



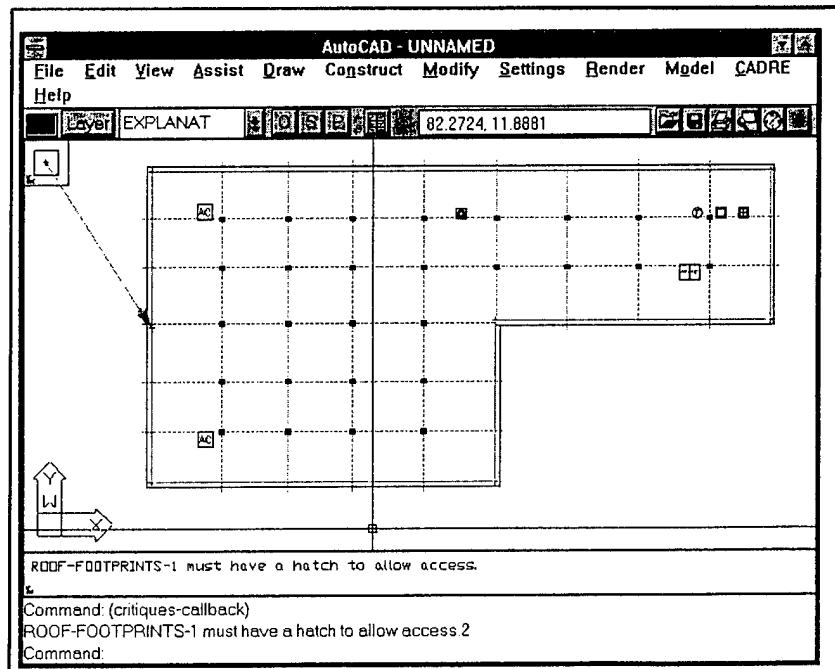
US Army Corps
of Engineers

Construction Engineering
Research Laboratories

USACERL Technical Report 97/37
April 1997

The SEDAR Reuse Libraries

by
Michael C. Fu, Jonathan E. Dapin, and E. William East



19970701 075

The Support Environment for Design And Review (SEDAR) System is an expert critiquing system for flat and low-slope roof design developed at the U.S. Army Construction Engineering Research Laboratories. SEDAR uses a task-based model of design for flexible control of its multi-strategy critiquing abilities. It is designed to support the existing design and review protocol for roof design for the U.S. Army Corps of Engineers.

This report describes reusable components of SEDAR. The components are: the expert critiquing shell, the flat and low-slope roof design domain knowledge base, a set of two-dimensional geometric reasoning routines, and a set of AutoCAD™ functions for information display. Each component's structure is described in detail, and necessary modifications for effective reuse are discussed. The appendices to this report contain file specifications and an index of the functions, rules, and rule sets of SEDAR.

DTIC QUALITY INSPECTED 3

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED

DO NOT RETURN IT TO THE ORIGINATOR

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)

2. REPORT DATE
April 1997

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
The SEDAR Reuse Libraries

5. FUNDING NUMBERS
4A162784
AT41
AR6

6. AUTHOR(S)
Michael C. Fu, Jonathan E. Dapin, and E. William East

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Construction Engineering Research Laboratories (USACERL)
P.O. Box 9005
Champaign, IL 61826-9005

8. PERFORMING ORGANIZATION
REPORT NUMBER

TR 97/37

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Headquarters, U.S. Army Corps of Engineers (HQUSACE)
ATTN: CEMP-CE
20 Massachusetts Avenue NW.
Washington, DC 20314-1000

10. SPONSORING / MONITORING
AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Copies are available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Support Environment for Design And Review (SEDAR) System is an expert critiquing system for flat and low-slope roof design developed at the U.S. Army Construction Engineering Research Laboratories. SEDAR uses a task-based model of design for flexible control of its multi-strategy critiquing abilities. It is designed to support the existing design and review protocol for roof design for the U.S. Army Corps of Engineers.

This report describes reusable components of SEDAR. The components are: the expert critiquing shell, the flat and low-slope roof design domain knowledge base, a set of two-dimensional geometric reasoning routines, and a set of AutoCADTM functions for information display. Each component's structure is described in detail, and necessary modifications for effective reuse are discussed. The appendices to this report contain file specifications and an index of the functions, rules, and rule sets of SEDAR.

14. SUBJECT TERMS
Support Environment for Design And Review (SEDAR) roofs
design criteria knowledge based systems
expert systems computer aided design

15. NUMBER OF PAGES
128

16. PRICE CODE

17. SECURITY CLASSIFICATION
OF REPORT
Unclassified

18. SECURITY CLASSIFICATION
OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION
OF ABSTRACT
Unclassified

20. LIMITATION OF
ABSTRACT
SAR

Foreword

This study was conducted for the Directorate of Military Programs, Headquarters, U.S. Army Corps of Engineers (HQUSACE) under Project 4A162784AT41, "Military Facilities Engineering Technology"; Work Unit AR6, "Domain Knowledge Structure and Process." The technical monitors were Robert Chesi, CEMP-CE and Stan Green, CEMP-CE.

The work was performed by the Engineering Processes Division (PL-E) of the Planning and Management Laboratory (PL), U.S. Army Construction Engineering Research Laboratories (USACERL). Dr. Michael P. Case is Chief, CECER-PL-E, and L. Michael Golish is Operations Chief, CECER-PL. The USACERL technical editor was Linda L. Wheatley, Technical Information Team.

The Roof Consultants Institute (RCI) provided computer-aided design (CAD) roof symbols for the project. IBM-PC, Microsoft Windows, Goldworks III, and AutoCAD are registered trademarks of International Business Machines, Microsoft, Gold Hill Computers, and Autodesk, respectively.

COL James T. Scott is Commander and Dr. Michael J. O'Connor is Director of USACERL.

Contents

SF 298	1
Foreword	2
List of Figures	5
1 Introduction	7
Background	7
Objective	7
Approach	7
Scope	8
Mode of Technology Transfer	8
Report Organization	8
2 SEDAR Overview	10
Critiquing and Suggestion in SEDAR	10
System Architecture	13
The Designer's Task Model and Its Use	13
Evaluation and Discussion	18
3 The Expert Critiquing Shell	19
Shell Overview	19
System Operation and Information Flow Across Components	29
System Reuse	37
Conclusion	44
4 The Flat and Low-Slope Roof Knowledge Base	45
SEDAR Knowledge Base	45
Knowledge Base Reuse	45
5 Geometric Reasoning Libraries	48
Description	48
Data Structures	48
Major Functions and Their Return Values	50
6 AutoCAD Information Display Functions	52
Text Display Boxes	52
The Design Objects Dialog Box	54

References	55
Appendix A: Files and Locations	58
Appendix B: Function Listings by File	61
Appendix C: Rules and Rule Set Listings by File	92
Appendix D: Alphabetical Listing of Goldworks III Lisp Functions	109
Appendix E: Alphabetical Listing of Autolisp Functions	118
Distribution	

List of Figures

Figures

1	Example of an Error Prevention Critique	12
2	Example of an Error Detection Critique	12
3	Example of a Design Suggestion	14
4	SEDAR architecture	15
5	A portion of the Designer's Task Model for flat and low-slope roof design showing interferes-with links to the <i>Air-Handler-Layout</i> task	17
6	Detailed view of Blackboard component	20
7	A portion of the Designer's Task Model for flat and low-slope roof design with task-subtask links shown as heavy black lines	21
8	The Requirements Hierarchy	23
9	The Materials Hierarchy	24
10	The Design Object Hierarchy	25
11	Relationship between critiquing/suggestion agents and the knowledge base	27
12	The SEDAR user interface	28
13	The iterative critiquing cycle	29
14	The trigger and condition portions of a design code	41
15	The Rule Frame	43
16	Example of a text display box	52
17	SEDAR architecture	53

1 Introduction

Background

The Support Environment for Design And Review (SEDAR) System is an expert critiquing system intended to support designers and reviewers in the domain of flat and low-slope roof design. Based on the IBM-PC hardware platform and the Microsoft Windows operating system, it uses a commercial, LISP-based expert system shell (Goldworks III) and a commercial computer-assisted design (CAD) program (AutoCAD). By providing an interactive, graphical interface for roof designers and reviewers, SEDAR is intended to increase the efficiency of the design review process.

Objective

The objective of this study was to identify and describe reusable components of the SEDAR project. This effort will help future developers interested in creating expert critiquing systems for other problem domains or in creating systems using the commercial applications mentioned above.

Approach

The SEDAR project has been supported by the U.S. Army Construction Engineering Research Laboratories (USACERL) since 1994. The initial architecture for the system was created after a thorough review of state-of-the-art construction management systems and existing documents from the U.S. Army Corps of Engineers. Preliminary testing of the SEDAR project for flat and low-slope roof design was conducted from May through June 1995. The testing led to revisions in the system involving the user interface. Finally, the existing SEDAR code was documented and reorganized in preparation for this report.

Scope

The SEDAR project acts as an agent in the ACE collaborative engineering project developed at USACERL and may also act as a standalone expert critiquing system. Currently SEDAR is in a second development cycle to incorporate enhancements from the testing phase and additional planned extensions.

Mode of Technology Transfer

The code developed under the SEDAR project is documented in this report. A diskette containing the reuse library files described in this report will be available upon request. The algorithms developed under this project will also be applied to the development of modules under the Modular Design System project.

Report Organization

The first chapter of this report is a brief overview of the capabilities of SEDAR. Each of the remaining chapters describes how various components of SEDAR may be reused for future research projects. In order of largest component to smallest component they are:

1. *The expert critiquing shell* may be adapted for use in other domains besides flat and low-slope roof design. The extent of shell reuse for a domain depends on several attributes of the domain. For example, shell reuse for other architectural domains maximizes the shell reuse due to their similarities to the roof domain. Other domains may require more developer adaptation. This part of the shell is written in Goldworks III.
2. *The flat and low-slope roof knowledge base* is a partial implementation of the constructibility review criteria established in East et al. (1995). Besides its use in SEDAR, this knowledge base may also be used for other applications for the flat and low-slope roof domain. The knowledge base is written in Goldworks III rule syntax.
3. *A set of two-dimensional (2-D) geometric reasoning functions* were implemented for the expert critiquing shell, and may be used in other architectural or spatial reasoning applications.

4. A set of *Autolisp functions for information display* were also developed for the user interface of SEDAR, which was an augmented version of AutoCAD. Several of these functions may be of general interest and are reported here.

The appendices to this report contain indices of functions, rules, and rule sets for the reusable code, and information about the organization of the code.

2 SEDAR Overview

The SEDAR System helps roof designers by providing critiques and simple suggestions as the roof design progresses. By providing feedback as design decisions are made, errors may be prevented or detected early in the design process, thereby reducing or eliminating the need for extensive redesign due to these errors. SEDAR assists reviewers in checking the correctness of a design by using review knowledge stored in its knowledge base. Because the process of design review is inherently a time-consuming and resource-constrained process, SEDAR will help reviewers by providing consistent and comprehensive automated reviews of the roof design. Use of SEDAR in the existing roof design and review process will help to reduce premature roof failures caused by poor quality roof designs. Roof failures resulting from errors and misjudgments in design constitute a serious legal threat to architects, contractors, and manufacturers alike (Griffin 1982), and result in high repair and maintenance costs to building owners.

SEDAR focuses the content of its critiques and suggestions through the use of a hierarchical decomposition of the roof design task called the *Designer's Task Model* (DTM). The DTM was created from observations of how experienced roof designers divide the roof design task into interdependent subtasks associated with the layout of functional subsystems, such as the drainage or walkway systems. The DTM is used to track the progress of roof designers flexibly and provides a basis for providing relevant critiques and suggestions at appropriate times in the design process.

A prototype version of SEDAR has been implemented for personal computers running Microsoft Windows using Goldworks III, a LISP-based expert system shell, and AutoCAD, a CAD tool. The results of an evaluation of the system were that users had favorable reviews of the system, that SEDAR helped to reduce the number of design errors, and that the functional decomposition of the DTM matched the users' conception of the roof design task.

Critiquing and Suggestion in SEDAR

Three critiquing strategies and one design suggestion strategy are currently implemented in SEDAR. These strategies (*error prevention, error detection, design*

review, and *simple design suggestion*) differ in their intent, timing, and intrusiveness. The error prevention, error detection, and design suggestion strategies provide advice as the roof designer creates the roof layout. The design review critiquing strategy is intended for use by reviewers and checks user-specified roof subsystems for review criteria violations. Each of these strategies may be turned on or off by the system user at any time; this level of flexibility is provided because individual users have different backgrounds, support needs, and preferences. The critiquing and design suggestion strategies use a common knowledge base containing flat and low-slope roof constructibility review criteria taken from East et al. (1995). Currently the knowledge base consists solely of condition-action rules; this knowledge representation was chosen because of its similarity to the knowledge expressed in East et al. (1995). Each of the strategies uses the DTM to focus the content of its advice.

The Error Prevention Strategy

The intent of the error prevention critiquing strategy is to prevent errors before they occur. The critiquing strategy shows "off-limits" areas on the existing layout when a user selects a design object (i.e., roof drain, air-handling unit, walkway, etc.) from the system's object palette. For example, the error prevention strategy in Figure 1 shows the designer where not to place the selected masonry chimney design object in the roof field. Cross-hatched areas that show minimum spatial separation between the existing objects on the design and masonry chimneys are shown.

The Error Detection Strategy

The intent of the error detection critiquing strategy is to detect errors as they occur. After the designer places an object in the roof field, the new object is checked for constraint violations using a set of relevant review criteria. The user may then examine any of the graphical-textual constraint violations. Figure 2 shows a constraint violation from placing the masonry chimney object too close to an existing chimney; the minimum distances between the objects are shown as cross-hatched areas, the area of the constraint violation is delineated by a dashed rectangle, and the textual portion of the critique is shown below the drawing area.

