x) =
        *viewable-slots-for-remap-issue*,
```

4. ADDING ATTRIBUTES TO ATTRIBUTES (FACETS)

In a manner similar to certainty for nodes, a certainty value for links was incorporated. This could represent, for example, a relative strength of a link such as *argument-depends-on-assumption*. This task proved very challenging in comparison to adding a certainty value to nodes.

The links in the REMAP/CD are REFINETM maps from one REMAP-NODE to another set of appropriate REMAP-NODES, as described earlier under DEFINING VARIABLES. (Chapter IV. C. 2.) Since the domain type of a map can only be simple REFINETM data types (i.e., string or integer) [Ref. 31], directly mapping a certainty value to a link, which is not a simple data type, is not possible. One of the creators of CD suggested a brilliant work around. This "map to a map" extension is represented in appendix G and henceforth referred to as a facet.

The first step in incorporating facets was creating a new class called *attribute-instance*. Two related variables were also created:

```
var attribute-instance: object-class subtype-of user-
    object
var attribute-instance-target: map(attribute-instance,
    re::universe) = {}
var attribute-instance-attribute: map(attribute-
    instance, re::binding) = {}
```

The new class *attribute-instance* represents a particular value of an attribute for a class. The *attribute-instance-target* maps the *attribute-instance* to the **REMAP-NODE** from which the link originates, and *attribute-instance-attribute* maps the *attribute-instance* to the **REMAP** link.

Facets are further explained through the function calls that use them to modify an attribute's certainty. Mousing left on a link makes available the menu option "**ADD CERTAINTY**". This activates the function *INVOKE-ADD-ATTRIBUTE-CERTAINTY* (see also *GRAPH-SN-MODEL-PATCH.RE*(appendix D)) as below:

```
function invoke-add-attribute-certainty(button: symbol,
    obj: object, pos: point, w:window) =
    let(new-certainty: integer: = ri::single-menu("Select
        Certainty", [0,1,2,3,4,5,6,7,8,9,10]))
    let(ai-target: re::universe = attribute-instance-
        target(obj),
```

```

    ai-attribute: re::binding = attribute-instance-
        attribute(obj),
let(source-icon: icon =
    find-existing-icon-for-spec-object(ai-target, ai-
        attribute, 'cd::certainty, new-certainty);
erase-object(obj);
update-sn-object(source-icon,w)

```

The *obj: object* is the *attribute-instance* of the link that was moused on. First, a certainty value is entered at the screen, by selecting an integer. The range of values that can be assigned can be change easily by including an appropriate set for "new-certainty" defined above. *ai-target* and *ai-attribute* are set to the value of the REMAP-NODE and the link of the *attribute instance*, respectively. In *FACETS.RE*, *Store-attribute-facet* is called with the appropriate input. A facet named "certainty" is created by the above procedure. A similar procedure can be used for creating other facets. The relevant code that follows are shown in *FACETS.RE* and can be traced by the interested reader. Here, the REMAP link and the facet name, *certainty* currently, are combined to create a matching link that has the certainty value attached. For the user, the certainty value appears mapped to the link, but, in fact, a map is attached to the REMAP-NODE, that is, the *attribute-instance-target*.

After completion of this process and returning back to the original function, the old object is erased, and a new icon representing the link and new certainty value is then drawn. This is done through the *MAKE-LABEL-FOR-SN-CERTAINTY* function.

5. INTEGRATION

All the variable definitions and functions described previously are used by functions defined in a file called *GRAPH-IBIS-MODEL-PATCH.RE*. Our implementation includes several functions to create a graphical browser for REMAP. The first function is called *GRAPH-REMAP-MODEL*. This function allows us to graph the basic REMAP primitives. The function also identifies the allowable variables to be manipulated. This function is called when you mouse left on a knowledge-base module within the "Graph-Used-by-Relations" diagram:

```
function graph-remap-model (objs: set(re::universe),
                           title-string: string) =
  graph-basic-sn-model(objs, find-attributes or complain
    ('issue-replaces-issue,'issue-questions-issue,
    '...etc...)), title-string);
let (dw: diagram-window =make-object('diagram-window),
    surf: diagram-surface = make-object('diagram-surface))
```

The function *GRAPH-REMAP-MODEL-FOR-MODULE* identifies the REMAP primitive nodes as KB module objects:

```
function graph-remap-model-for-module (kbm: kb-module) =
  let (remap-nodes: set(remap-node) = {x| (x) x in
    owned-object(kbm) & remap-node(x)})
```

Also within *GRAPH-IBIS-MODEL-PATCH.RE* are two functions that manipulate the shape and color of the various **REMAP-NODES**. This functionality allows the user an easier means to differentiate the various types of nodes defined in REMAP or to display a node in different colors depending on the value of an attribute. The function *FIND-SHAPE-FOR-SN-ICON* uses a series of simple if-elseif statements to represent

all IBIS nodes as boxes, a REMAP-ASSUMPTION as a diamond, and all other REMAP nodes as ellipses. The code follows:

```
function find-shape-for-sn-icon(obj: object) :symbol=  
  if remap-position(obj) then 'ri::box  
  elseif remap-issue(obj) then 'ri::box  
  elseif remap-argument(obj) then 'ri::box  
  elseif remap-assumption(obj) then 'ri::diamond  
  else 'ri::ellipse
```

The second function, *FIND-COLOR-FOR-SN-ICON*, adds color to each of the nodes. With the REMAP-ASSUMPTION node however, increased functionality is demonstrated with an additional if-elseif statement embedded within the first. REMAP-ASSUMPTION is created as a diamond and the additional ability to change icon colors based on certainty value is provided. The following code demonstrates how a node is colored:

```
function find-color-for-sn-icon(obj: object) :symbol =  
  let(cert: real = node-certainty(obj))  
  if remap-position(obj) then cw::magenta  
  elseif remap-argument(obj) then cw::turquoise  
  ...etc...  
  elseif remap-assumption(obj) then  
    if cert <= 1.0 then cw::red  
    elseif cert > 1.0 and cert <= 2.0 then yellow  
    else cw::green  
  else cw::white
```

These functions demonstrate a means to implement rules within the REMAP model using shapes and colors to visually identify values.

Of key importance to integration within the KBSA framework, are high-level editing commands (HLECs). The file *HLEC-PATCH.RE* contains one such example. The format is more fully explained in the Concept Demo User's Manual. When a node's certainty is changed, for example, more than stored value of real number must change. If that node is **REMAP-ASSUMPTION**, the icon that represents the node may have to change colors. HLECs enable the system as a whole to be made aware of the change, so that all other affected areas are also changed accordingly.

VI. RECOMMENDATIONS

A. COMMENTS

There was a steep learning curve to overcome in this implementation. Numerous software packages were encountered for the first time by us. Obviously, the more experience the user has in the underlying environments, the easier an implementation such as ours will be. The REFINE™ environment is based on common lisp. The documentation is reasonably well written. A CAI system provided with the package was a great starting point for our work. As noted in the discussion on facets, the REFINE™ language has some shortcomings as an object oriented language. However, REFINE™ has the functionality to satisfy our requirements. The graphical support in the REFINE™ environment provides excellent layout features, but it has limited capability for displaying objects in different colors and shapes as required in our work.

With the extensive functionality that results from an environment like REFINE™ and Concept Demo comes complexity. The CD tutorial and user's manual were pieced together with the generous help of one of the authors of Concept Demo. Without this help, the learning process would have been much slower and very difficult.

B. FUTURE INITIATIVES

1. EASE OF USE

The primary design objective in our effort has been ease-of-use for the end-users. The end-user must easily be able to enter deliberation information, and conduct conversations with other participants. The user should be

able to enter, organize and retrieve design deliberation information with minimal training on the system. The current implementation is only partially successful in this respect. Future work emphasizing "user-friendliness" would be a definite benefit. The following subsections include areas for consideration.

a. INPUT

An alternate interface which accepts textual representation of the REMAP nodes and links, could be very useful for "off line" capture of conversations without much intrusion into a deliberation session. For example, during a planning meeting, a scribe can quickly record, in textual format, pertinent deliberation data in terms of REMAP primitives. The system should be able to parse the text and develop the appropriate information in the knowledge base.

b. EASE OF UNDERSTANDING

The deliberations information should be displayed in a manner that allows for easy understanding. Realistically, we believe that the user will have to understand the concept behind the REMAP model and the logical progression of information development through the REMAP paradigm. With some basic knowledge of the REMAP method, the user would benefit from visual cues provided by the implementation itself which would include judicious use of color and shape to differentiate nodes of the model and the perceived weight given to each node and/or link.

An additional node menu option could be incorporated that would present a color coded version of the REMAP model with help facilities that describe the model components and provides a comprehensive example. This would help in the early stages of use as additional knowledge is

gained with the use of the model and its deliberation capturing ability. A color coded legend could easily be added that would change color to match the node that the pointer is in, and display appropriate options for that node.

c. EASE OF INFORMATION RETRIEVAL

In large complex problems, the number of REMAP nodes and links may be extremely large and the user may be overwhelmed. The user should have an option for limiting the amount of information that is presented. For instance, a manager might want to limit his view to **REMAP-ISSUES** that are unresolved to identify problem areas for project planning and monitoring. Mechanisms to provide aggregate views of a network of nodes could be used to provide a high level view of the deliberation information. The aggregate nodes could be exploded to provide lower level details. Additionally, filtering and query mechanisms can be setup to selectively retrieve information pertaining to topics of interest.

2. EASE OF CAPTURE

Multimedia advances have made it possible to incorporate detailed accounts of deliberation activities in terms of the REMAP model. Video and voice clips could be attached to a REMAP node instead of, or to enhance the textual support already provided by the system. These facilities would greatly enhance the ease of capture of deliberation information with minimal intervention.

3. DECISION SUPPORT

As the information in the knowledge base has a formal representation, reasoning with this knowledge can be performed to support various stakeholders. For instance, domain and task specific deductive rules can be set up to provide automated inference. An example of such a rule would be one that uses the strengths of **ASSUMPTION** and the degree of

support they provide to **ARGUMENTS** in computing the strength of belief in **ARGUMENTS**. This information, in turn, can be used in evaluating **POSITIONS** that respond to **ISSUES**.

4. SECURITY

Security is not currently addressed in this prototype system. Any actual use of the system would, of course, have to take this into serious consideration. Appropriate access (read/write) permission schemes should be developed. Also, the means to allow for "invisible" deliberations (e.g., between two upper-level managers that is not visible to lower level employees) would be very useful.

APPENDIX A

A.

NPS-KB-MODULE-LIBRARY.RE