The Design Review Strategy

The design review strategy is intended to assist reviewers in the process of checking roof designs according to established review criteria, but during the evaluation of the SEDAR prototype many designers used the design review critiquing strategy to check portions of their roof layouts. After the user selects a roof subsystem to review from a graphical/textual dialog box, the system checks the existing design for all

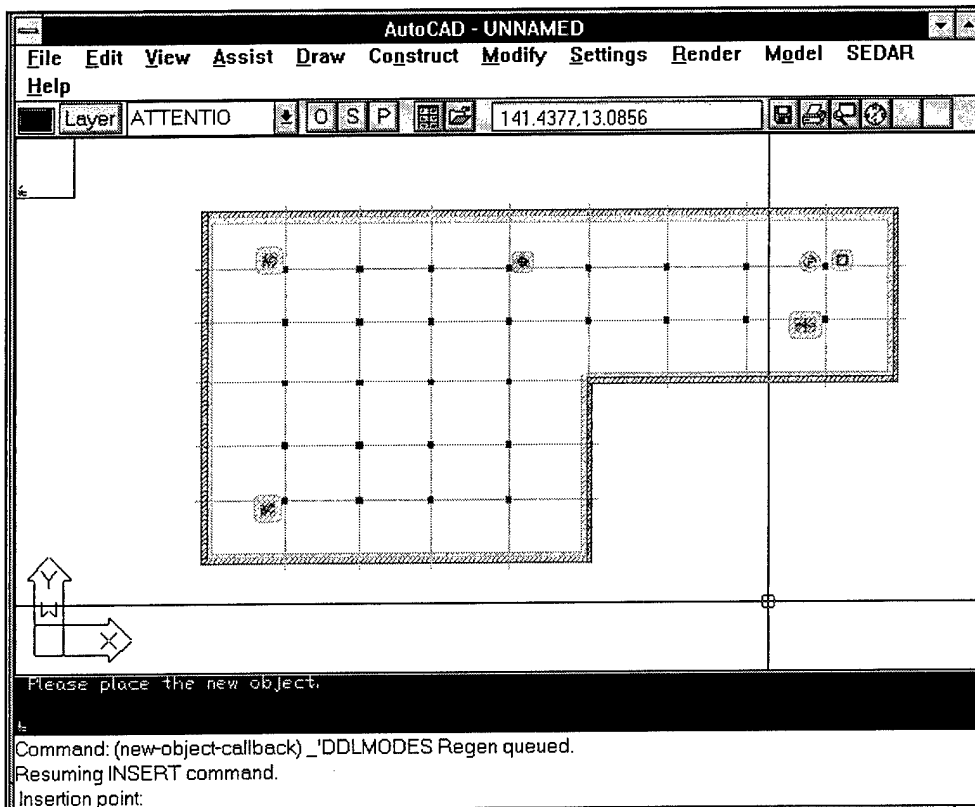


Figure 1. Example of an Error Prevention Critique.

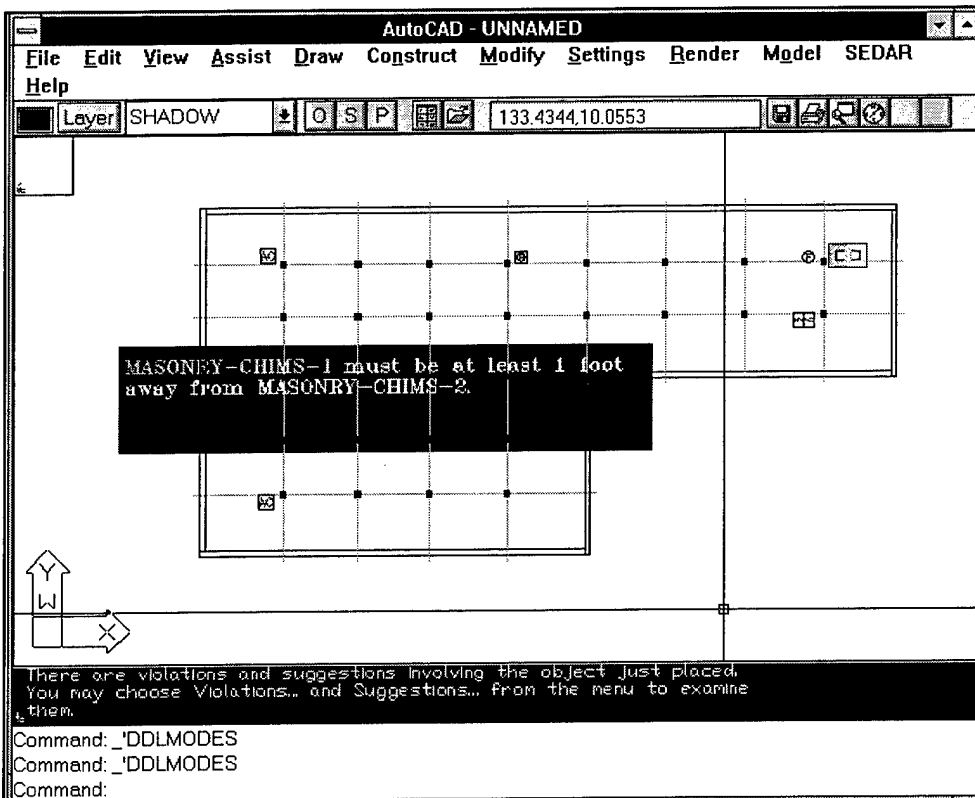


Figure 2. Example of an Error Detection Critique.

constraint violations using rules relevant to the selected subsystem. As in the error detection strategy, the user may examine the resulting graphical-textual critiques. The primary differences between the design review strategy and the error detection strategy are that (1) the design review strategy is user-activated and (2) the design review strategy checks a roof subsystem completely, while the error detection strategy checks for the legality of a single design object.

Simple Design Suggestions

Simple design suggestions are made by the system to guide a user toward a legal configuration of a roof subsystem. For example, the system will suggest the placement of an access hatch on the roof layout if no other means of accessing the roof has already been specified. Figure 3 shows the hatch design suggestion; a hatch is displayed in the upper lefthand corner of the drawing, an arrow is shown connecting the hatch to the roof, and a textual explanation is shown. In addition to these types of suggestions, SEDAR also provides a limited form of design completion. For example, when a saddle-type drainage area is placed in the roof field a roof drain is automatically placed at the low point of the saddle.

System Architecture

The architecture of SEDAR is shown in Figure 4. The *User Interface* is the communication medium between the designer and SEDAR and is an augmented version of AutoCAD. The user may add, delete, or move design objects (i.e., roof drains, air-handling units, walkways, etc.), examine the state of the DTM, view the existing critiques on the design, and turn any of the critiquing strategies on or off. User actions are communicated to the *Critic Management Agent* (CMA), which selects a critiquing strategy to apply and updates the shared data structures on the *Blackboard* (specifically, the DTM and the design representation) to reflect the modification. It then activates the appropriate *Support Strategies* (here the Critiquing and Suggestion Agents), which perform the design analysis according to the selected critiquing strategy, and translates their results into graphical/textual critiques. The critiques are then sent back to the User Interface for display.

The Designer's Task Model and Its Use

A primary contribution of this work to the field of expert critiquing systems is its use of the DTM to focus the content of its advice to issues relevant to the system user. Structurally, the DTM is a subtask hierarchy of the roof design task, consisting of

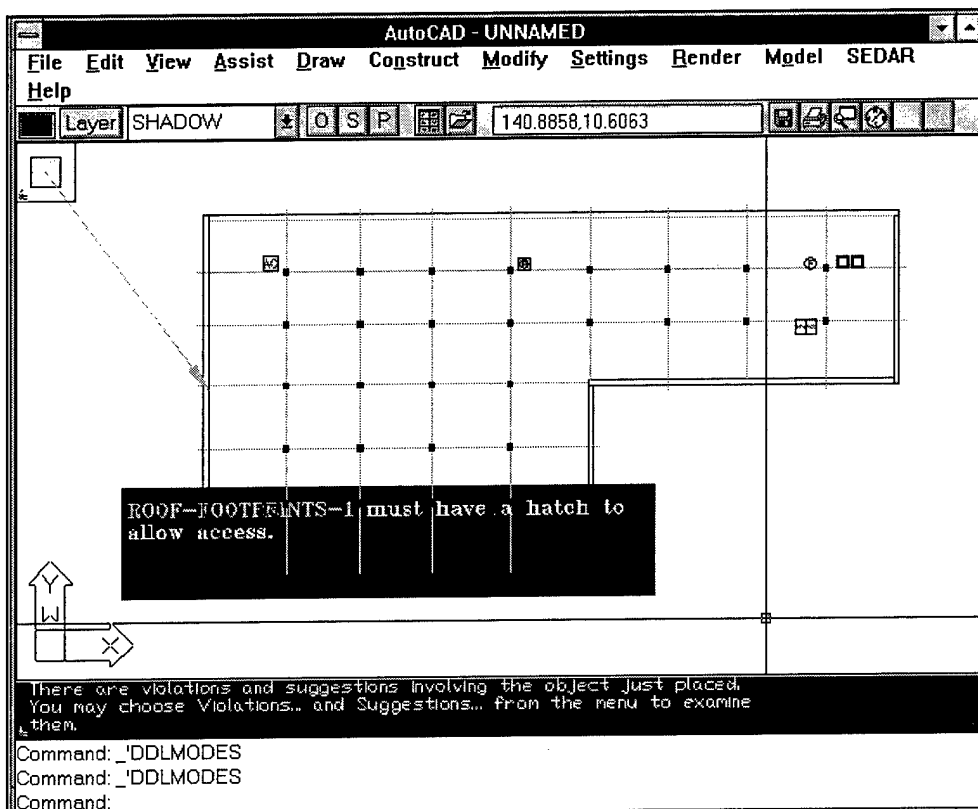


Figure 3. Example of a Design Suggestion.

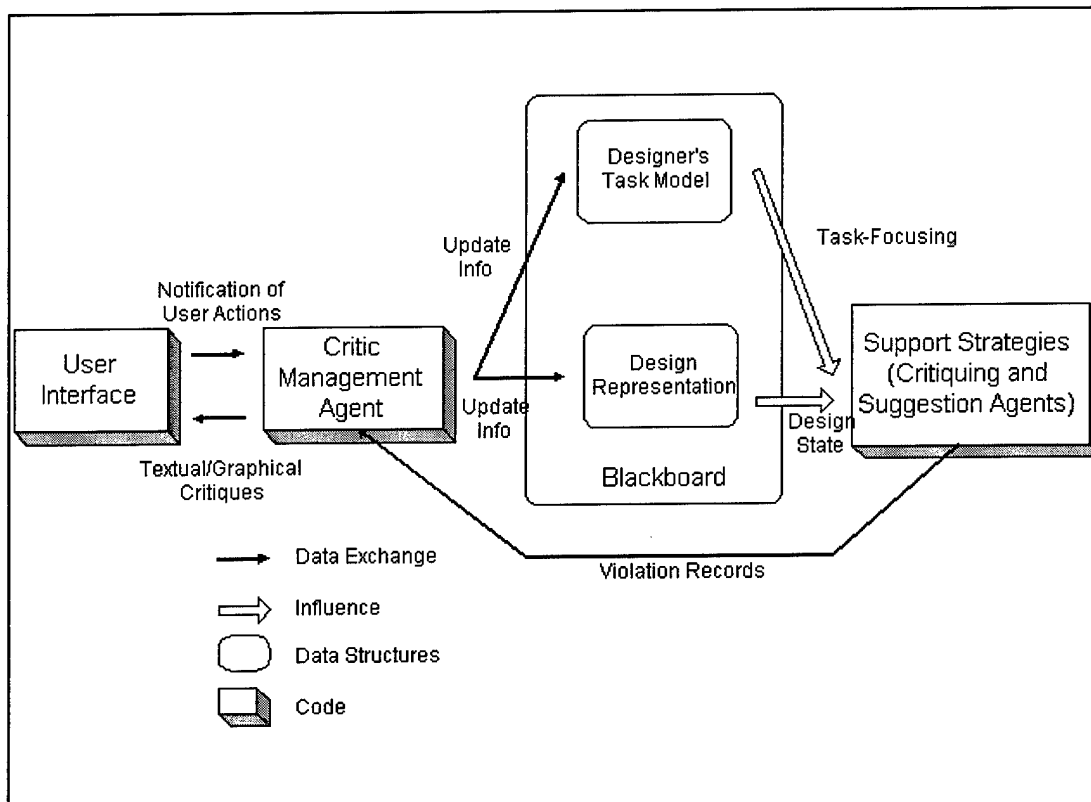


Figure 4. SEDAR architecture.

design tasks that a user may encounter during a roof design. The DTM influences system behavior in three ways: (1) it is used to track the user's progress throughout the design task, (2) the state of the DTM resulting from the tracking process determines the set of review knowledge applied to the existing roof design for each critiquing episode, and (3) the state of the DTM is used to organize the display of advice to the system user.

Structure of the DTM

Figure 5 shows a portion of the DTM; the task at the left, *Roof-Layout*, is the most abstract task. The leaf nodes of the hierarchy (i.e., *Drain-Layout*, *Walkway-Layout*, etc.) represent the design of specific functional subsystems. *Part-of* links, shown as solid lines in Figure 5, describe the task-subtask relationships. *Interferes-with* links represent possible interferences among tasks. Only the *interferes-with* links related to the *Air-Handler-Layout* task are shown in Figure 5. For example, the *Air-Handler-Layout* and *Walkway-Layout* tasks are related by an *interferes-with* link because walkways should not overlap air-conditioning units. Each subtask in the DTM is associated with a set of review criteria (in the form of condition-action rules) in the critiquing and suggestion agents specifying acceptable layout conditions.