```
!! in-package("CD")
!! in-grammar('kb-module-grammar)

%% Local library directory is set here.

the-kb-module agl-ibis-nodes
  uses-kb-modules (upper-model)
  kb-module-path "/files/is1/remap/extend/"
```


APPENDIX B

A.

SAMPLE-LOAD.LISP

```
(in-package 'USER)
```

```
;; This file compiles those files necessary to implement  
;; REMAP within the Concept Demo.
```

```
;; change the following function to match your specific site.
```

```
(defun sample-path (file-name)  
  (concatenate 'string "/files/isl/remap/extend/" file-name))
```

```
(defun load-sample ()  
  (cload (sample-path "hypertext-patch"))  
  (cload (sample-path "facets"))  
  (cload (sample-path "hlec-patch"))  
  (cload (sample-path "graph-ibis-model-patch"))  
  (cload (sample-path "graph-sn-model-patch"))  
  (cload (sample-path "viewable-slots-patch"))  
  (cload (sample-path "nps-kb-module-library"))  
  )
```

APPENDIX C

A.

HYPertext-PATCH.RE

```
!! in-package("CD")
!! in-grammar('user)
```

```
%% This file contains the variables for the REMAP model and the slots
%% associated with those node variables. It exports them and puts the slots
%% into the viewable slots.
```

```
var remap-node: object-class subtype-of hypertext-node

var remap-assumption: object-class subtype-of remap-node
var remap-constraint: object-class subtype-of remap-node
var remap-design-object: object-class subtype-of remap-node
var remap-decision: object-class subtype-of remap-node
var remap-requirement: object-class subtype-of remap-node
var remap-position: object-class subtype-of remap-node
var remap-issue: object-class subtype-of remap-node
var remap-argument: object-class subtype-of remap-node

var decision-modifies-requirement: map(remap-decision, set(remap-requirement))
  computed-using decision-modifies-requirement(@@) = {}

var requirement-generalizes-requirement: map(remap-requirement, set(remap-requirement))
  computed-using requirement-generalizes-requirement(@@) = {}

var requirement-specializes-requirement: map(remap-requirement, set(remap-requirement))
  computed-using requirement-specializes-requirement(@@) = {}

var requirement-generates-issue: map(remap-requirement, set(remap-issue))
  computed-using requirement-generates-issue(@@) = {}

var requirement-leads-to-decision: map(remap-requirement, set(remap-decision))
  computed-using requirement-leads-to-decision(@@) = {}

var issue-generalizes-issue: map(remap-issue, set(remap-issue))
  computed-using issue-generalizes-issue(@@) = {}

var issue-specializes-issue: map(remap-issue, set(remap-issue))
  computed-using issue-specializes-issue(@@) = {}

var issue-replaces-issue: map(remap-issue, set(remap-issue))
  computed-using issue-replaces-issue(@@) = {}

var issue-questions-issue: map(remap-issue, set(remap-issue))
  computed-using issue-questions-issue(@@) = {}

var issue-suggested-by-issue: map(remap-issue, set(remap-issue))
  computed-using issue-suggested-by-issue(@@) = {}

var issue-suggested-by-position: map(remap-issue, set(remap-position))
  computed-using issue-suggested-by-position(@@) = {}

var issue-suggested-by-argument: map(remap-issue, set(remap-argument))
  computed-using issue-suggested-by-argument(@@) = {}

var position-responds-to-issue: map(remap-position, set(remap-issue))
  computed-using position-responds-to-issue(@@) = {}

var argument-supports-position: map(remap-argument, set(remap-position))
  computed-using argument-supports-position(@@) = {}

var argument-objects-to-position: map(remap-argument, set(remap-position))
  computed-using argument-objects-to-position(@@) = {}
```

```

var argument-depends-on-assumption: map(remap-argument, set(remap-assumption))
  computed-using argument-depends-on-assumption(@@) = {}

var assumption-qualifies-argument: map(remap-assumption, set(remap-argument))
  computed-using assumption-qualifies-argument(@@) = {}

var decision-resolves-issue: map(remap-decision, set(remap-issue))
  computed-using decision-resolves-issue(@@) = {}

var decision-selects-position: map(remap-decision, set(remap-position))
  computed-using decision-selects-position(@@) = {}

var decision-generalizes-decision: map(remap-decision, set(remap-decision))
  computed-using decision-generalizes-decision(@@) = {}

var decision-specializes-decision: map(remap-decision, set(remap-decision))
  computed-using decision-specializes-decision(@@) = {}

var decision-depends-on-decision: map(remap-decision, set(remap-decision))
  computed-using decision-depends-on-decision(@@) = {}

var decision-implies-constraint: map(remap-decision, set(remap-constraint))
  computed-using decision-implies-constraint(@@) = {}

var decision-leads-to-constraint: map(remap-decision, set(remap-constraint))
  computed-using decision-leads-to-constraint(@@) = {}

var decision-generates-constraint: map(remap-decision, set(remap-constraint))
  computed-using decision-generates-constraint(@@) = {}

var constraint-creates-design-object: map(remap-constraint, set(remap-design-object))
  computed-using constraint-creates-design-object(@@) = {}

var constraint-removes-design-object: map(remap-constraint, set(remap-design-object))
  computed-using constraint-removes-design-object(@@) = {}

var constraint-modifies-design-object: map(remap-constraint, set(remap-design-object))
  computed-using constraint-modifies-design-object(@@) = {}

var design-object-depends-on-constraint: map(remap-design-object, set(remap-constraint))
  computed-using design-object-depends-on-constraint(@@) = {}

var node-certainty: map(remap-node, real)
  computed-using
  remap-position(p) & ~empty(argument-supports-position(p)) =>
  node-certainty(p) =
    reduce(+, image(node-certainty, argument-supports-position(p))) /
      size(argument-supports-position(p)),
  true => node-certainty(@@) = 1.5

```

```

%% If any of the below links, nodes, etc. are changed, you must go to
%% graph-ibis-model-patch and change the function graph-remap-model
%% and graph-remap-model-for-module.
%%

```

```

form export-hypertext-patch

```

```

  export({'remap-node,'remap-decision,'remap-constraint,'remap-design-object,
    'remap-assumption,'remap-position,'remap-issue,'remap-argument,
    'remap-requirement,
    'assumption-certainty,'node-certainty,
    'decision-modifies-requirement,'requirement-generalizes-requirement,
    'requirement-specializes-requirement,'requirement-generates-issue,
    'requirement-leads-to-decision,
    'issue-generalizes-issue,'issue-specializes-issue,
    'issue-replaces-issue,'issue-questions-issue,

```

```
' issue-suggested-by-issue,' issue-suggested-by-position,  
' issue-suggested-by-argument,' position-responds-to-issue,  
' argument-supports-position,' argument-objects-to-position,  
' argument-depends-on-assumption,  
' assumption-qualifies-argument,' decision-resolves-issue,  
' decision-selects-position,  
' decision-generalizes-decision,' decision-specializes-decision,  
' decision-depends-on-decision,  
' decision-implies-constraint,' decision-leads-to-constraint,  
' decision-generates-constraint,  
' constraint-creates-design-object,' constraint-removes-design-object,  
' constraint-modifies-design-object,  
' design-object-depends-on-constraint},  
' cd)
```

APPENDIX D

A.

GRAPH-SN-MODEL-PATCH.RE