Use of the DTM

As a designer works on the roof design, the DTM is used to track the designer's focus of attention. Each task in the DTM is either an *inactive*, *active*, or *focus* task. The set of all task states in the DTM forms an *activation pattern*. *Focus* tasks represent SEDAR's interpretation of the user's current focus. Each task is associated with a set of design objects; when a new object is added to the design, all tasks associated with the object and all of the tasks' ancestors in the part-of hierarchy are focus tasks. In Figure 5, the user's selection of a *masonry chimney* object causes the *Chimney-Layout* task and its ancestor, the *Equipment-Layout* task, to become *focus* tasks. *Active* tasks are related to the focus tasks by an *interferes-with* relation, are subtasks of a task with an *interferes-with* relation to a focus task, or were focus tasks previously. They represent tasks that not only have been addressed by the user in the past, but also those that should be considered by the user. Finally, *inactive* tasks are those that have not been addressed yet by the user.

During a critiquing episode, SEDAR uses only those review criteria that are linked with focus and active tasks so that the resulting critiques and suggestions are relevant to the user's focus of attention. In Figure 1, for example, all of the "off-limits" areas were generated from rules relevant to masonry chimneys; had the user

selected an air-handling unit instead of a chimney a different set of areas would have been shown.

Evaluation and Discussion

A prototype of SEDAR was evaluated in two experiments. The first experiment was a system usability evaluation, which rated the performance of SEDAR along various usability issues. While the full results of this experiment are reported elsewhere (Fu 1994), one outcome of this experiment was an informal verification of the appropriateness of the functional decomposition of roof subsystems of the DTM. The second experiment measured the prototype system's error reduction effectiveness, and showed that designers can use SEDAR to reduce the number of errors in their roof layouts.

The two classes of errors that the system was not able to prevent were optimality issues regarding object placement. The placement of the design object was legal according the review criteria, but the object was placed in a "suboptimal" location. Although the SEDAR prototype does not deal with the optimality of subsystem design, recognizing and advising in these situations was expressed as a need by the system evaluators for future development. Additionally ways are being sought to critique and support designers throughout the design process, from early conceptual design to later detailed design (e.g., Brown and Chandrasekaran 1986).

3 The Expert Critiquing Shell

One goal of the work on SEDAR was to develop an expert critiquing shell that can be adapted for different problem domains. The system is divided into two parts: a “domain-independent” critiquing shell and a knowledge base containing information specific to the flat and low-slope roof layout domain. The first section of this chapter divides the architecture shown in Figure 4 into the shell and knowledge base components. The second section of this chapter describes the data structures and information flow within the expert critiquing shell. The final section of this chapter discusses modification or replacement of the domain-specific knowledge base to allow critiquing in different domains.

No shell is truly completely domain independent, so SEDAR is best used for domains with certain intrinsic qualities. While these domain qualities are not essential, reuse of the shell is maximized in domains that meet many of these qualities. First and foremost, SEDAR is intended for use in domains involving “routine design.” Routine design is where the tasks and processes for solving a design task are clearly defined. The DTM of SEDAR is a representation of these tasks and processes, and a consistent model should be elicited from expert designers. Second, SEDAR is best suited for domains in which the solution is constructed from a set of atomic objects. SEDAR’s Design Object Hierarchy is a record of the types of these atomic objects. Third, SEDAR contains a library of geometric reasoning functions for use with 2-D spatial layout domains. Chapter 4 discusses this library in greater detail. Researchers who wish to use SEDAR for domains involving 2-D spatial reasoning may use the library as a foundation for their own geometric reasoning routines.

Shell Overview

The architecture shown in Figure 4 provides a component breakdown at a high level of abstraction. The critiquing shell components described in this section have been zipped using Pkzip v.2.04 into the file *sedar-sh.zip*. Of the four major components—the User Interface, the Critic Management Agent, the Blackboard, and the Critiquing Agents—two (the Blackboard and the Critiquing Agents) contain both shell and domain-specific components. The other two components (the User Interface and the CMA) may be reused in their entirety. Two commercial software applications

serve as the base for the four system components: AutoCAD (for the User Interface) and Goldworks III (for the Blackboard, Critiquing Agents, and the Critic Management Agent). The two applications communicate through a DDE interface written by the Concurrent Engineering Team at USACERL.

The Blackboard

The Blackboard contains five subcomponents that are domain-specific. Figure 6 shows a more detailed view of the blackboard and its constituent components. The Blackboard consists of five subcomponents: the DTM, a Requirements Hierarchy, a Materials Hierarchy, a Design Object Hierarchy, and the Design Representation. Each of these subcomponents may be modified to suit other problem domains; only the DTM, the Design Object Hierarchy, and the Design Representation are essential to the operation of the expert critiquing shell.

The Designer's Task Model. The DTM is a hierarchical model of the tasks involved in the problem domain. A DTM for the flat and low-slope roof design domain is shown in Figure 7. As discussed in Chapter 1, the tasks are ordered according to three types of semantic links. Task-subtask links describe the generality ordering between tasks and are shown as heavy black lines in Figure 7. Interferes-with links describe potential interferences between different tasks at the same level of

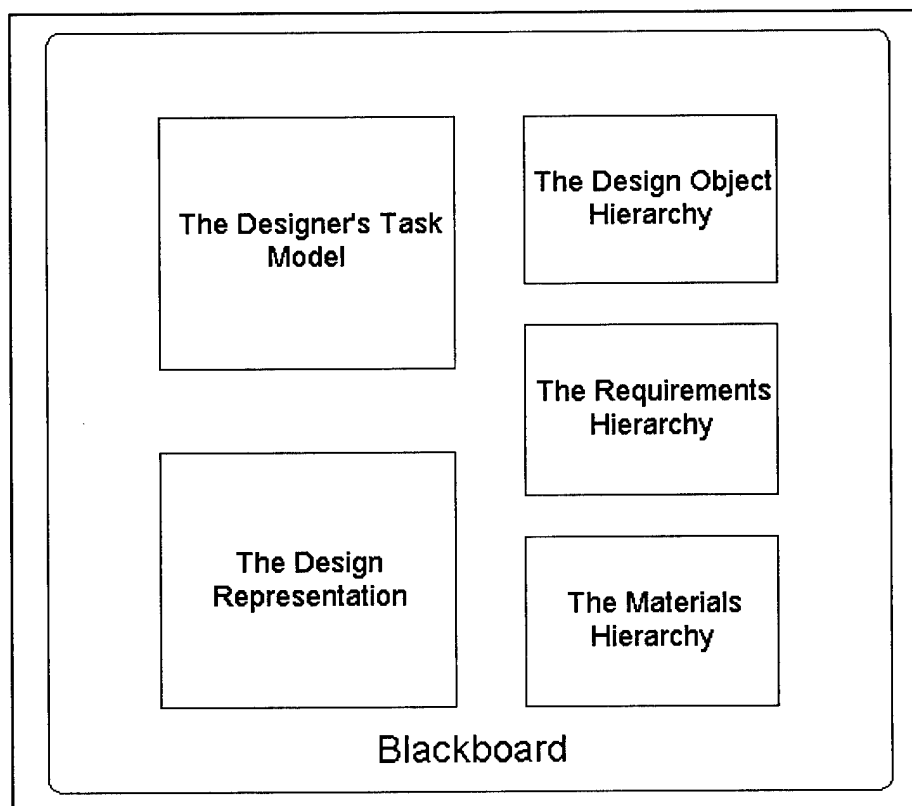


Figure 6. Detailed view of Blackboard component.

abstraction. Finally, before-task links encode orderings of task execution observed in human expert designers. Only the task-subtask and interferes-with links are used in the current version of SEDAR. The DTM is defined in two files: *frames.lsp*, which contains the task definitions and task-subtask semantic link definitions, and *assert.lsp*, which contains the definitions for the interferes-with and before-task links.

The Requirements Hierarchy. The Requirements Hierarchy is a set of goals or, in the case of design domains, a set of functional requirements that the solution must satisfy. Each goal or functional requirement is linked to a set of rules in the agent knowledge base describing conditions that satisfy (or violate) the requirement. Figure 8 depicts a portion of the Requirements Hierarchy for flat and low-slope roof design. The Requirements Hierarchy is defined in the file *frames.lsp*.

The Materials Hierarchy. Considering the interactions between materials on a roof is also important for quality roof design. For this reason, the Materials Hierarchy contains the various materials used in roofing systems (Figure 9). Individual roof components inherit not only from their parent object types but also from a material; for example, a roof deck may be made of steel, wood, or a type of concrete. Strictly speaking, however, the Materials Hierarchy is not necessary to the operation of the expert critiquing shell. Its use is an artifact of the rules in the flat and low-slope knowledge base rather than of the shell. Like the other hierarchies, the Materials Hierarchy is defined in *frames.lsp*.

The Design Object Hierarchy. The Design Object Hierarchy (Figure 10) is a hierarchical ordering of the different types of objects used to compose the solution in SEDAR. For the flat and low-slope roof design domain, this hierarchy consists of generalized design objects like roof-drains, air-handling units, saddles, and crickets. The design object frames are organized in a part-of hierarchy. The root of the tree is the abstract *physical-system-components* object. All the nonleaf nodes of the hierarchy are used as shell classes and thus are noninstantiable. The leaves of the hierarchy are the instantiable design objects (e.g., *roof-drains*, *ac-units-curbed*, and *attic-vents*). Each design object inherits from its parent in the design object hierarchy, from a set of material frames, and from a shape frame that defines the intrinsic shape of the design object. The shapes of objects are defined in greater detail in the third section of this chapter. The design object frames have slots that describe and structure the attributes associated with the type of design object represented by the frame. When the user selects and places a design object on the drawing, an instance of the generalized design object is made and its slot values filled. Like the other hierarchies, the design object hierarchy is defined in *frames.lsp*.

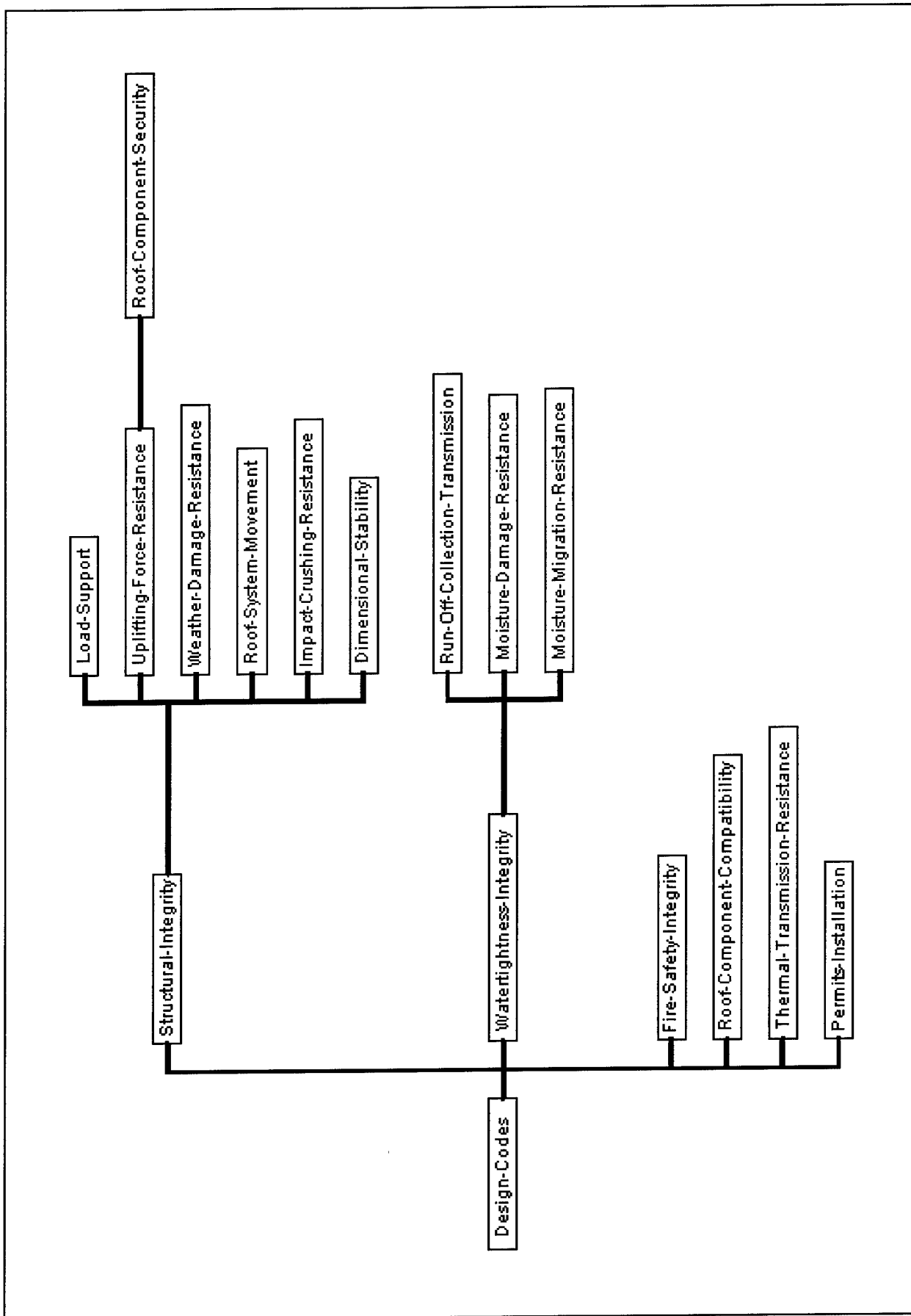


Figure 8. The Requirements Hierarchy.