```
!! in-package('cd-ui)
!! in-grammar('gr)
```

```
function add-informal-object (dw: diagram-window) =
  let (informal-obj-class-names: set (symbol) =
      image (name, class-subclasses (find-object-class ('remap-node), true)))
  let (ht-class-name: symbol =
      single-menu ("Choose the type of Informal Object",
                  informal-obj-class-names,
                  '::display-function, 'string-capitalize,
                  '::moving?, false, '::abort-value, *abort-value*))
  user-create-instance (find-object-class (ht-class-name),
                       diagram-window-spec-module ())
```

```
method user-initialize-new-instance (obj: remap-issue,
                                     kbm: kb-module) =
  edit-hyperstring (obj);
  add-informal-functional-requirement-hlec (obj, kbm)
```

```
method user-initialize-new-instance (obj: ri::icon,
                                     kbm: kb-module) =
  edit-hyperstring (obj);
  add-informal-functional-requirement-hlec (obj, kbm)
```

```
method user-initialize-new-instance (obj: attribute-instance,
                                     kbm: kb-module) =
  edit-hyperstring (obj);
  add-informal-functional-requirement-hlec (obj, kbm)
```

```
method user-initialize-new-instance (obj: remap-position,
                                     kbm: kb-module) =
  edit-hyperstring (obj);
  add-informal-functional-requirement-hlec (obj, kbm)
```

```
method user-initialize-new-instance (obj: remap-argument,
                                     kbm: kb-module) =
  edit-hyperstring (obj);
  add-informal-functional-requirement-hlec (obj, kbm)
```

```
method user-initialize-new-instance (obj: remap-assumption,
                                     kbm: kb-module) =
  edit-hyperstring (obj);
  add-informal-functional-requirement-hlec (obj, kbm)
```

```
method user-initialize-new-instance (obj: remap-constraint,
                                     kbm: kb-module) =
  edit-hyperstring (obj);
  add-informal-functional-requirement-hlec (obj, kbm)
```

```
method user-initialize-new-instance (obj: remap-design-object,
                                     kbm: kb-module) =
  edit-hyperstring (obj);
  add-informal-functional-requirement-hlec (obj, kbm)
```

```
method user-initialize-new-instance (obj: remap-decision,
                                     kbm: kb-module) =
  edit-hyperstring (obj);
  add-informal-functional-requirement-hlec (obj, kbm)
```



```
method user-initialize-new-instance (obj: remap-requirement,
                                     kbm: kb-module) =
  edit-hyperstring(obj);
  add-informal-functional-requirement-hlec(obj, kbm)
```

For Remap-Position

```
var *remap-position-menu-items*: any-type =
  <
  <"Call Emacs", (lambda (b, o, p, w)
                  excl::run-shell-command("/usr/local/bin/emacs") )>,
  <"Show multi-media", (lambda (b, o, p, w)
                        excl::run-shell-command(string(format (false, "/usr/openwin/bin/pageview
                        /files/is1/remap/figures/~a &", (symbol-to-string(name(o))) )) )) )>,
  <"Show REMAP model", (lambda (b, o, p, w) excl::run-shell-command
                        ("/usr/openwin/bin/pageview /files/is1/remap/figures/model.ps &"))>,
  <"Cuckoo", (lambda (b, o, p, w) excl::run-shell-command("/usr/demo/SOUND/cuckoo.clock"))
  >

var *menu-for-remap-positions*: menu-object =
  make-menu-object("Options for Position Node",
                  combine-items([*sn-objects-menu-items*, *informal-objects-menu-items*,
                                *remap-position-menu-items*]),
                  'cd::semantic-net-window-options)

method MOUSE-LEFT-METHOD (obj: cd::remap-position, p, w) =
  activate-cd-menu(*menu-for-remap-positions*, 'mouse-left, obj, p, w)
```

For Remap-requirement

```
var *requirement-menu-items*: any-type =
  <
  <"Call Emacs", (lambda (b, o, p, w)
                  excl::run-shell-command("/usr/local/bin/emacs") )>,
  <"Show multi-media", (lambda (b, o, p, w)
                        excl::run-shell-command(string(format (false, "/usr/openwin/bin/pageview
                        /files/is1/remap/figures/~a", (symbol-to-string(name(o))) )) )) )>,
  <"Show REMAP model", (lambda (b, o, p, w)
                        excl::run-shell-command("/usr/openwin/bin/pageview
                        /files/is1/remap/figures/model.ps"))>,
  <"Cuckoo", (lambda (b, o, p, w) excl::run-shell-command("/usr/demo/SOUND/cuckoo.clock"))
  >

var *menu-for-requirements*: menu-object =
  make-menu-object("Options for Requirement Node",
                  combine-items([*sn-objects-menu-items*, *informal-objects-menu-items*,
                                *requirement-menu-items*]),
                  'cd::semantic-net-window-options)

method MOUSE-LEFT-METHOD (obj: cd::remap-requirement, p, w) =
  activate-cd-menu(*menu-for-requirements*, 'mouse-left, obj, p, w)
```

For Remap-Issue

```
var *remap-issue-menu-items*: any-type =
```

```

<
<"Call Emacs", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/local/bin/emacs") )>,
<"Show multi-media", (lambda (b,o,p,w)
    excl::run-shell-command(string(format (false, "/usr/openwin/bin/pageview
    /files/isl/remap/figures/~a", (symbol-to-string(name(o))) )) )) >,
<"Show REMAP model", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/openwin/bin/pageview
    /files/isl/remap/figures/model.ps"))>,
<"Cuckoo", (lambda (b,o,p,w) excl::run-shell-command("/usr/demo/SOUND/cuckoo.clock"))
>

```

```

var *menu-for-remap-issue*: menu-object =
  make-menu-object("Options for Issue Node",
    combine-items([*sn-objects-menu-items*, *informal-objects-menu-items*,
    *remap-issue-menu-items*]),
    'cd::semantic-net-window-options)

```

```

method MOUSE-LEFT-METHOD (obj: cd::remap-issue,p,w) =
  activate-cd-menu(*menu-for-remap-issue*, 'mouse-left, obj, p, w)

```

For Remap-Constraint

```

var *remap-constraint-menu-items*: any-type =
  <
  <"Call Emacs", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/local/bin/emacs") )>,
  <"Show multi-media", (lambda (b,o,p,w)
    excl::run-shell-command(string(format (false, "/usr/openwin/bin/pageview
    /files/isl/remap/figures/~a", (symbol-to-string(name(o))) )) )) >,
  <"Show REMAP model", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/openwin/bin/pageview
    /files/isl/remap/figures/model.ps"))>,
  <"Cuckoo", (lambda (b,o,p,w) excl::run-shell-command("/usr/demo/SOUND/cuckoo.clock"))
  >

```

```

var *menu-for-remap-constraint*: menu-object =
  make-menu-object("Options for Constraint Node",
    combine-items([*sn-objects-menu-items*, *informal-objects-menu-items*,
    *remap-constraint-menu-items*]),
    'cd::semantic-net-window-options)

```

```

method MOUSE-LEFT-METHOD (obj: cd::remap-constraint,p,w) =
  activate-cd-menu(*menu-for-remap-constraint*, 'mouse-left, obj, p, w)

```

For Remap-Argument

```

var *remap-argument-menu-items*: any-type =
  <
  <"Call Emacs", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/local/bin/emacs") )>,
  <"Show multi-media", (lambda (b,o,p,w)
    excl::run-shell-command (string(format (false, "/usr/openwin/bin/pageview
    /files/isl/remap/figures/~a", (symbol-to-string(name(o))) )) )) >,
  <"Show REMAP model", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/openwin/bin/pageview

```

```

    /files/is1/remap/figures/model.ps"))>,
  <"Cuckoo", (lambda (b,o,p,w) excl::run-shell-command("/usr/demo/SOUND/cuckoo.clock"))
  >

```

```

var *menu-for-remap-argument*: menu-object =
  make-menu-object("Options for Argument Node",
    combine-items([*sn-objects-menu-items*, *informal-objects-menu-items*,
      *remap-argument-menu-items*]),
    'cd::semantic-net-window-options)

```

```

method MOUSE-LEFT-METHOD (obj: cd::remap-argument,p,w) =
  activate-cd-menu(*menu-for-remap-argument*, 'mouse-left, obj,p,w)

```

%%%%%%%% For Assumption