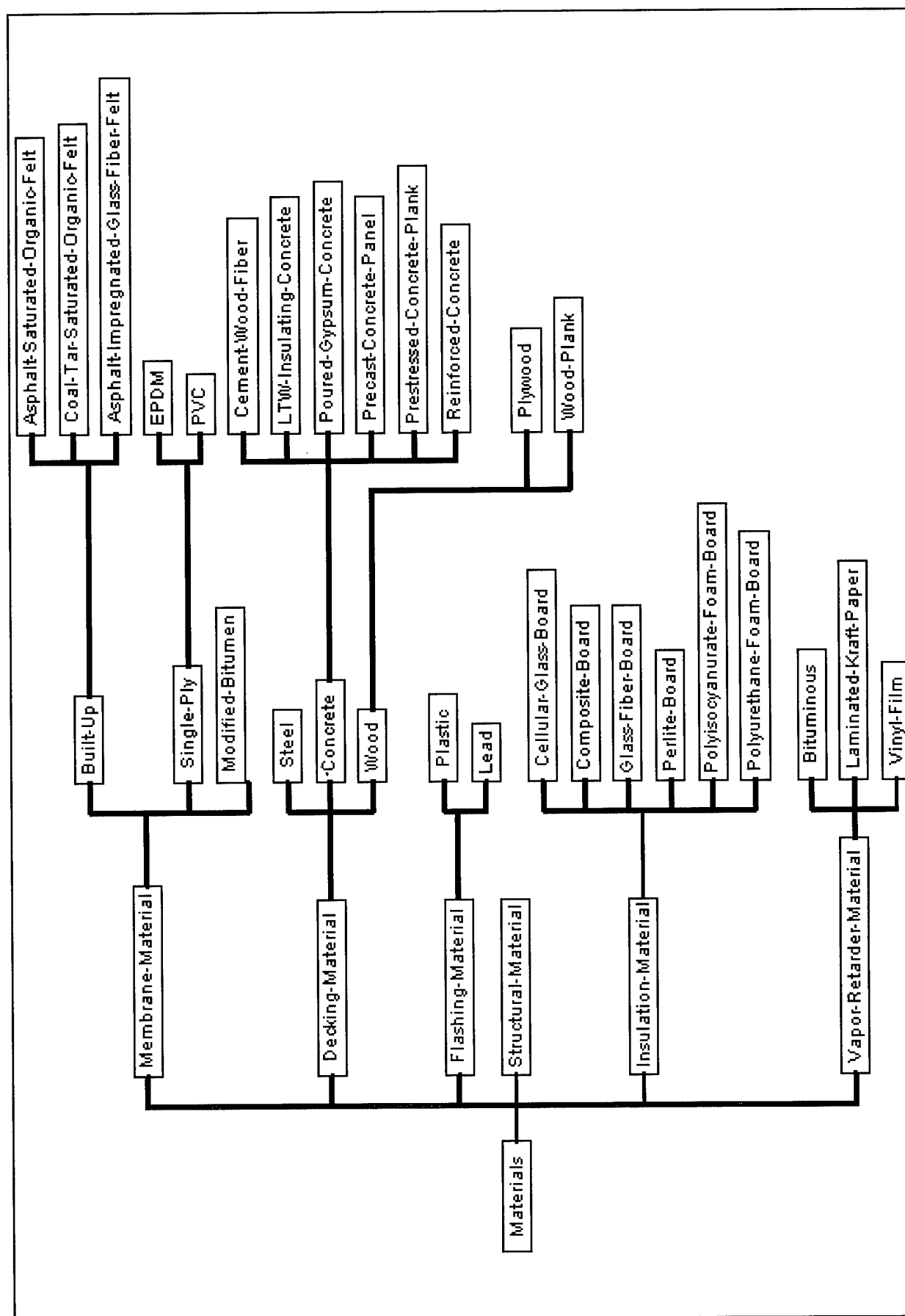


Figure 9. The Materials Hierarchy.

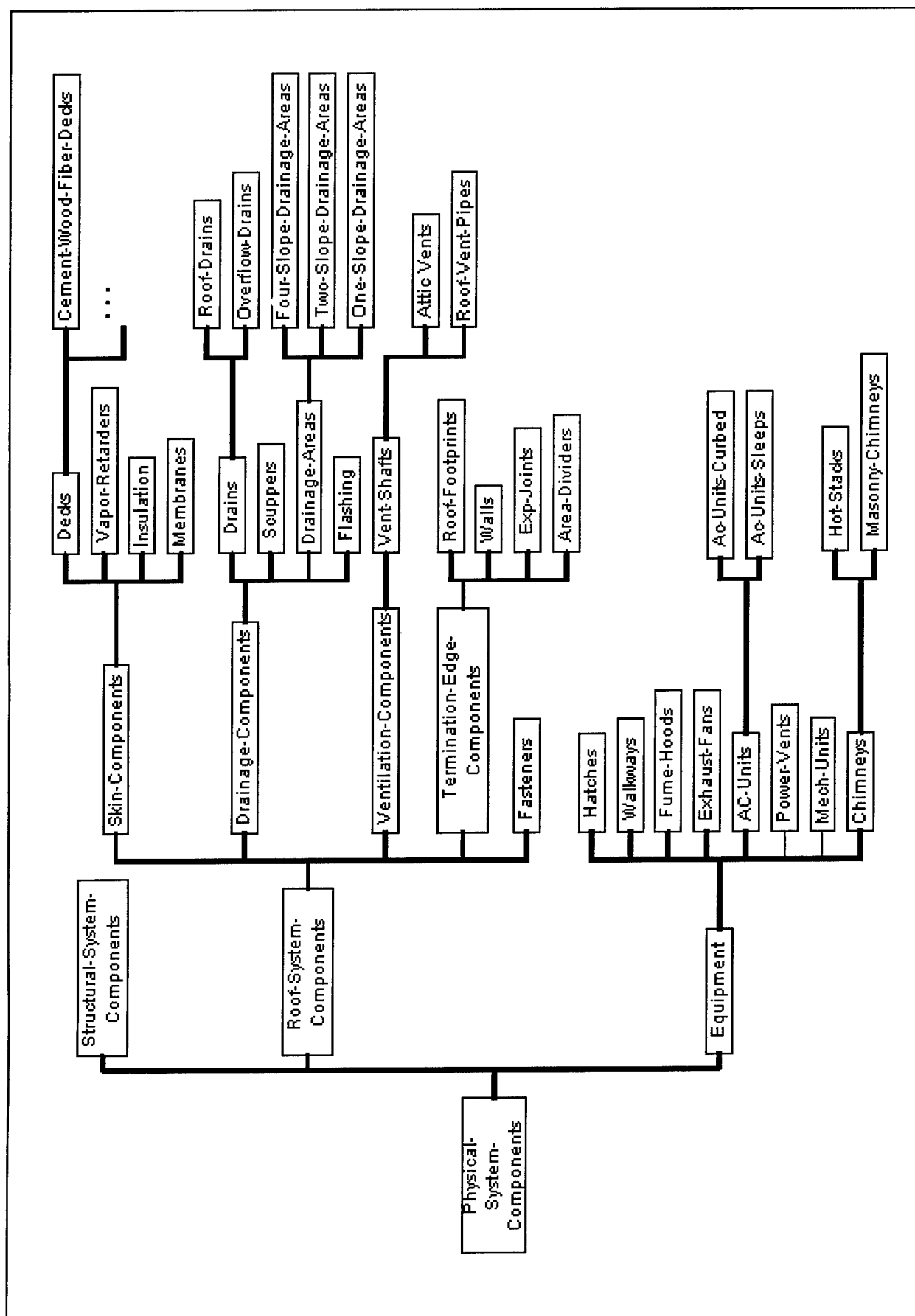


Figure 10. The Design Object Hierarchy.

The Design Representation. The Design Representation consists of object instances and semantic links between the objects. The object instances are created by the human user in the User Interface, and the semantic links are created by a set of Goldworks III rules and LISP functions attached to the generalized object definition in the Design Object Hierarchy. The rules and functions (written in Goldworks III) are automatically fired when an instance of the object is created and are defined in the files *obj-rule.lsp* and *obj-fn.lsp*.

Summary. The reuse of the blackboard is in terms of the conceptual structures required by the expert critiquing system rather than the actual content of those structures, which currently contain information for the flat and low-slope roof design domain. Of these five structures, the DTM, the Design Object Hierarchy, and the Design Representation are the most essential. The DTM is a representation of the problem-solving process of human experts and is used extensively by the Critic Management Agent and the Critiquing Agents. A cognitive task analysis and elicitation of problem-solving structure for human experts in the domain is required for proper definition of the DTM. The Design Object Hierarchy defines the set of objects which, when combined, constitutes a solution for a problem in the domain. The Design Representation encapsulates the critiquing system's representation of the solution being created by the human user. All inferencing and subsequent analysis by the Critiquing Agents is performed on the design representation. The requirements for redefinition of these three subcomponents is discussed in greater detail in the third section of this chapter.

The Critiquing Agents

SEDAR supports three distinct critiquing agents and one design suggestion agent. The critiquing agents are: the error prevention critic, the error detection critic, and the design review critic. The suggestion agent is called the simple design suggestion agent. Each of these agents use rules defined in a central knowledge base—the flat and low-slope roof design knowledge base—for the current implementation of SEDAR. The relationship between the agents and knowledge base is shown in Figure 11. The agents differ in their timing, intrusiveness, and intention for the user. The error prevention critic attempts to steer users away from anticipated error patterns before they have the chance to commit them. The error detection critic complements the error prevention strategy by checking the solution for errors concerning the rules in the flat and low-slope roof design knowledge base. Finally, the design review strategy allows the user to select various solution subcomponents to critique. In the case of roof design, solution subcomponents are roof subsystems like the drainage system design.

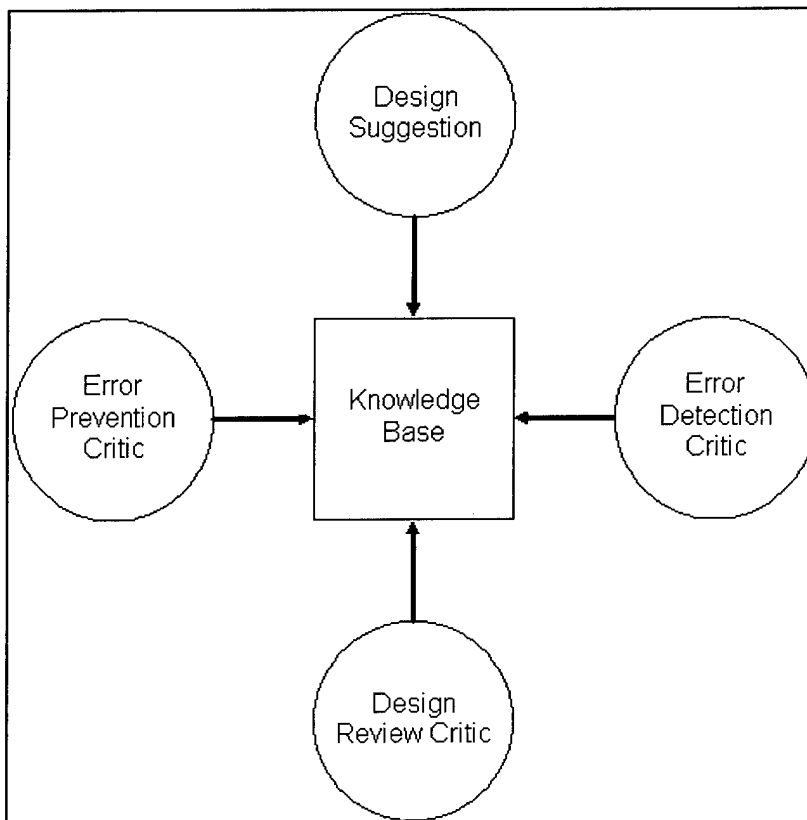


Figure 11. Relationship between critiquing/suggestion agents and the knowledge base.

The critic agents themselves are encoded in the file *cma-main.lsp* and are part of the expert critiquing shell. The knowledge base, comprised of files of Goldworks III rules in the \kb subdirectory under the gcl44\sedar directory, are specific to the roof domain only.

The User Interface

The user interface is an augmented CAD system (AutoCAD™) that allows direct manipulation of both the design and the criticism generated by SEDAR. This part of the system may also be termed as the “front-end” of SEDAR; it is the medium through which the interaction between the human designer and the critiquing system takes place. Furthermore, the user interface constitutes a powerful design environment within which the user may compose a design, control the critiquing system, and view the generated critiques. Figure 12 shows a screen capture of the SEDAR interface of a partially completed roof design and a critique generated by the system. The menu displayed in the figure is the Action Menu from which the user selects operations to perform on the design. The interface is divided into the Design, Suggestion, and Dialog windows. The large area in Figure 12 containing the top-down view of the roof design is the Design Window. Critiques generated by the system are

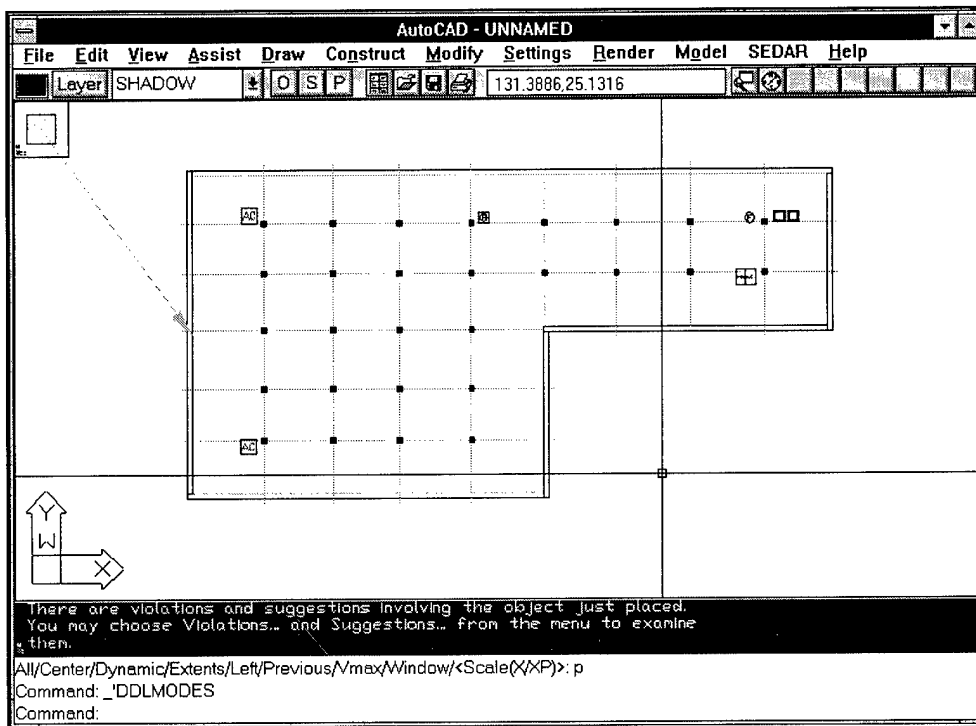


Figure 12. The SEDAR user interface.

displayed here. The small window at the upper left corner of the Design Window is the Suggestion Window. Critiques that involve design suggestions use this window in addition to the Design Window. In Figure 12, the current suggestion is that a hatch be placed on the design to allow access to the roof from below. The suggested hatch object is shown in the Suggestion Window. Finally, the Dialog Window at the bottom of the Design Window displays textual information, including prompts and the textual portions of critiques.

The code for the user interface resides in the files under the \sedar directory. Besides *.lsp files, files are available for the design objects and menus used in the user interface.

The Critic Management Agent

The CMA is the control unit of the expert critiquing system. It receives and interprets descriptions of user actions from the user interface, updates the representations on the blackboard, selects which critiquing strategies to apply, and activates the proper critic agents. The CMA selects from one of four agents: three critiquing agents (error prevention, error detection, and design review) and a simple design suggestion agent. After the critiquing process is finished, the CMA gathers the generated critiques, translates them into critique display descriptions that the user interface understands, and sends them to the user interface. The CMA operates in

a loop called the iterative critiquing cycle, which is described in the second part of this chapter. The main file containing the CMA Lisp functions is *cma-main.lsp*.

System Operation and Information Flow Across Components

System Operation: The Iterative Critiquing Cycle

SEDAR uses the *iterative critiquing cycle*, which forms the framework in which all SEDAR's actions are organized. The cycle is maintained by the CMA and has six stages, as shown in Figure 13. Each phase of the cycle is annotated with the components that are involved in its completion. This section describes the iterative critiquing cycle at a high level.

Stage 1: Receive User Input. The user selects an action to perform, such as adding, moving, deleting, or resizing existing design objects, or selecting goals for review. Depending on the selected action, the interface may query the user for additional information. The interface then sends a message to the critic management agent notifying it of the user's action and providing information that the critic management agent will need.

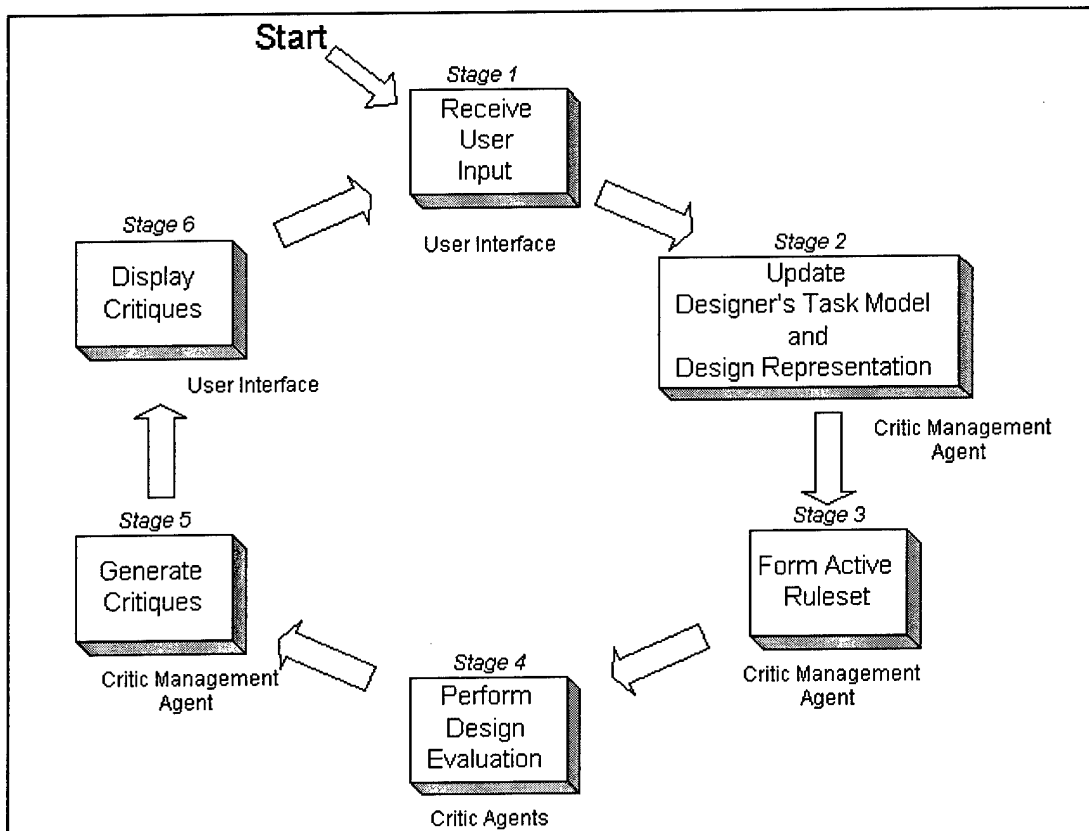


Figure 13. The iterative critiquing cycle.

Stage 2: Update the DTM and the Design Representation. Upon receiving the message from the user interface, the first task of the CMA is to update the DTM. Specifically, the CMA uses the previous DTM activation pattern and the current user action to decide which tasks in the DTM to make focus or active for the current critiquing session. This method of task activation allows for greater flexibility in the interaction between the user and the system. For example, some users may like to operate on multiple tasks simultaneously. While SEDAR does not actively enforce a particular ordering of satisfaction of its goals, it does have the capability to provide suggestions as to which tasks should be dealt with before or concurrently with the current set of tasks.

The second task for the CMA is to modify the design representation according to the user action. For example, the CMA may make a "temporary" object or a "real" object. If a "real" object is instantiated on the design representation, additional semantic links may also be created at this time to link the new design object to the previously existing objects.

Finally, the critiquing strategy is selected. Depending on the user's actions, the CMA selects from the error prevention, error correction, and design review critiquing strategies. The method of selection is static in nature.

Stage 3: Forming the Active Rulesets. During this stage, the set of design codes to be applied for the current critiquing cycle is created. All design codes are taken from the constructibility knowledge base. Only the rules linked to tasks with focus and active activations in the DTM are included in this set.

The CMA may then modify the rules in the active ruleset, depending on the critiquing strategy. This modification is done to focus the activity of the next stage on relevant objects and to improve efficiency.

Stage 4: Perform the Design Evaluation. The active set of design rules is then applied to the existing design on the blackboard. Each design code rule is a condition-action rule taken from a published handbook of low-slope roofing specifications (NRCA 1985). If the preconditions of a design code rule match a set of features in the design representation, a *design code violation* is specified with respect to those features. In every critiquing cycle, only a subset of the knowledge base of rules is applied to the design. This improves the efficiency of the design evaluation stage and, more importantly, ensures that the set of critiques and suggestions provided by the system is appropriate given the state of the design and is relevant to the user's current focus.

Stage 5: Generate Critiques. In this stage, the violation data from the previous stage are collected by the CMA and are used to generate the critiques seen by the user. An overview of this important element of the process is described here.

Critiques have separate graphical and textual portions. The CMA uses design-code specific information to create a graphical critique component in a graphical language understood by the user interface. In particular, the violation data is used to instantiate unbound variables in a stored graphical component template. The textual component generation process follows the graphical component generation. An explanation template containing unbound variables is instantiated with the violation data.

During this stage the critiques are also arranged in order of display to the user. The DTM plays an important role here; the critiques most relevant to the current focus of the user have greater priority over the rest of the critiques, which are arranged according to a serialization of the before-task partial task ordering.

Stage 6: Display Critiques. Depending on the critiquing strategy, the user interface may show the graphical/textual critiques immediately or by user request. The error prevention strategy displays all of the generated critiques on the drawing without user prompting. The error correction and design review strategies, however, simply display a notification to the user that critiques were found.

After this stage, the system loops back to Stage 1 and waits for a user action on the design. The process terminates when the user exits from SEDAR.

Known Problems. During the development of the expert critiquing system, the distinction between the iterative critiquing process and the individual critic agents was blurred due to pragmatic concerns. As a result, the task of carrying out the iterative critiquing process is split between code for the CMA and code for the individual Critic Agents. More specifically, Stage 5, which is conceptually the responsibility of the CMA, is actually performed by the Critic Agents themselves. This problem will be dealt with in future releases of the system.

Detailed System Operation and Information Flow Across Stages

Although a complete description of the system behavior is outside the scope of this report, an attempt will be made to provide the reader with a more detailed account of system activities. This account is, as in the previous section, defined in terms of the iterative critiquing cycle described at a high level above. Particular attention is given to the interactions between the expert critiquing shell and the domain-dependent portions of the system described in the first section of this chapter.

Stage 1: Receive User Input. When the user selects an entry from the Action Menu, an appropriate callback function is activated. For example, suppose the user selects **New Object...** from the Action Menu. The new-object-callback function calls a function that activates the New Object Dialog box. After the user selects a type of object, the new-object-callback function creates a unique identifier for the object and calls the CMA, passing along the user request and additional information about the object. This is accomplished by using a LISP function call to *call-gcl*. The parameters to *call-gcl* are eventually evaluated by Goldworks; hence, to activate the CMA, the initial component of the parameter to *call-gcl* is an s-expression containing a call to the top-level function of the CMA. The new-object-callback function then waits for the value returned from the CMA.

Information Transfer Between Stage 1 (User Interface) and Stage 2 (Critic Management Agent). As noted in the previous paragraph, the *call-gcl* function is called with an s-expression corresponding to an invocation of the top-level CMA function, *ac-message*. The parameters passed to the CMA within this s-expression depend on the type of request made in the user interface. The complete set of requests supported by the CMA is described in the comment for the *ac-message* function in the file *cma-main.lsp*. In this case, the user has requested a new object placement, and the s-expression resulting in the call to *ac-message* is:

```
(ac-message <query-id> "user-select-object" (<object-type> <object-id>)).
```

<query-id> is a number maintained by the system to keep track of requests and information generated by the requests on the blackboard. The string "user-select-object" identifies the type of user request. Since the user has just requested a new object (and has not yet placed the object), the only object information available is the <object-type> (e.g. roof-drains, ac-units-curbed) and the unique identifier of the object, <object-id>.

Stage 2: Update DTM and Design Representation. Upon receiving the request the *ac-message* function calls the appropriate LISP function to carry out the user request. In general, the names of these functions correspond to the user request; for example, the function called by *ac-message* given the user-select-object request is *do-select-object*.

The *do-select-object* function embodies the activities of the error prevention critic agent. It first updates the DTM according to the user request and the type of object selected by the user. All tasks directly related to the new object type are asserted as focus tasks. All focus tasks from the previous iteration and all tasks related to the new focus tasks by an interferes-with relation are asserted as active tasks. The

activations, which are asserted into the working memory, look something like the following:

(focus-task <query-id> <task-name>)
and
(active-task <query-id> <task-name> <activation-type>).

The set of these assertions record the state of the DTM for the current request. Previous focus-task and active-task assertions are not retracted from the working memory and are used as a history of DTM activations.

After updating the DTM, the *do-select-object* function makes a *shadow instance* of the selected object type in the design representation. A shadow instance is simply an instantiation of the object type without slot information (since the location of the object is not known). The shadow assertion is made so that the rules inside the knowledge base can be defined with consistent semantics. Conceptually, each rule in the knowledge base checks on a relationship between two or more design objects. Thus a shadow object is required in this case.

Stage 3: Form Active Ruleset. After performing the updates of the DTM and the design representation, the *do-select-object* function then forms the set of active rules to apply for the critiquing episode. Since this stage is within the same function as the previous stage, no information is explicitly transferred between system components. The process of forming the active ruleset is embodied in two functions: *get-active-rules*, which collects the set of rules from the knowledge base based on the state of the DTM, and *make-object-select-ruleset*, which modifies the selected rules to work with the error prevention critic. The *get-active-rules* function simply generates a union of the rules associated with focus and active DTM tasks. The set is returned as a list of rule names to *make-object-select-ruleset*. *Make-object-select-ruleset* then forms the set of active rules by modifying each rule in the selected set. Each rule is specialized to apply to the new shadow object so that the constraint information generated by the application of these rules is pertinent to not only the current state of the DTM but also the newly selected object type. After modifying the rules, the *make-object-select-ruleset* function defines a new rule set in Goldworks III containing the modified rules and deactivates it in preparation for the next stage.