```

function invoke-modify-node-certainty(button: symbol, obj: object,
                                     pos: point, w: window) =
  let(new-certainty: real = ri::single-menu("Select Node Certainty",
    [0.5,1.0,1.5,2.0,2.5,3.0]))
  let (node-cert: re::binding = cd::find-or-create-attribute
    ('node-certainty,'obj,'real,diagram-window-spec-module(w)))
  let(dg-surface: diagram-surface = surface-viewed(w))
  let(source-icon: icon =
    find-existing-icon-for-spec-object(obj,dg-surface))
  store-attribute(obj,node-cert,new-certainty);
  cd-ui::icon-color(find-existing-icon-for-spec-object(obj,dg-surface))
    <- cd-ui::find-color-for-sn-icon(obj);
  cd::make-inconsistent-spec-object(obj);
  modify-node-certainty-hlec(obj,node-cert,new-certainty)

```

```

var *assumption-menu-items*: any-type =
  <
  <"Add Assumption Certainty",'invoke-add-assumption-certainty>,
  <"Modify Node Certainty",'invoke-modify-node-certainty>,
  <"Call Emacs", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/local/bin/emacs") )>,
  <"Show multi-media", (lambda (b,o,p,w)
    excl::run-shell-command(string(format(false,"/usr/openwin/bin/pageview
    /files/is1/remap/figures/~a", (symbol-to-string(name(o)))))) )>,
  <"Show REMAP model", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/openwin/bin/pageview
    /files/is1/remap/figures/model.ps"))>,
  <"Cuckoo", (lambda (b,o,p,w) excl::run-shell-command("/usr/demo/SOUND/cuckoo.clock"))
  >

```

```

var *menu-for-assumption*: menu-object =
  make-menu-object("Options for Assumption Node",
    combine-items([*sn-objects-menu-items*, *informal-objects-menu-items*,
      *assumption-menu-items*]),
    'cd::semantic-net-window-options)

```

```

method MOUSE-LEFT-METHOD (obj: cd::remap-assumption,p,w) =
  activate-cd-menu(*menu-for-assumption*, 'mouse-left, obj,p,w)

```

%%%%%%%% For Remap-Decision

```

var *decision-menu-items*: any-type =

```

```

<
<"Call Emacs", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/local/bin/emacs" )>,
<"Show multi-media", (lambda (b,o,p,w)
    excl::run-shell-command(string(format(false,"/usr/openwin/bin/pageview
    /files/is1/remap/figures/~a", (symbol-to-string(name(o)))) )) )>,
<"Show REMAP model", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/openwin/bin/pageview
    /files/is1/remap/figures/model.ps"))>,
<"Cuckoo", (lambda (b,o,p,w) excl::run-shell-command("/usr/demo/SOUND/cuckoo.clock"))
>

var *menu-for-decision*: menu-object =
  make-menu-object("Options for Decision Node",
    combine-items([*sn-objects-menu-items*, *informal-objects-menu-items*,
    *decision-menu-items*]),
    'cd::semantic-net-window-options)

method MOUSE-LEFT-METHOD (obj: cd::remap-decision,p,w) =
  activate-cd-menu(*menu-for-decision*, 'mouse-left, obj,p,w)

%%%%%%%%%% For Remap-Design-Object

var *remap-design-object-menu-items*: any-type =
  <
  <"Call Emacs", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/local/bin/emacs" )>,
  <"Show multi-media", (lambda (b,o,p,w)
    excl::run-shell-command(string(format(false,"/usr/openwin/bin/pageview
    /files/is1/remap/figures/~a", (symbol-to-string(name(o)))) )) )>,
  <"Show REMAP model", (lambda (b,o,p,w)
    excl::run-shell-command("/usr/openwin/bin/pageview
    /files/is1/remap/figures/model.ps"))>,
  <"Cuckoo", (lambda (b,o,p,w) excl::run-shell-command("/usr/demo/SOUND/cuckoo.clock"))
  >

var *menu-for-remap-design-object*: menu-object =
  make-menu-object("Options for Design Object Node",
    combine-items([*sn-objects-menu-items*, *informal-objects-menu-items*,
    *remap-design-object-menu-items*]),
    'cd::semantic-net-window-options)

method MOUSE-LEFT-METHOD (obj: cd::remap-design-object,p,w) =
  activate-cd-menu(*menu-for-remap-design-object*, 'mouse-left, obj,p,w)

function invoke-add-attribute-certainty(button: symbol, obj: object,
    pos: point, w: window) =
  let(new-certainty: integer = ri::single-menu("Select Certainty",
    [0,1,2,3,4,5,6,7,8,9,10]))
  let(ai-target: re::universe = attribute-instance-target(obj),
    ai-attribute: re::binding = attribute-instance-attribute(obj),
    dg-surface: diagram-surface = surface-viewed(w))
  let(source-icon: icon =
    find-existing-icon-for-spec-object(ai-target,dg-surface))
  cd::store-attribute-facet(ai-target,ai-attribute,
    'cd::certainty,new-certainty);

```

```
erase-object(obj);
update-sn-icon(source-icon,w)
```

```
function invoke-retrieve-certainty(button: symbol, obj: object,
                                   pos: point, w: window) =
  cd::retrieve-attribute-facet(attribute-instance-target(obj),
                              attribute-instance-attribute(obj),'cd::certainty)
```

```
var *sn-objects-menu-items*: any-type =
  <
  <"Modify Certainty",'invoke-add-attribute-certainty,'sn-graph?>,
  <"Retrieve Certainty",'invoke-retrieve-certainty,'sn-graph?>,
  <"Add Link",'add-sn-link,'sn-graph?>,
  <"Create Possible Resolutions",'invoke-make-possible-resolutions,
  'issue?>,
  <"Show Links in Current View",'show-sn-links-and-refresh,'sn-graph?>,
  <"Show Selected Links",'show-selected-links-and-refresh,'sn-graph?>
  >
```

```
var *menu-for-slots*: menu-object =
  make-menu-object("Options for Slots",*sn-objects-menu-items*,
                  'cd::options-for-classes)
```

```
method mouse-left-method (obj: ri::icon,pos: point, w: window) =
  if agl-link-label?(obj)
  then mouse-left-method(spec-object-for-link(agl-link-for-label(obj)),
                        pos,w)
  else mouse-left-method(spec-object-for-icon(obj),pos,w)
```

```
method mouse-left-method (obj: attribute-instance,pos: point, w: window) =
  activate-cd-menu(*menu-for-slots*,'mouse-left,obj,pos,w)
```

APPENDIX E

A.

HLEC-PATCH.RE

```
!! in-package("CD")
!! in-grammar('gr)
```

```
function modify-node-certainty(obj:object, node-cert: re::binding, new-certainty: real) =
  cd::make-inconsistent-spec-object(obj);
  cd::make-inconsistent-spec-object(node-cert)
```

```
!! in-grammar('hlec-grammar)
```

```
hlec modify-node-certainty-template
  context-name-pattern["Changed the value of", "0"]
  context-description-pattern["Set the value of", "2", "on", "0", "to", "3"]
```

```
!! in-grammar('user)
```

```
form gen-hlecs
generate-hlecs ()
```

APPENDIX F

A.

VIEWABLE-SLOTS-PATCH.RE


```
!! in-package("CD-UI")
!! in-grammar('gr)
```

```
%% This file is primarily a duplicate of the original
%% Concept Demop work with minor changes to incorporate
%% the REMAP implementation.
```

```
var *viewable-slots-for-diagram-windows* : set(re::binding) =
{find-attribute-or-complain('cd-ui::max-tree-depth),
 find-attribute-or-complain('cd-ui::tree-depth-spacing),
 find-attribute-or-complain('cd-ui::tree-breadth-spacing),
 find-attribute-or-complain('cd-ui::tree-graph-direction)
}
```

```
var *default-viewable-slots*: set(re::binding) =
{find-attribute-or-complain('cd::class),
 find-attribute-or-complain('attached-notes)
 %%find-attribute-or-complain('agenda-items)
}
union
{find-attribute-or-complain('created-by),
 find-attribute-or-complain('last-modified-by),
 find-attribute-or-complain('cd::created-on-date),
 find-attribute-or-complain('cd::last-modified-on-date)}
```

```
var *viewable-slots-for-kb-module-objects*: set(re::binding) =
{find-attribute-or-complain('owned-by),
 find-attribute-or-complain('created-by),
 find-attribute-or-complain('last-modified-by),
 find-attribute-or-complain('formalizes),
 find-attribute-or-complain('implements),
 find-attribute-or-complain('unresolved-issues)}
union find-attributes-or-complain({'created-by-task,'deleted-by-task,
 'modified-by-tasks})
```

```
union
{find-attribute-or-complain('cd::created-on-date),
 find-attribute-or-complain('cd::last-modified-on-date)}
```

```
var *viewable-slots-for-modules*: set(re::binding) =
*default-viewable-slots* union
{find-attribute-or-complain('owned-objects),
 find-attribute-or-complain('uses-kb-modules),
 find-attribute-or-complain('exported-object-names),
 find-attribute-or-complain('exported-objects),
 find-attribute-or-complain('kb-module-version),
 find-attribute-or-complain('development-history)
}
```

```
var *viewable-slots-for-sms*: set(re::binding) =
*default-viewable-slots* union *viewable-slots-for-modules* union
{find-attribute-or-complain('requirement-modules),
 find-attribute-or-complain('test-modules),
 find-attribute-or-complain('used-by-kb-modules)}
```

```
var *viewable-slots-for-tms*: set(re::binding) =
*default-viewable-slots* union *viewable-slots-for-modules* union
{find-attribute-or-complain('test-module-of)}
```

```
var *viewable-slots-for-rms*: set(re::binding) =
*default-viewable-slots* union *viewable-slots-for-modules* union
{find-attribute-or-complain('implementation-modules)}
```

```
var *viewable-slots-for-relations*: set(re::binding) =
*default-viewable-slots*
```

```

union *viewable-slots-for-kb-module-objects* union
{find-attribute-or-complain(' relation-range),
 find-attribute-or-complain(' relation-domain),
 find-attribute-or-complain(' range-cardinality),
 find-attribute-or-complain(' domain-cardinality),
 find-attribute-or-complain(' used-by),
 find-attribute-or-complain(' certainty),
 find-attribute-or-complain(' modified-by)}

var *viewable-slots-for-er-atts*: set(re::binding) =
*default-viewable-slots*
union *viewable-slots-for-kb-module-objects* union
{find-attribute-or-complain(' used-by),
 find-attribute-or-complain(' modified-by)}

var cd-ui::*viewable-slots-for-hypertext* : set(re::binding) =
{find-attribute-or-complain(' formalized-by),
 find-attribute-or-complain(' implemented-by),
 find-attribute-or-complain(' responds-to),
 find-attribute-or-complain(' supports),
 find-attribute-or-complain(' objects-to),
 find-attribute-or-complain(' questions),
 find-attribute-or-complain(' replaces),
 find-attribute-or-complain(' generalizes),
 find-attribute-or-complain(' specializes),
 find-attribute-or-complain(' is-suggested-by),
 find-attribute-or-complain(' substitutes-for),
 find-attribute-or-complain(' substituted-for-by),
 find-attribute-or-complain(' suggests),
 find-attribute-or-complain(' defines),
 find-attribute-or-complain(' qualifies),
 find-attribute-or-complain(' triggers),
 find-attribute-or-complain(' exemplifies),
 find-attribute-or-complain(' precedes),
 find-attribute-or-complain(' follows),
 find-attribute-or-complain(' requires-input),
 find-attribute-or-complain(' produces-output),
 find-attribute-or-complain(' done-using),
 find-attribute-or-complain(' check-for-interaction),
 find-attribute-or-complain(' check-for-redundancy),
 find-attribute-or-complain(' check-for-conflict),
 find-attribute-or-complain(' resolve-contradiction),
 find-attribute-or-complain(' describes-incompleteness-of),
 find-attribute-or-complain(' describes-incorrectness-of),
 find-attribute-or-complain(' describes-general-inadequacy-of),
 find-attribute-or-complain(' see-also),
 find-attribute-or-complain(' unresolved-issues),
 find-attribute-or-complain(' describes-issue-for),
 find-attribute-or-complain(' described-issue-for),
 find-attribute-or-complain(' resolved-by),
 find-attribute-or-complain(' possible-resolutions),

%%% REMAP specific attributes are added here.

find-attribute-or-complain(' node-certainty),
find-attribute-or-complain(' assumption-certainty),
find-attribute-or-complain(' decision-modifies-requirement),
find-attribute-or-complain(' requirement-generalizes-requirement),
find-attribute-or-complain(' requirement-specializes-requirement),
find-attribute-or-complain(' requirement-generates-issue),
find-attribute-or-complain(' requirement-leads-to-decision),
find-attribute-or-complain(' issue-generalizes-issue),
find-attribute-or-complain(' issue-specializes-issue),
find-attribute-or-complain(' issue-replaces-issue),
find-attribute-or-complain(' issue-questions-issue),

```

```

find-attribute-or-complain(' issue-suggested-by-issue),
find-attribute-or-complain(' issue-suggested-by-position),
find-attribute-or-complain(' issue-suggested-by-argument),
find-attribute-or-complain(' position-responds-to-issue),
find-attribute-or-complain(' argument-supports-position),
find-attribute-or-complain(' argument-objects-to-position),
find-attribute-or-complain(' argument-depends-on-assumption),
find-attribute-or-complain(' assumption-qualifies-argument),
find-attribute-or-complain(' decision-resolves-issue),
find-attribute-or-complain(' decision-selects-position),
find-attribute-or-complain(' decision-generalizes-decision),
find-attribute-or-complain(' decision-specializes-decision),
find-attribute-or-complain(' decision-depends-on-decision),
find-attribute-or-complain(' decision-implies-constraint),
find-attribute-or-complain(' decision-leads-to-constraint),
find-attribute-or-complain(' decision-generates-constraint),
find-attribute-or-complain(' constraint-creates-design-object),
find-attribute-or-complain(' constraint-removes-design-object),
find-attribute-or-complain(' constraint-modifies-design-object),
find-attribute-or-complain(' design-object-depends-on-constraint)

```

```

}
```

```

var *viewable-slots-for-hypernodes*: set(re::binding) =
  *default-viewable-slots*
  union *viewable-slots-for-kb-module-objects*
  union find-attributes-or-complain({' sub-nodes,' super-node})
  union *viewable-slots-for-hypertext*
```

```

var *viewable-slots-for-explanations* :set(re::binding) =
  find-attributes-or-complain({' object-explained}) union
  *viewable-slots-for-hypernodes*
```

```

var *viewable-slots-for-hypertext-requirements*: set(re::binding) =
  *default-viewable-slots*
  union *viewable-slots-for-hypernodes*
  union *viewable-slots-for-hypertext*
```

```

var *viewable-slots-for-notes*: set(re::binding) =
  *default-viewable-slots*
  union *viewable-slots-for-kb-module-objects*
  union {find-attribute-or-complain(' attached-to)}
```

```

var *viewable-slots-for-agenda-items*: set(re::binding) =
  *default-viewable-slots*
  union *viewable-slots-for-kb-module-objects*
  %%{find-attribute-or-complain(' spec-objects-for-agenda-item)}
```

```

%% Viewable slots for each REMAP node that only includes
%% the appropriate links for that node.
```

```

var *viewable-slots-for-remap-requirement-: set(re::binding) =
  {find-attribute-or-complain(' requirement-leads-to-decision),
  find-attribute-or-complain(' requirement-generates-issue),
  find-attribute-or-complain(' requirement-generalizes-requirement),
  find-attribute-or-complain(' requirement-specializes-requirement)}
```

```

var *viewable-slots-for-remap-issue*: set(re::binding) =
  {find-attribute-or-complain(' issue-generalizes-issue),
  find-attribute-or-complain(' issue-specializes-issue),
  find-attribute-or-complain(' issue-replaces-issue),
  find-attribute-or-complain(' issue-questions-issue),
  find-attribute-or-complain(' issue-suggested-by-issue),
  find-attribute-or-complain(' issue-suggested-by-argument),
  find-attribute-or-complain(' issue-suggested-by-position)}
```

```

var *viewable-slots-for-remap-position*: set(re::binding) =
  {find-attribute-or-complain('position-responds-to-issue)}

var *viewable-slots-for-remap-argument*: set(re::binding) =
  {find-attribute-or-complain('argument-supports-position),
   find-attribute-or-complain('argument-objects-to-position),
   find-attribute-or-complain('argument-depends-on-assumption)}

var *viewable-slots-for-remap-assumption*: set(re::binding) =
  {find-attribute-or-complain('assumption-qualifies-argument)}

var *viewable-slots-for-remap-decision*: set(re::binding) =
  {find-attribute-or-complain('decision-modifies-requirement),
   find-attribute-or-complain('decision-generalizes-decision),
   find-attribute-or-complain('decision-specializes-decision),
   find-attribute-or-complain('decision-depends-on-decision),
   find-attribute-or-complain('decision-implies-constraint),
   find-attribute-or-complain('decision-leads-to-constraint),
   find-attribute-or-complain('decision-generates-constraint),
   find-attribute-or-complain('decision-resolves-issue)}

var *viewable-slots-for-remap-constraint*: set(re::binding) =
  {find-attribute-or-complain('constraint-creates-design-object),
   find-attribute-or-complain('constraint-removes-design-object),
   find-attribute-or-complain('constraint-modifies-design-object)}