Stage 4: Perform Design Evaluation. The new rule set is activated and allowed to forward chain to completion on the design representation. The result of the Perform Design Evaluation stage for the error prevention critic is a set of *check-condition*

assertions made by the active rules. These check-condition assertions have the form:

(check-condition <notification-id> <query-id> <rule-name> <variable-binding-list>).

The <notification-id> is a unique identifier for the check-condition assertion. The <query-id> is as previously defined. The <rule-name> represents the rule that created the check-condition assertion. Finally, the <variable-binding-list> records the bindings of rule variables to objects in the design representation. Since the rules were originally modified to apply to the shadow object in Stage 4, one of the elements of the <variable-binding-list> is always a binding involving the shadow object. For other critic agents this will not be the case.

Duals of check-condition assertions are removed during this stage. An example of duals is:

(check-condition CONST-AREA-1 1 RULE-6 ((“?drain-1” DRAIN-1) (“?drain-2”
DRAIN-2)))

and

(check-condition CONST-AREA-2 1 RULE-6 ((“?drain-1” DRAIN-2) (“?drain-2”
DRAIN-1))).

The primary difference between the two check-condition assertions is that the bindings of design objects to rule variables are reversed. The second check-condition assertion is eliminated.

Another issue is that of assertions resulting from rules of different *levels*. Rules in the knowledge base are separated into three categories: *physical-level*, *specification-level*, and *preference* rules. Physical-level rules check for physical impossibilities (e.g., placing a drain outside the roof field). Although these are “common-sense” rules, they of all rules are the most important. Specification-level rules are those specified in published code books. For the case of flat and low-slope roof design, specification-level rules were taken from the work (East et al. 1995) and other handbook sources (NRCA 1985). An example of a specification-level rule would be: “Drains should be placed at least 1 foot away from other drains.” Finally, preference-level rules encode individual designers’ preferences. A roof designer may like to place overflow drains close to roof drains to alleviate ponding from drains clogged by debris. Another designer may choose to use scuppers cut through the parapet wall surrounding the roof field for overflow drainage instead. The check-

condition assertions resulting from physical-level rules are given preference over specification-level rules, which are in turn given preference over preference-level rules. For the error prevention critic, all check-condition assertions are kept and passed to the next phase, but for the error detection and design review critics only the constraint violations (for a particular object) of the highest level are kept and passed to the next phase.

Stage 5: Generate Critiques. After the active rulesets are allowed to forward chain in Stage 4, the resulting check-condition assertions are collected and turned into graphical/textual critiques. Each rule has both a textual and graphical template which is used to generate the critique. The templates reference variables used within the rule. For example, RULE-21, which checks to see if a piece of equipment is accessible via walkways from the roof access mechanism, has the following two critique templates:

Textual Critique Template:

("There should be a walkway from “?e1” “ to “?e2” “.”)

Graphical Critique Template:

(MULTIPLE-DRAW

(DRAW-BOUNDARY-AREA “?e1” UNKNOWN INTERIOR 0)

(DRAW-BOUNDARY-AREA “?e2” RECTANGULAR-
COMPOSITION INTERIOR 0)).

The textual critique template consists of a list of strings. Each string may either be text (e.g., “There should be a walkway from” and “.”) or a variable (e.g., “?e1”). Variable strings have a ? as the first character, and refer to variables within the body of the rule. The graphical critique template consists of a recursive list of graphical commands for the User Interface, and also contains strings corresponding to variables in the rule body. Critique generation for each check-condition assertion from Stage 4 is a replacement of the variables within the templates with the variable bindings in the <variable-binding-list> portion of the check-condition assertion.

The generated textual and graphical portions of the critique are prepended with information about the source of the critique:

(<constraint-area-name> <rule-name> <task-name> <violation-level> <graphical-critique-portion> <textual-critique-portion>).

The <constraint-area-name> is taken from the check-condition assertion and serves as the unique identifier of the critique in both the expert critiquing shell and the

user interface. <rule-name> is the name of the rule that generated the critique. <task-name> is the name of the focus or active task associated with the rule. <violation-level> declares the level of the rule (physical-level, specification-level, or preference). Finally, <graphical-critique-portion> and <textual-critique-portion> are the components of the critique described previously. An example of an instance of this construct would be:

```
(  CONST-AREA-1
  RULE-21
  WALKWAY-LAYOUT
  SPECIFICATION-LEVEL
  (MULTIPLE-DRAW
    (DRAW-BOUNDARY-AREA AC-UNITS-1 UNKNOWN INTERIOR 0)
    (DRAW-BOUNDARY-AREA HATCHES-2 RECTANGULAR-
      COMPOSITION INTERIOR 0))
  ("There should be a walkway from AC-UNITS-1 to HATCHES-2.")
).
```

Information Transfer Between Stage 5 (CMA) and Stage 6 (User Interface). The information passed back to the waiting user interface component varies according to the user requested action. In the case of a user-select-object action, the do-select-object function returns two components in a list: the set of constraint areas resulting from Stage 5 and the set of current DTM activations. Both of these sets are represented as lists; thus the whole return value has the following form:

```
(
  (<constraint-area-1> <constraint-area-2> <constraint-area-3> ...)
  (<task-activation-1> <task-activation-2> <task-activation-3> ...)
).
```

The information passed back to the user interface differs according to the user request. All CMA functions pertaining to user requests may be found in the file *cma-main.lsp*.

Stage 6: Display Critiques. After the CMA returns the list of constraint areas and task activations to AutoCAD, the original new-object-callback function takes the set of constraint areas and proceeds from the original call to Goldworks. The display of the critiques is handled differently depending on which critic agent generated the critiques. Since the goal of the error prevention critic is to display "off-limits" situations to prevent errors from occurring, all the generated critiques are displayed immediately in the drawing area by iterating over the *draw-constraint-action*

function. In the case of the error detection critic, only a textual message notifying the user of the constraint violations are displayed; the user may then page through the critiques using additional dialog boxes.

System Reuse

This final section of the chapter describes what domain-specific components are required to use the SEDAR expert critiquing shell in other domains. In the previous section, we have discussed the necessary domain-specific components of the Blackboard (the DTM, the Design Object Hierarchy, and the Design Representation) and of the Critiquing Agents (the central knowledge base used by the critiquing and suggestion agents).

Adapting the Blackboard Components

The Designer's Task Model. The DTM should be created from protocol analyses with human experts in the problem domain. Combining expertise (e.g., forming a union of the commonly encountered tasks) is allowed because the DTM is used to track rather than guide user behavior. As such, the set of tasks in the DTM may be a superset of the tasks of any individual designer. One pitfall that must be accounted for is the possible existence of multiple fundamentally different task breakdowns for the problem domain; in this case, additional functionality to represent, select, and update multiple DTMs (each of which represents one of the different task breakdowns) is needed.

The DTM for a problem domain is defined in two files: *frames.lsp* and *assert.lsp*. The *frames.lsp* file contains Goldworks III frame definitions that represent the task-subtask semantic links among the tasks. An example frame definition is:

```
(DEFINE-FRAME DRAIN-LAYOUT
(:IS DRAINAGE-SYSTEM-LAYOUT)).
```

This statement is a definition of the *Drain-Layout* task, whose parent is the *Drainage-System-Layout* task. The *assert.lsp* file contains additional information about the DTM, including lookup knowledge about the subtree structure of the DTM, for example:

```
(goal-subtree-assoc architectural equipment-layout
(equipment-layout air-handler-layout walkway-layout chimney-layout)).
```

This goal-subtree-assoc assertion lists all of the tasks in the subtree of the *Equipment-Layout* task, including *Equipment-Layout* itself. The interferes-with semantic links are encoded as *possible-goal-interference* assertions:

(possible-goal-interference architectural equipment-layout ventilation-shaft-layout).

The possible-goal-interference assertion specifies a pair of possibly interfering tasks. In the example, the tasks are *Equipment-Layout* and *Ventilation-Shaft-Layout*; the layout of mechanical equipment on the roof (e.g., air-handling-units) may interfere with the layout of ventilation shafts.

Each task has a set of *trigger objects*. When the user selects a new object for the solution, all tasks in the DTM with a trigger object of the selected object type are activated as focus tasks. A task may have more than one trigger object. An example of a pairwise goal-object-assoc assertion defining a walkway as a trigger object for the Walkway-Layout task is:

(goal-object-assoc architectural walkway-layout walkways).

Finally, each task in the DTM is associated with a set of rules from the knowledge base for the Critiquing Agents. The union of the set of rules associated with focus and active tasks for a critiquing episode constitutes the selected set of rules for that episode. Rule 21 checks whether each piece of equipment on the roof is accessible via a walkway from the roof access mechanism. Because air-handling-units are considered equipment, the following assertion exists:

(rule-goal-assoc architectural rule21 air-handler-layout).

The Design Object Hierarchy. Like the DTM, the Design Object Hierarchy is arranged along class-subclass relations. The objects in the Design Object Hierarchy define the basic building blocks of solutions in the problem domain. Each "node" in the hierarchy constitutes a "class" of objects. When the user selects a type of object to include within the solution, an instance of the class of the selected object is created.

Each design object class has two types of slots: inherited and unique. In SEDAR, each design object class has two types of inherited slots: slots that pertain to the shape of the object and slots that pertain to the material of the object. For reuse of the geometric libraries written for the flat and low-slope roof design version of SEDAR, the shape of the object must either be a Circle or a Rectangular-Composition. All objects within the flat and low-slope roof design version of SEDAR have

one of these two shapes. These objects are described at length in Chapter 5, which discusses the reuse of the geometric reasoning libraries. The slots that pertain to the material of the object are also domain-specific and may not be needed for other problem domains. In general, design objects may have any number of inherited slots. Besides the inherited slots, each class of design object may have its own set of unique slots that describe features specific to the class. An example of the set of unique slots for the class of expansion joints on a roof is:

```
(DEFINE-FRAME Exp-joints
  (:IS (TERMINATION-EDGE-COMPONENTS RECTANGULAR-COMPOSITION))
  (ENDPOINT1 :DEFAULT-VALUES (NIL))
  (ENDPOINT2 :DEFAULT-VALUES (NIL))
  (WIDTH :CONSTRAINTS (:LISP-TYPE NUMBER))
  (user-modifiable-slots :default-values ((endpoint1 endpoint2 width)))
  (activate-when-created-ruleset :default-values (expansion-joint-ruleset))
  (activate-when-created-functions :default-values ((complete-expansion-joint-slots)))).
```

The name of the class is *Exp-joints*. The second line defines the direct ancestors of the *Exp-joints* class; it is a form of *Termination-Edge-Component* and inherits shape slots from the *Rectangular-composition* class of shapes. The *Exp-joints* class has three unique slots: *Endpoint1*, *Endpoint2*, and *Width*. The final three lines of the *Exp-joints* definition contain more information about the class. The *User-modifiable-slots* field contains a list of the slots that may be altered by the user. The *Activate-when-created-ruleset* and *Activate-when-created-functions* fields contain lists of rulesets and/or functions that act when a new instance of the object class is created. For example, when a new expansion joint is created by the user, the *expansion-joint-ruleset* will fire, and the *complete-expansion-joint-slots* LISP function will be called with the name of the new expansion joint. These rulesets and functions are located in the files *obj-rule.lsp* and *obj-fn.lsp*.

Finally, each new object type that is created should result in new assertions in *assert.lsp*:

- *goal-object-assoc* assertions that link tasks to their trigger objects
- *rule-object-assoc* assertions that link rules in the knowledge base to object classes.

The Design Representation. The design representation consists of a set of object instances and a set of semantic links among the object instances. These two sets are highly domain-dependent and are closely linked to the rules in the knowledge base; the rules in the knowledge base may look for certain types of semantic links between

object instances. The semantic links may be general spatial relation links (e.g. *distance-greater-than*, *area-enclosed-within*) or they may be more specific. A set of general spatial relation links are provided by the geometric reasoning library discussed in Chapter 5. Domain-dependent semantic links are often defined in the files *obj-rule.lsp* and *obj-fn.lsp*, which contain the rulesets and functions called automatically when an object is created.

Adapting the Critiquing Agent Knowledge Base

Each rule in the knowledge base has three parts: *trigger*, *condition*, and *rule information*. The condition-action nature of each rule was captured in the trigger and condition portions, which are themselves expressed in a condition-action form using the Goldworks III rule syntax. The trigger portion of the design code is used to check the solution for the basic applicability of the rule. This involves checking the solution for the correct types of objects and whether or not the particular set of objects has ever been checked before. If the basic applicability conditions are satisfied, the *condition* portion of the rule is invoked. The condition portion usually involves the calculation of a relationship between the two objects, and is generally more expensive to apply than the design code trigger. If the condition portion is satisfied, a note is made of the violation and a critique is generated. The trigger and condition portions of Rule 21 is in Figure 14.

Both the trigger and condition portions are expressed as if-then rules. The antecedent of the trigger portion is a conjunction of conditions. The first two conditions establish the type of objects (here any type of equipment and a hatch) and bind instantiated design objects to the variables (?e1 and ?e2). The third condition (not-equal ?e1 ?e2) ensures that ?e1 and ?e2 are not the same object. The last condition of the trigger checks to see if the rule has been checked previously and found not to be in violation. If it has been checked, then there is no reason to continue with the current rule check. The record of previously checked rules is updated when design objects are moved, resized, or deleted; clearly, if a design object has been modified, then the previous rule checks are no longer valid.

The consequent of the trigger portion asserts a message (a check-condition assertion) for the condition portion of the design code. In particular, it establishes an identification tag for the rule check and the variable bindings for the check. In the case of a user select object request (the error prevention critic), forward chaining of the rules in the knowledge base stops at this point. However, for the error detection and design review critics, and the simple suggestion critic, the condition portion of the rule is then applied. The condition portion of the rule is not applied for the error prevention critic because the information about the shadow object (e.g., the physical

Design Code

Rule 21: Equipment on the roof should be accessible via walkways from a hatch.

Trigger Portion

```
(define-rule rule21-trigger (:priority 100)
  (instance ?e1 is equipment)
  (instance ?e2 is hatches)
  (not-equal ?e1 ?e2)
  (equal (checked-before-dual 'rule21 (list "?e1" ?e1) (list "?e2" ?e2))) ' ( ))
  THEN
  (check-condition ?new-violation-name ?current-query rule21 (( "?e1" ?e1) (" ?e2" ?e2))))
```

Condition Portion

```
(define-rule rule21-condition (:priority 0)
  (check-condition ?name ?current-query rule21 ((?t1 ?e1) (?t2 ?e2)))
  (equal (connected? ?e1 ?e2) ' ( ))
  THEN
  (violation ?name ?current-query rule21 ((?t1 ?e1) (?t2 ?e2))))
```

Figure 14. The trigger and condition portions of a design code.

location, the unique slot values) are not known at that time. This information is known when the other critics and suggestion agents are applied.

The antecedent of the condition portion performs the actual violation check between the objects. In the example, this check is performed in the line (equal (connected? ?e1 ?e2) '()). The connected? relation is implemented as a LISP function that checks to see if the two design objects are accessible via a sequence of walkways. If the relation fails, the equipment object is not accessible via the hatch and a violation message is created, to be processed in the Generate Critiques phase (Stage 5) of the iterative critiquing cycle.

The reason for splitting the trigger and condition portions is discussed in detail in Chapter 4. The *rule frame* portion corresponding to Rule 21 is shown in Figure 15. The rule frame portion contains information about the rule: the variable-object type association list, the rule level, the rule type, and critique generation information.

The variable-object type association list relates the variables in the body of the rule to legally bindable object types in the Design Object Hierarchy. For Rule 21, the variable ?eq1 should be bound to an instance of *Equipment*, and the variable ?eq2 should be bound to an instance of *Hatches*.

The rule level slot defines the level of the rule: physical-level, specification-level, or preference-level.

Each rule may either be an *object-relation* rule or an *object-existence* rule (rule type slot). *Object-relation* rules detect problems between existing objects on the design and are rules used by the critiquing agents; *object-existence* rules make suggestions for adding (or removing) objects to and from the design and hence are used by the simple design suggestion agent.

As was discussed in the second section of this chapter, the critique generation information from the *text*, *bindable-list*, *explanation*, and *violation-action* slots is used to create the graphical and textual critiques described in Chapter 4. The graphical component of the critique is generated from the contents of the violation-action slot and the textual component of the critique is generated from the contents of the explanation slot.

The knowledge base may be spread across several files. For the version of SEDAR for the flat and low-slope roof domain, the files containing the trigger and condition portions of the rules may be found in the `sedar\kb` subdirectories. Appendix A lists the specific files. When creating a new knowledge base, the knowledge base files for

Instance: RULE21
 Parent: DESIGN-CODES
 Slots:

Name	RULE21
Level	SPECIFICATION
Rule-Type	OBJECT-EXISTENCE
Permanent	T
Text	"All equipment should be accessible via walkways from the hatch."
Trigger	RULE21-TRIGGER
Condition	RULE21-CONDITION
Object-Driven	RULE21-INTERACT
Bindable-List	((?E1 EQUIPMENT) (?E2 EQUIPMENT))
Explanation	("There should be a walkway from " ?e1 " " to " ?e2 " " ")
Violation-Action	(MULTIPLE-DRAW (DRAW-BOUNDARY-AREA "?e1" UNKNOWN INTERIOR 0) (DRAW-BOUNDARY-AREA "?e2" RECTANGULAR-COMPOSITION INTERIOR 0))

Figure 15. The Rule Frame.

the flat and low-slope roof domain may be removed by altering the set of files loaded in *a.lsp*. The rule frame portions of the rules are defined with the trigger and condition portions of the rules. The semantic link assertions made in *assert.lsp* should also be updated to reflect the new set of rules in the knowledge base. Finally, the file *kb.lsp* contains a registry of all of the trigger and condition portions of rules in the knowledge base and should be updated to reflect the content of the new knowledge base.

Conclusion

Adapting SEDAR to work with new domains requires the modification of two components of the existing architecture – the domain-specific portions of the Blackboard (the DTM, the Design Object Hierarchy, and the Design Representation) and the domain-specific portions of the Critiquing Agents (the knowledge base). Altering the DTM requires a cognitive task analysis of human experts in the new problem domain. The Design Object Hierarchy defines the fundamental building blocks of solutions for the problem domain. The Design Representation, consisting of object instances and semantic links amongst the object instances, is the system's representation of the human user's partial solution. The semantic links may include links created by the 2-D geometric reasoning routines (discussed in greater detail in Chapter 5) and domain-specific semantic links. The knowledge base consists of domain rules for critiquing the human's solution and for making suggestions. The knowledge base is discussed in greater detail in Chapter 4, which discusses how to use the existing flat and low-slope roof design knowledge base for other applications.

4 The Flat and Low-Slope Roof Knowledge Base

SEDAR Knowledge Base

To date, the flat and low-slope roof design knowledge base is a partial implementation of 120 of the constructibility codes specified in East et al. (1995). While most of the major component types have been addressed in the knowledge base, not all of the codes specified were amenable for use in SEDAR. The codes used in the existing knowledge base pertained to the layout of roof components in the roof field. Some rules pertained to the construction process rather than the design of roofs. Other rules dealt with construction details, a level of specificity not supported by the current version of SEDAR. While the implementation of the codes in East et al. (1995) is incomplete, the existing implementation is believed to be an acceptable starting point for a software system.

Knowledge Base Reuse

Because the constructibility codes are defined as Goldworks III rules, researchers who wish to develop systems for flat and low-slope roofs in Goldworks III may be able to reuse SEDAR's knowledge base. To reuse SEDAR's knowledge base, four components of the current SEDAR system should be retained:

- the files containing the rules in the knowledge base
- the set of roof components defined in the Design Object Hierarchy
- the geometric reasoning libraries (found in *sedar-ge.zip*)
- the semantic links between the roof components.

The necessary components (excluding the geometric reasoning libraries) have been zipped using Pkzip v. 2.04 into the file *sedar-kb.zip*.

The files containing the rules in the knowledge base are in files under the *sedar\kb* subdirectory. These files are:

- *areadiv.lsp*

- drains.lsp
- equip.lsp
- expansio.lsp
- roof.lsp
- scuppers.lsp
- vents.lsp

The files contain the trigger and condition portions of the rules shown below:

```

1  (define-rule rule21-trigger (:priority 100)
2    (instance ?e1 is equipment)
3    (instance ?e2 is hatches)
4    (not-equal ?e1 ?e2)
5    (unknown (instance ?e1 is hatches))
6    (unknown (instance ?e1 is walkways))
7    (unknown (instance ?e2 is walkways))
8    (equal (checked-before-dual 'rule21 (list "?e1" ?e1) (list "?e2" ?e2)) '())
9    (bind ?new-violation-name (violation-name))
10   (bind ?current-query *CURRENT-QUERY*)
11  THEN
12   (check-condition ?new-violation-name ?current-query rule21 ((" ?e1" ?e1)
      (" ?e2" ?e2))))
13 (define-rule rule21-condition (:priority 0)
14   (check-condition ?name ?current-query rule21 ((?t1 ?e1) (?t2 ?e2)))
15   (equal (connected? ?e1 ?e2) '())
16  THEN
17   (retract (check-condition ?name ?current-query rule21 ((?t1 ?e1) (?t2 ?e2))))
18   (violation ?name ?current-query rule21 ((?t1 ?e1) (?t2 ?e2))))

```

The antecedent of the trigger portion of the rule contains type checking information (Lines 2 to 7), a check for a previously cached attempt to apply the rule (Line 8), and additional bindings for the unique identification (id) of the rule application attempt (Line 9) and the current query number (Line 10). The consequent of the trigger portion is a single check-condition assertion into the working memory of SEDAR (Line 12), which contains the unique id, the query number, the rule name, and the variable/object binding list. The condition portions of the rules are designed to fire only after all the trigger portions of the rules have been fired. The trigger portions of the rules are assigned a priority of 100 (Line 1), while the condition portions of the rules are assigned a priority of 0 (Line 13). This prioritization allows the system developer to “insert” rules that fire between the application of the trigger and condition portions of the rules. For example, the developer may wish to eliminate

duals of rule applications (described in Chapter 3), which may be accomplished by writing rules at intermediate levels of priority (i.e., less than 100 and greater than 0) that remove dual check-condition assertions. The first line of the antecedent (Line 14) checks for the check-condition assertion made by the trigger portion of the rule and binds the necessary variables. Line 15 contains the possibly expensive check of the relationship between the objects specified in the rule—in this case, ?e1 and ?e2 are checked to see if they are *connected?*. The *connected?* function is a domain-specific function that tries to find a path (defined by walkways) between the roof-mounted equipment bound to ?e1 and the hatch bound to ?e2. If the two components are not connected (i.e., the call to *connected?* returns nil [false]), the rule consequent is applied. In the rule consequent, the original check-condition assertion is replaced with a violation assertion containing the same information. Thus, a record is kept of rule violations (violation assertions) as well as previous rule checks of object relationships that are satisfied by the existing design (the surviving check-condition assertions).

Each of the design codes is also associated with a rule frame component. The rule frame, described in the third section of Chapter 3, contains information pertaining to the applicability of the rule and constraint templates. These templates are non-essential components with respect to reuse of the roof knowledge base, but are included for the additional reference.

To use the set of rules in Goldworks III, the reader is referred to the Goldworks III reference manual, which describes how to add these rules to a rule set and how to apply these rules by activating the rule set, calling the *forward-chain* function, and then deactivating the rule set.

5 Geometric Reasoning Libraries

Description

This component contains LISP functions for computing various quantities and properties related to the geometric positions of shapes in a 2-D Cartesian coordinate system. Two files contain geometric reasoning routines:

geometry.lsp
decomp.lsp

The geometric reasoning functions in each of these files use filtering processes to quickly eliminate obviously false solutions. Additionally, these function cache previously computed geometric relationships on the blackboard to speed up computation. These two files are included in the zipped file *sedar-ge.zip*. Finally, this library assumes that objects are represented in terms of two types of shapes: circles and rectangular-compositions, which are described below.

Data Structures

The functions in this component take *objects* as their arguments. These objects should be Goldworks instances. They must have a *shape-type* slot, and the slot-value for this must be *rectangular-composition* or *circle*. Each object must have a *coordinate-info* slot.

If the object is a circle, the *coordinate-info* slot contains the center point of the object, which is a two-element list, representing x-y coordinates. The object must also have a "radius" slot containing the radius of the circle.

If the object is a rectangular-composition, the *coordinate-info* slot contains a list of the vertices of the rectangular-composition. In addition, the object must have an *extent* slot containing a list of two points that represent the bounding box for the rectangular-composition. There must also be slots called *vertical-borders* and *horizontal-borders*, containing lists of borders. A border is a two-element list (location extent). The location of a vertical-border is its x-location, and the extent of a

vertical-border is a list of two y-coordinates. The location of a horizontal-border is its y-location, and the extent of a horizontal-border is a list of two x-coordinates.

Example Data Structures

The circle and rectangular-composition frames are:

```
(DEFINE-FRAME CIRCLE
  (:IS OBJECT-GEOMETRY)
  (COORDINATE-INFO :CONSTRAINTS (:LISP-TYPE LIST))
  (RADIUS :CONSTRAINTS NIL :DEFAULT-VALUES (0.25))
  (SHAPE-TYPE :DEFAULT-VALUES (CIRCLE)))

(DEFINE-FRAME RECTANGULAR-COMPOSITION
  (:IS OBJECT-GEOMETRY)
  (COORDINATE-INFO :DEFAULT-VALUES (NIL)
    :CONSTRAINTS (:LISP-TYPE LIST))
  (VERTICAL-BORDERS :DEFAULT-VALUES (NIL)
    :CONSTRAINTS (:LISP-TYPE LIST))
  (HORIZONTAL-BORDERS :DEFAULT-VALUES (NIL)
    :CONSTRAINTS (:LISP-TYPE LIST))
  (SHAPE-TYPE :DEFAULT-VALUES (RECTANGULAR-COMPOSITION))
  (EXTENT :DEFAULT-VALUES (NIL)
    :CONSTRAINTS (:LISP-TYPE LIST)))
```

A portion of an instance of the circle frame is:

```
(:IS ATTIC-VENTS)
(COORDINATE-INFO (62.7 36.2))
(RADIUS 0.25)
(SHAPE-TYPE CIRCLE)
```

A portion of an instance of the rectangular-composition frame is:

```
(:IS AC-UNITS-CURBED)
(COORDINATE-INFO ((86.3 62.7) (89.3 62.7) (89.3 59.7) (86.3 59.7)))
(VERTICAL-BORDERS ((86.3 (59.7 62.7)) (89.3 (59.7 62.7))))
(HORIZONTAL-BORDERS ((59.7 (86.3 89.3)) (62.7 (86.3 89.3))))
(SHAPE-TYPE RECTANGULAR-COMPOSITION)
(EXTENT ((86.3 59.7) (89.3 62.7)))
```

Major Functions and Their Return Values

In geometry.lsp:

- (compute-distance object1 object2)* - Given two objects, find the minimum distance between the objects.
- (complete-overlap object1 object2)* - Returns 't if object2 is completely contained within object1. Returns nil if not.
- (no-overlap object1 object2)* - Returns 't if object1 and object2 have no overlap except possibly on a point or a line. Returns nil otherwise.
- (adjacent object1 object2)* - Returns 't if object1 touches object2. Returns nil otherwise.
- (intersection object1 object2)* - Returns 't if object1 intersects object2, nil otherwise. This function is the opposite of no-overlap.
- (aligned object1 object2 tolerance)* - Given two adjacent objects, returns 't if one of their edges is aligned within the given tolerance. Returns nil otherwise.
- (next-to-outside object1 object2)* - Returns 't if object1 is next to object2 on the outside. Returns nil if not.
- (next-to-inside object1 object2)* - Returns 't if object2 is completely contained within object1 and is next to object1. Returns nil otherwise.
- (area-of object)* - Given an object, lookup or compute its area and return it.
- (compute-distance object1 object2)* - Computes and returns the distance between two objects.
- (north-of rect1 rect2)*
- (south-of rect1 rect2)*
- (east-of rect1 rect2)*
- (west-of rect1 rect2)* - Given two rectangular areas (simple rectangular areas, not complex rectangular composition), return 't if the desired relative positions are true. Returns nil otherwise.
- (exceeds-max-distance-p obj obj2 maxd)* - This function is intended to quickly check if the distance between two objects exceeds maxd. Note that if this function returns T, the two objects are definitely more than maxd apart. However, if this function returns nil, the objects might still be more than maxd apart. The purpose of this function is to quickly filter out pairs of objects that are far apart.