var *viewable-slots-for-remap-design-object*: set(re::binding) =
  {find-attribute-or-complain('design-object-depends-on-constraint)}

var *viewable-slots-for-invariant-violations*: set(re::binding) =
  {find-attribute-or-complain('violation-objects),
   find-attribute-or-complain('violated-invariant)} union
  *default-viewable-slots* union *viewable-slots-for-kb-module-objects*

var *viewable-slots-for-invariants*: set(re::binding) =
  {find-attribute-or-complain('maintained-by)} union
  *default-viewable-slots* union *viewable-slots-for-kb-module-objects*

var *viewable-slots-for-demons*: set(re::binding) =
  {find-attribute-or-complain('maintains)} union
  *default-viewable-slots* union *viewable-slots-for-kb-module-objects*

var *viewable-slots-for-people*: set(re::binding) =
  *default-viewable-slots*
  union *viewable-slots-for-kb-module-objects*
  union {find-attribute-or-complain('home-directory),
         find-attribute-or-complain('cd-user-directory),
         find-attribute-or-complain('cd::assigned-to-tasks),
         find-attribute-or-complain('on-critical-path?),
         find-attribute-or-complain('percent-committed)
        }

var *viewable-slots-for-staff*: set(re::binding) =
  *viewable-slots-for-people*
  union
  find-pma-attributes({'pma::PERCENTAGE-AVAILABILITY,'pma::cost-per-day,
                     'pma::skills})

function viewable-slots-for-run-time-instance (o: object) =
  {a | (a a in class-attributes(instance-of(o), true) &
        defined?(owned-by(a)))} union {find-attribute-or-complain('cd::class)}

var *viewable-slots-for-pma-task-plans*: set(re::binding) =

```

```

    *default-viewable-slots* union
(if cd::*pma-loaded?*
 then
  find-pma-attributes({'pma::task-type,'pma::scheduled-start,
    'pma::earliest-start,
    'pma::latest-start,
    'pma::scheduled-finish,
    'pma::earliest-finish,
    'pma::latest-finish,
    'pma::personnel-commitments})
 else {})

var *viewable-slots-for-cd-tasks*: set(re::binding) =
  find-attributes-or-complain({'objects-created,'objects-modified,
    'resolves,'cd::completed?,
    'sub-tasks,'cd::issue-to-resolve,
    'cd::assigned-to,'critical-path-task?,
    'estimated-person-hours,'milestone?,
    'in-violation-of-policies,
    'scheduled-completion-date})
  union *viewable-slots-for-pma-task-plans*
  union *viewable-slots-for-hypernodes*

var *non-functional-req-slots-for-classes*: set(re::binding) =
  find-attributes-or-complain({'maximum-number-of-instances,
    'minimum-number-of-instances,'average-number-of-instances,
    'maximum-creation-frequency,'average-creation-frequency})

var *non-functional-req-slots-for-functions*: set(re::binding) =
  find-attributes-or-complain({'occurance-frequency,'minimum-processing-time,
    'required-reliability,'required-fault-tolerance,'required-survivabilit,})

var *viewable-slots-for-functions*: set(re::binding) =
  *default-viewable-slots*
  union *viewable-slots-for-kb-module-objects* union
    find-attributes-or-complain({'uses,'modifies,'used-by}) union
  *non-functional-req-slots-for-functions*

var *viewable-slots-for-classes*: set(re::binding) =
  *non-functional-req-slots-for-classes*
  union *default-viewable-slots* union
  *viewable-slots-for-kb-module-objects*

%% REMAP viewable slots.

var viewable-slots: map(re::universe, set(re::binding))
  computed-using

  remap-requirement(x) => viewable-slots(x) =
  *viewable-slots-for-remap-requirement*,
  remap-issue(x) => viewable-slots(x) =
  *viewable-slots-for-remap-issue*,
  remap-position(x) => viewable-slots(x) =
  *viewable-slots-for-remap-position*,
  remap-argument(x) => viewable-slots(x) =
  *viewable-slots-for-remap-argument*,
  remap-assumption(x) => viewable-slots(x) =
  *viewable-slots-for-remap-assumption*,
  remap-decision(x) => viewable-slots(x) =
  *viewable-slots-for-remap-decision*,
  remap-constraint(x) => viewable-slots(x) =
  *viewable-slots-for-remap-constraint*,
  remap-design-object(x) => viewable-slots(x) =
  *viewable-slots-for-remap-design-object*,

```

```

object-class?(x) => viewable-slots(x) =
*viewable-slots-for-classes*,
person(x) => viewable-slots(x) =
*viewable-slots-for-people*,
pma::staff(x) => viewable-slots(x) =
*viewable-slots-for-staff*,
cd-task(x) => viewable-slots(x) = *VIEWABLE-SLOTS-FOR-CD-TASKS*,
re::invariant-op(x) => viewable-slots(x) =
*viewable-slots-for-invariants*,
re::demon-op(x) => viewable-slots(x) =
*viewable-slots-for-demons*,
invariant-violation-record(x) => viewable-slots(x) =
*viewable-slots-for-invariant-violations*,
run-time-instance?(x) => viewable-slots(x) =
viewable-slots-for-run-time-instance(x),
kb-module(x) => viewable-slots(x) = *viewable-slots-for-modules*,
re::vfunction-op(x) => viewable-slots(x) = *viewable-slots-for-functions*,
er-attribute?(x) => viewable-slots(x) = *viewable-slots-for-er-atts*,
er-relation?(x) => viewable-slots(x) = *viewable-slots-for-relations*,
%%agenda-item(x) =>
%%viewable-slots(x) = *viewable-slots-for-agenda-items*,
hypertext-requirement(x) =>
viewable-slots(x) = *viewable-slots-for-hypertext-requirements*,
explanation(x) => viewable-slots(x) = *viewable-slots-for-explanations*,
note(x) =>
viewable-slots(x) = *viewable-slots-for-notes*,
pma-thing?(x) => viewable-slots(x) = viewable-slots-for-pma-object(x),
hypertext-node(x) => viewable-slots(x) = *viewable-slots-for-hypernodes*,
ri::diagram-window(x) => viewable-slots(x) =
*viewable-slots-for-diagram-windows*,
true => viewable-slots(@@) =
*default-viewable-slots* union *viewable-slots-for-kb-module-objects*

```

```

var *display-undefined-attributes?: boolean = false

```

```

function interesting-value? (val:any-type) :boolean =
if cd::*specification-debugging-enabled-p*
then defined?(val)
else defined?(val) and not(null(val))

```

```

var slots-to-hide: map(re::universe, set(re::binding))
computed-using slots-to-hide()

```

```

function find-viewable-slots (o:object) :seq(re::binding) =
let(possible-attributes = setdiff(viewable-slots(o),
slots-to-hide(instance-of(o))))
if *display-undefined-attributes?*
then sort-objects(possible-attributes)
else
sort-objects(filter(lambda (a) interesting-attribute?(o,a),
possible-attributes))

```

```

var *default-hypertext-slots*: set(re::binding) =
{find-attribute-or-complain('hypertext-string)}

```

```

var *all-hypertext-slots*: set(re::binding) =
*default-hypertext-slots*

```

```

var *default-max-depth*: integer = 50
var max-tree-depth: map(diagram-window, integer)
computed-using max-tree-depth(@@) = *default-max-depth*
var tree-depth-spacing: map(diagram-window, real)
computed-using tree-depth-spacing(@@) = 1.8

```

```

var tree-breadth-spacing: map(diagram-window, real)
  computed-using tree-breadth-spacing(@@) = 1.0
var tree-graph-direction: map(diagram-window, symbol)
  computed-using tree-graph-direction(@@) = 'right

form go-form
  cache('max-tree-depth, true);
  cache('tree-depth-spacing, true);
  cache('tree-breadth-spacing, true);
  cache('tree-graph-direction, true)

!! in-package('cd)

function find-atts-to-copy (obj-class: object) :set(re::binding) =
  let(atts-to-copy =
    class-attributes(obj-class, true) less find-attribute('name))
  atts-to-copy <-
    filter(lambda(att) symbol-package(name(att)) ~= find-package("CD-UI"),
      atts-to-copy);
  %%atts-to-copy <- filter(lambda(att) empty(re::using-assertions(att)),
  %%
    atts-to-copy);
  atts-to-copy <- setdiff(atts-to-copy,
    find-attributes-or-complain({'created-by-task,'deleted-by-task,
      'modified-by-tasks,'created-on-date,'last-modified-on-date,
      'created-by,'last-modified-by,'pma::version-name}));
  atts-to-copy

```

APPENDIX G

A.

FACETS.RE


```
!! in-package("CD")
!! in-grammar('user)
```

```
##### Latest version of facets.re that allows attributes of
##### attributes, and adds attribute-instance object
```

```
function attribute-facet-name (attr: re::binding, facet-name: symbol)
: symbol =
let (facet-package: any-type = symbol-package(facet-name),
    facet-name-string: string = string-upcase(format(false, "~a--a",
                                                    name(attr), facet-name)))
intern(facet-name-string, facet-package)
```

```
function find-attribute-facet (attr: re::binding, facet-name: symbol) =
find-attribute(attribute-facet-name(attr, facet-name))
```

```
function find-or-create-attribute-facet (attr: re::binding,
                                         facet-name: symbol) =
let (existing-facet: re::binding = find-attribute-facet(attr, facet-name))
if defined?(existing-facet)
then existing-facet
else create-attribute-facet(attr, facet-name)
```

```
function create-attribute-facet (attr: re::binding, facet-name: symbol) =
let (facet-domain-class: re::binding = find-relation-domain-class(attr),
    attribute-facet-name: symbol = attribute-facet-name(attr, facet-name))
display-debug-message(format(false,
    "~%Generating new attribute to serve as facet: ~a~%", attribute-facet-name));
make-attribute(attribute-facet-name, name(facet-domain-class),
    `any-type', true)
```

```
function retrieve-attribute-facet (target: re::universe, attr: re::binding,
                                   facet-name: symbol) =
let (attribute-facet: re::binding = find-attribute-facet(attr, facet-name))
if defined?(attribute-facet)
then retrieve-attribute(target, attribute-facet)
else display-debug-message(format(false,
    "~%Warning! Tried to retrieve undefined facet: ~a from ~a",
    facet-name, name(attr)));
undefined
```

```
function store-attribute-facet (target: re::universe, attr: re::binding,
                                facet-name: symbol, new-value: any-type) =
let (attribute-facet: re::binding =
    find-or-create-attribute-facet(attr, facet-name))
store-attribute(target, attribute-facet, new-value)
```

```
##### Defining a new class "attribute-instance" to stand for
##### the particular value of an attribute on a class
```

```
var attribute-instance: object-class subtype-of user-object
var attribute-instance-target: map(attribute-instance, re::universe) = {}
var attribute-instance-attribute: map(attribute-instance, re::binding) = {}
```

```
function find-or-create-attribute-instance (instance-target: re::universe,
                                           instance-attribute: re::binding)
: attribute-instance =
let (existing-attribute-instance: attribute-instance =
    find-attribute-instance(instance-target, instance-attribute))
if defined?(existing-attribute-instance)
then existing-attribute-instance
```

```

else create-attribute-instance(instance-target,instance-attribute)

function find-attribute-instance (instance-target: re::universe,
                                instance-attribute: re::binding)
: attribute-instance =
let(all-attribute-instances: set(attribute-instance) =
    instances(find-object-class('attribute-instance),true))
let(existing-attribute-instance: attribute-instance =
    some(x)(x in all-attribute-instances &
            attribute-instance-target(x) = instance-target &
            attribute-instance-attribute(x) = instance-attribute))
existing-attribute-instance

function create-attribute-instance (instance-target: re::universe,
                                   instance-attribute: re::binding)
: attribute-instance =
let(new-attribute-instance: attribute-instance =
    make-object('attribute-instance))
attribute-instance-target(new-attribute-instance) <- instance-target;
attribute-instance-attribute(new-attribute-instance) <- instance-attribute;
new-attribute-instance

form export-attribute-instance-info
export({'cd::find-or-create-attribute-instance,'cd::attribute-instance,
       'cd::attribute-instance-target,'cd::attribute-instance-attribute,
       'cd::store-attribute-facet, 'cd::retrieve-attribute-facet,
       'cd::find-attribute-instance},
      'cd)

```

APPENDIX H

A.

GRAPH-IBIS-MODEL-PATCH.RE

```

!! in-package("CD-UI")
!! in-grammar('user)

function graph-ibis-model (objs: set(re::universe), title-string: string) =
  graph-basic-sn-model(objs,
    find-attributes-or-complain({'responds-to,'supports,'objects-to,
      'questions,'replaces,'generalizes,
      'specializes,'is-suggested-by}),
    title-string)

function graph-ibis-model-for-module (kbm: kb-module) =
  let(ibis-nodes: set(ibis-node) = {x|(x) x in owned-objects(kbm)
    & ibis-node(x)})
  graph-ibis-model(ibis-nodes,
    format(false, "IBIS Model for KB-module: ~a",
      name(kbm)))

function contains-ibis-nodes? (button: symbol, kbm: object,
  pos: point, w: window) :boolean =
  ex(x)(x in owned-objects(kbm) & ibis-node(x))

function invoke-graph-ibis-model-for-module (button: symbol, thing: object,
  pos: point, w: window) = graph-ibis-model-for-module(thing)

%% Incorporated to work with attribute facets.

function show-sn-linkage (att: object, obj: object, att-value: any-type,
  dw:diagram-window,only-if-visible?: boolean) =
  (if listp(att-value)
    then
      (if length(att-value) > 3 & ~only-if-visible? then
        (display-message(
          format(false, "~a has ~a values.~%Do you really want to graph them?",
            name(att),length(att-value)));
          (if yes-or-no-menu("Graph Them?") then
            (enumerate v over att-value do show-sn-linkage(att, obj,v, dw,
              only-if-visible?))))
        else (enumerate y over att-value do show-sn-linkage(att, obj,v, dw,
          only-if-visible?)))
      else
        let(existing-ai: attribute-instance =
          find-attribute-instance(obj,att))
        let(existing-link: link =
          if defined?(existing-ai)
            then find-2-way-link(existing-ai, obj, att-value,dw)
            else undefined)
        (if defined?(existing-link)
          then existing-link
          else create-sn-link(att, obj, att-value,dw)))

function make-link-for-sn-object (att: re::binding,
  obj: object, s:diagram-surface)
  : ri::drawable-object =
  let(attribute-instance: object: attribute-instance =
    find-or-create-attribute-instance(obj, att))
  let(dw: diagram-window = arb(viewports(s)))
  let (new-link =
    if agl-window?(dw)
      then make-default-agl-link()

```

```

    else default-make-link(dw))
set-link-label(new-link,
               make-sn-label-for-certainty(attribute-instance-object));
links-for-spec-object(attribute-instance-object) <-
links-for-spec-object(attribute-instance-object) with new-link;
set-link-surface(new-link,s);
new-link

```

```

function make-sn-label-for-certainty (ai: attribute-instance): seq(string)
=
let(ai-target: re::universe = attribute-instance-target(ai),
    ai-attribute: re::binding = attribute-instance-attribute(ai))
let(ai-certainty: any-type =
    cd::retrieve-attribute-facet(ai-target,ai-attribute,'cd::certainty))
(if undefined?(ai-certainty) then ai-certainty <- 1);
make-label(format(false,"~a: ~a", name(ai-attribute), ai-certainty))

```

```

function create-sn-link (att: object, obj: object, att-value:object,
                       dw:diagram-window) =
with-screen-updates-disabled(dw,
let(surf: diagram-surface = surface-viewed(dw))
let(source-icon: icon = find-or-create-sn-icon(obj,surf,true),
    target-icon: icon = find-or-create-sn-icon(att-value,surf,true),
    sn-link: link = make-link-for-sn-object(att,obj,surf))
(if ~agl-window?(dw) then
dynamic?(sn-link) <- true);
set-cd-target-arrow(sn-link,true);
set-cd-source(sn-link,source-icon);
set-cd-target(sn-link,target-icon);
(if agl-window?(dw) then
create-arc-link(sn-link,dw)));
refresh-window(dw)

```

```

function Make-Label-for-sn-icon (tn: object): seq(string)
=
let(obj-class: re::binding = instance-of(tn))
make-label(format(false,"~a: ~a", name(obj-class),
                 obj-name(tn)))

```

%% IBIS nodes are boxes, assumptions are diamonds, others are rectangles.

```

function find-shape-for-sn-icon (obj: object) :symbol =
if remap-position(obj) then 'ri::box
elseif remap-issue(obj) then 'ri::box
elseif remap-argument(obj) then 'ri::box
elseif remap-assumption(obj) then 'ri::diamond
else 'ri::ellipse

```

```

function find-color-for-sn-icon (obj: object) :symbol =
let(cert: real = node-certainty(obj))
if remap-position(obj) then cw::magenta
elseif remap-argument(obj) then cw::turquoise
elseif remap-issue(obj) then cw::blue
elseif remap-requirement(obj) then cw::black
elseif remap-decision(obj) then cw::magenta
elseif remap-constraint(obj) then cw::turquoise
elseif remap-design-object(obj) then cw::blue
elseif remap-assumption(obj) then
    if cert <= 1.0 then cw::red
    elseif cert > 1.0 and cert <= 2.0 then yellow
    else cw::green
else cw::white

```

```

function make-icon-for-sn-object (f: object, surf: diagram-surface) :icon =
  let (new-icon = make-object('icon))
    size-factor(new-icon) <- 1.0;
    height-width-ratio(new-icon) <- .7;
    icon-type(new-icon) <- find-shape-for-sn-icon(f);
    label(new-icon) <- Make-Label-for-sn-icon(f);
    icon-color(new-icon) <- find-color-for-sn-icon(f);
    icons-for-spec-object(f) <-
    icons-for-spec-object(f) with new-icon;
    home-surface(new-icon) <- surf;
    new-icon

function graph-basic-sn-model(sel-objs: set(object), rel-atts: set(object),
  title: string) =
  let (dw: diagram-window = make-object('diagram-window),
    surf: diagram-surface = make-object('diagram-surface)
    )
    (if *use-color* then
      do-color(native-window(dw), cda::turquoise, cda::black));
  presentation-type(dw) <- 'sn-graph;
  add-colored-icons-to-window(dw);
  sn-window-related-attributes(surf) <- rel-atts;
  diagram-window-spec-module(dw) <- owned-by(arb(sel-objs));
  with-screen-updates-disabled(dw,
    (window-title(dw) <- title;
    window-region(dw) <- default-output-region();
    surface-viewed(dw) <- surf;
    mouse-handler(dw) <- 'sn-diagram-mouse-handler;
    window-mouse-handler(dw) <- 'bring-forward;
    setup-agl-window(dw);
    agl-window?(dw) <- *use-agl?*;
    make-sn-icons(sel-objs, rel-atts, surf));
    show-all-links-in-view(dw, surf));
  view-surface(dw, surf);
  expose-window(dw)

function graph-remap-model (objs: set(re::universe), title-string: string) =
  graph-basic-sn-model(objs,
    find-attributes-or-complain({'decision-modifies-requirement,
      'assumption-certainty,
      'node-certainty,
      'requirement-generalizes-requirement,
      'requirement-specializes-requirement,
      'requirement-generates-issue,
      'requirement-leads-to-decision,
      'issue-generalizes-issue, 'issue-specializes-issue,
      'issue-replaces-issue,
      'issue-questions-issue, 'issue-suggested-by-issue,
      'issue-suggested-by-position,
      'issue-suggested-by-argument, 'position-responds-to-issue,
      'argument-supports-position, 'argument-objects-to-position,
      'argument-depends-on-assumption,
      'assumption-qualifies-argument, 'decision-resolves-issue,
      'decision-selects-position,
      'decision-generalizes-decision, 'decision-specializes-decision,
      'decision-depends-on-decision,
      'decision-implies-constraint, 'decision-leads-to-constraint,
      'decision-generates-constraint,
      'constraint-creates-design-object, 'constraint-removes-design-object,
      'constraint-modifies-design-object,
      'design-object-depends-on-constraint}),
    title-string);

```

```
let (dw: diagram-window = make-object('diagram-window),
    surf: diagram-surface = make-object('diagram-surface)
    )
add-colored-icons-to-window(dw)
```

```
function graph-remap-model-for-module (kbm: kb-module) =

  let(remap-nodes: set(remap-node) = {x|(x) x in owned-objects(kbm)
                                     & remap-node(x)})
  let(kbm-remap-issues: set(remap-issue) = filter(remap-issue, remap-nodes))
  let(kbm-remap-decisions: set(remap-decision) = filter(remap-decision,
                                                         remap-nodes))
  let(kbm-remap-constraints: set(remap-constraint) = filter(remap-constraint,
                                                             remap-nodes))

  let(kbm-remap-design-objects: set(remap-design-object) =
      filter(remap-design-object, remap-nodes))

  let(kbm-remap-assumptions: set(remap-assumption) = filter(remap-assumption,
                                                             remap-nodes))

  let(kbm-remap-positions: set(remap-position) =
      filter(remap-position, remap-nodes))
  let(kbm-remap-arguments: set(remap-argument) =
      filter(remap-argument, remap-nodes))
  let(kbm-remap-requirements: set(remap-requirement) =
      filter(remap-requirement, remap-nodes))
```

```
graph-remap-model (remap-nodes,
                  format(false, "REMAP Model for KB-module: ~a",
                          name(kbm)))
```

```
function contains-remap-nodes? (button: symbol, kbm: object,
                                pos: point, w: window) :boolean =
  ex(x)(x in owned-objects(kbm) & remap-node(x))
```

```
function invoke-graph-remap-model-for-module (button: symbol, thing: object,
                                                pos: point, w: window) = graph-remap-model-for-module(thing)
```

LIST OF REFERENCES

1. Baya, V., et al., "An Experimental Study Of Design Information Reuse," *PROCEEDINGS OF THE DESIGN THEORY AND METHODOLOGY CONFERENCE*, p. 141, 1992.
2. Endoso, J., "House Takes Tough Stance on DoD Budget." *Government Computer News*, 20 July 1992.
3. Martin, J., and McClure, C., *Software Maintenance: The Problem and its Solution.*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.
4. Ramesh, B., Dhar, V., "Supporting System Development By Capturing Deliberations During Requirements Engineering," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, Vol. 18, No. 6, p. 498, June 1992.
5. Green, C., et al, "Report on a Knowledge-Based Software Assistant," Rome Air Development Center report RADC-TR-83-195, August 1983.
6. White, D.A., "The Knowledge-Based Software Assistant: A Program Overview," *PROCEEDINGS OF THE SIXTH ANNUAL KNOWLEDGE-BASED SOFTWARE ENGINEERING CONFERENCE*, p. 2, August 1991.
7. Rome Air Development Center, *An Overview of RADC's Knowledge Based Software Assistant Program*, by D.M. Elefante, p. 8, 1988.
8. Debellis, M., "The Concept Demonstration Rapid Prototype System", *PROCEEDINGS OF THE FIFTH CONFERENCE ON KBSA*, p. 222, 1990.
9. White, op. cit., p. 3.
10. Jullig, R., et al., " KBSA Project Management Assistant," Rome Air Development Center report RADC-TR-87-78, July 1987.
11. Elefante, loc. cit., p. 8.
12. White, op. cit., p. 3.
13. White, op. cit., p. 4.
14. Goldberg, A., et al., "KBSA Performance Estimation Assistant Program Specification," Documentation, RADC Contract F30602-86-C-0026.

15. White, op.cit., p. 5.
16. White, op. cit., pp. 3-4.
17. Elefante, op. cit., p. 14.
18. Rome Air Development Center Report RADC-TR-90-349, *KBSA Framework*, by Larson, A., et al., December 1990.
19. Johnson, et al., "The Knowledge-Based Specification Assistant, Final Report", Rome Air Development Center, Contract No. F30602-85-C-0221, 1988.
20. White, op. cit., p. 5.
21. Markosian, L., Abrai'do-Fandino, L., and Katzman, S., *Modern Software Engineering Foundations and Current Prospectives*, p. 478, VonNostrand Reinhold, 1990.
22. Markosian, op. cit., pp. 450-452.
23. DeBellis, M., "The Concept Demonstration Rapid Prototype System," PROCEEDINGS OF THE FIFTH CONFERENCE ON KNOWLEDGE-BASED SOFTWARE ASSISTANT, pp. 211-212, September 1990.
24. Baya, V., et al. op. cit., p. 142.
25. Ramesh, B., Dhar, V., "Supporting System Development by Capturing Deliberations During Requirements Engineering". IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. Vol. 18, No.6, June 1992.
26. Rittel, H., "Dilemmas in a General Theory of Planning", *Policy Science*, vol. 4, pp. 155-169, 1973.
27. Andersen Consulting. *Software User's Manual: KNOWLEDGE-BASED SOFTWARE ASSISTANT CONCEPT DEMONSTRATION SYSTEM*, Chicago, January 1993.
28. Reasoning Systems. *REFINE USER'S GUIDE*. Palo Alto, Ca. 1990.
29. Reasoning Systems. op. cit., p. 3-203.
30. Reasoning Systems. op. cit., p. 3-15.
31. Reasoning Systems. op. cit., p. 3-113.

DISTRIBUTION LIST

- | | | |
|----|--|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 0142
Naval Postgraduate School
Monterey, California 93943-5002 | 2 |
| 3. | B. Ramesh, Code AS/RA
Naval Postgraduate School
Monterey, California 93943-5002 | 5 |
| 4. | Roger Stemp, Code CS/SP
Naval Postgraduate School
Monterey, California 93943-5002 | 1 |
| 5. | Lieutenant Commander Frank Hughes
809 First Street
Watervliet, New York 12189 | 2 |
| 6. | Captain Steven C. Kendall
2538 Eisenhower Avenue
Ames, Iowa 50010 | 2 |
| 7. | Director, Training and Education
MCCDC, Code C46
1019 Elliot Road
Quantico, VA 22134-5027 | 1 |