In *decomp.lsp*:

(maximum-decomposition rect-obj) - Takes as argument the name of a rectangular-composition. Returns a list of the maximal set of simple rectangular regions making up the rectangular-composition.

(horizontal-decomposition rect-obj) - Takes as argument the name of a rectangular composition. Returns a list of the set of horizontal slices of the rectangular composition. Each slice is a simple rectangular region.

(vertical-decomposition rect-obj) - Takes as argument the name of a rectangular composition. Returns a list of the set of vertical slices of the rectangular composition. Each slice is a simple rectangular region.

(subtract-area start-list subtract-list) - Takes as arguments two lists of rectangular extents. Geometrically “subtracts” the extents in *subtract-list* from *start-list* and returns what is left. More precisely, the areas of overlap between *start-list* and *subtract-list* are removed from *start-list* and the remainder is returned as a list of rectangular extents (or possibly an empty list if nothing is left).

Besides these major functions, numerous supporting functions have also been written for the geometric reasoning library, and are contained in the files *geometry.lsp* and *decomp.lsp*.

6 AutoCAD Information Display Functions

Besides the reuse of the user interface in the context of the expert critiquing shell, two aspects of the interface may be reused by interface developers working within AutoCAD. The first reuse component is that of the text display boxes used to display the textual portions of critiques in the AutoCAD drawing screen (Figure 16). The second reuse component is the design objects dialog box used to select an object from a palette (Figure 17). Each of these components is described below and included in the file *sedar-ac.zip*.

Text Display Boxes

File: *ac-expl.lsp* — This component contains AutoLISP functions for displaying textual explanations in a solid rectangle overlaying an AutoCAD design. The explanation box may be temporarily displayed and then erased without affecting the rest

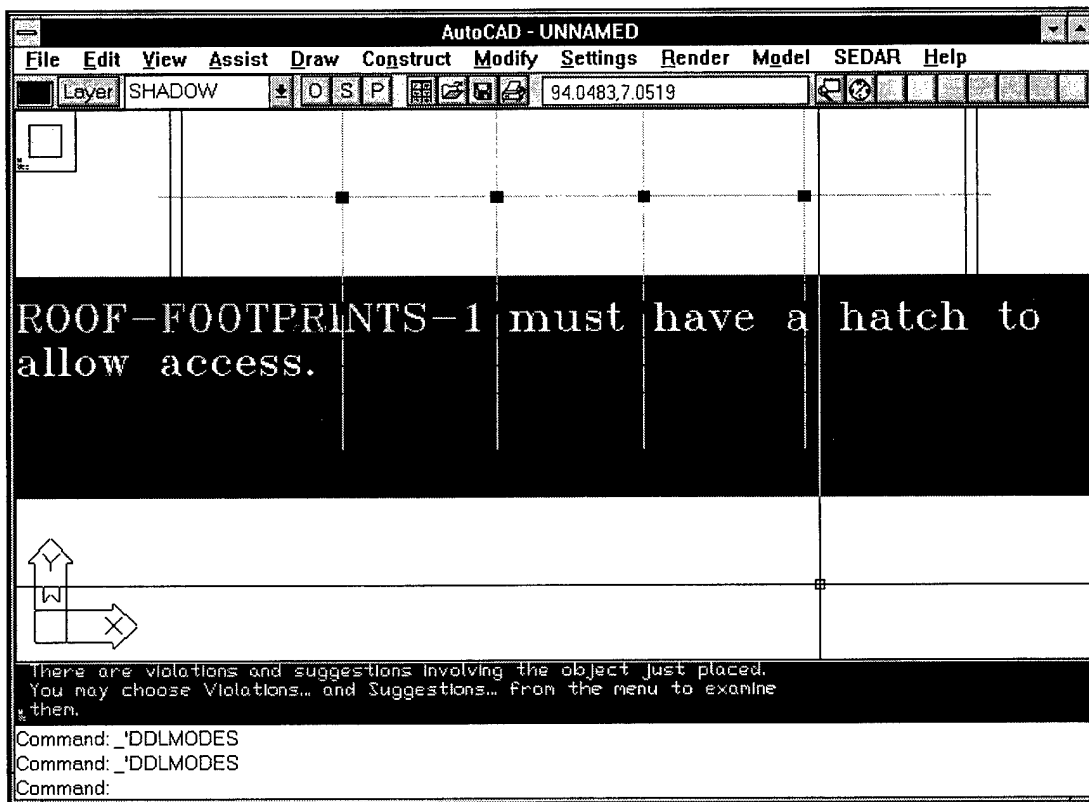


Figure 16. Example of a text display box.

of the drawing. The explanation box is drawn on a layer called the SHADOW layer. This layer needs to be created elsewhere.

The color of the explanation box will be the default color of the SHADOW layer. The text will be white, except for the object names, whose colors are determined by the function `get-color-from-violation-type` and the global variable `*SECONDARY-COLOR*`.

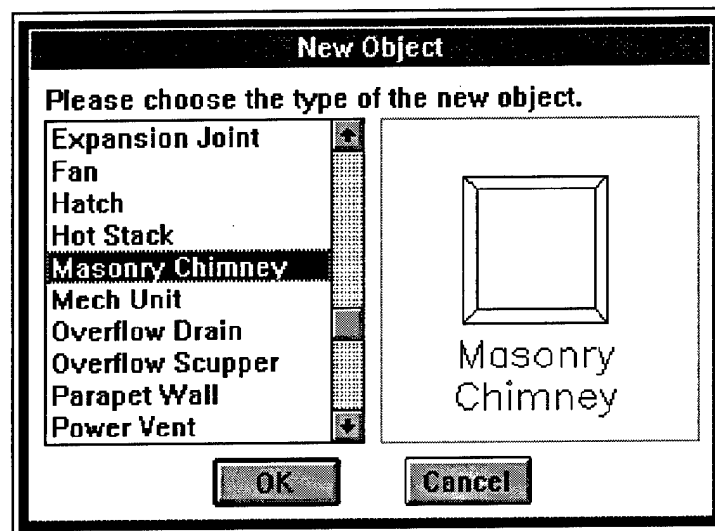


Figure 17. SEDAR architecture.

Major Function

(draw-explanation-box-and-text object-list explanation-list violation-level)

Parameters: object-list: List of names of objects involved in the explanation

explanation-list: The explanation in the form of a list of strings. Object names are separate strings, with a leading ? as a sentinel.

Here is a sample explanation-list:

("There should be a walkway from "
"?AC-UNITS-CURBED-1" " to " "?HATCHES-1"
".")

violation-level: Either physical, specification, or preference

This function may be reused in multiple ways. If used in an expert critiquing system that provides explanations of violations, then it can be used as it was originally intended. The object-list will contain the list of objects in the design that are referred to in the explanation. When the explanation box is displayed, the object names within the explanation will be colored differently from the rest of the text.

Alternatively, this function can be used simply to write out any string in a box overlaying an AutoCAD design. In this case, object-list and violation-level would be set to nil. Explanation-list would be a list of one element—the string to be displayed.

Note that this function does not check if the text will fit within the explanation box. Four lines of text will fit with the given settings.

Supporting Functions

```
(lower-left-of-exp-box object-list)
(draw-explanation-box the-point)
(get-first-word string)
(trim-leading-whitespace string)
(get-leading-whitespace string)
(all-spaces string)
(fits str-test left-x right-x ht)
(my-textbox string height)
(show-text string start-location left-margin right-margin line-ht char-ht
      tlw end-pt)
(object-name-p string)
(process-explanation-list explanation-list violation-type color-num start-
      location left-margin right-margin line-ht char-ht tlw)
(draw-explanation-text lower-left explanation-list level)
```

The Design Objects Dialog Box

Files: *ac-objs.lsp*, *globals.lsp*, *objects.dcl*, **.sld* — This component contains routines for displaying an AutoCAD dialog box showing names of design objects and their corresponding images. A list of names scrolls on the left, and one image is shown on the right. Whenever the user clicks on an object name, the image of that object is displayed. The information about objects and their images needs to be stored in a global variable called **OBJECTS**. The images themselves need to be stored in individual AutoCAD slide (.sld) files. The main function, *get-new-object-type*, has been separated from the rest of *ac-shell.lsp* and put into a file called *ac-objs.lsp*.

More generally, this dialog box could be used in any situation in an AutoCAD application in which a user must select one item out of a list, and each item has a corresponding image. A sample of the **OBJECTS** global variable is:

```
(setq *OBJECTS*
  Type          Dialog text name  Slide   Block   Shape          Size
  -----
  '(
    (ac-units-curbed "AC Unit on Curb" "ac-curb" "ac-curb" rectangular-composition 3.0)
    (ac-units-sleeps "AC Unit on Sleep" "ac-sleep" "ac-sleep" rectangular-composition 3.0)
    (area-dividers "Area Divider" "areadiv" nil nil nil)
    (attic-vents "Attic Vent" "vent" "vent" circle 0.25)
  ))
```

References

Cited

- [Brown 1986] D. Brown and B. Chandrasekaren. "Knowledge and control for a mechanical design expert system". *IEEE Computer*, 19(7) 92-100.
- [East 1995] E.W. East, T.L. Roessler, M.D. Lustig, and M.C.M. Fu. "The Reviewer's Assistant System: System Design Analysis and Description". Technical Report (TR) FF-95/09/ADA294604, U.S. Army Construction Engineering Research Laboratory, Champaign, IL.
- [Fu 1994] M. Fu et al. "Using a goal-based model of design in an expert critiquing system". In *Design Cognition and Design Education: Focus on the Role of Experience*, EduTech Symposium, Georgia Institute of Technology, Atlanta, GA, pages 14-19.
- [Griffin 1982] C. Griffin. *Manual of Built-Up Roof Systems*. McGraw-Hill Book Company, New York, NY.
- [NRCA 1985] National Roofing Contractors Association, Chicago. *The NRCA Roofing and Waterproofing Manual*, 2nd edition, Chicago, IL.

Uncited

- [Baykan 1992] C. Baykan and M. Fox. "Wright: a constraint-based spatial layout system". In *Artificial Intelligence in Engineering Design*, volume 1, chapter 11. Academic Press, Inc., New York, NY.
- [Cacciabue 1992] P. Cacciabue et al. "A cognitive model in a blackboard architecture: synergism of AI and psychology". *Reliability Engineering and System Safety*, 36:187-197.
- [Case 1994] M. Case. "The Discourse Model for Collaborative Engineering Design: A Distributed and Asynchronous Approach". PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL.
- [Clarke 1991] J. Clarke and D. Randall. "An intelligent front-end for computer-aided building design". *Artificial Intelligence in Engineering*, 6(1):36-45.
- [Echeverry 1991] D. Echeverry. "Factors for Generating Initial Construction Schedules." TR P-91/54/ADA243662, U.S. Army Construction Engineering Research Laboratory, September 1991.

- [Fazio 1989] P. Fazio and K. Gowri. "A knowledge-based system for the selection and design of roof systems". *The Journal of CIB Batiment International: Building Research and Practice*, 17(5):294-298.
- [Fischer 1993] G. Fischer et al. "Embedding critics in design environments". *The Knowledge Engineering Review*, 8(4):285-307.
- [Marcus 1992] S. Marcus et al. "VT: an expert elevator designer that uses knowledge-based backtracking". In *Artificial Intelligence in Engineering Design*, volume 1, chapter 11. Academic Press, Inc., New York, NY.
- [Mastaglio 1990] T. Mastaglio. "User modeling in computer-based critics". In *Proceedings of the 23rd Hawaii International Conference on System Sciences, Volume 3: Decision Support and Knowledge-Based Systems*, pages 403-411.
- [Miller 1986] P. Miller. *Expert Critiquing Systems: Practice-Based Medical Consultation by Computer*. Springer, New York, NY.
- [Morad 1994] A. Morad and Y. Beliveau. "Geometric-based reasoning system for project planning". *Journal of Computing in Civil Engineering*, 8(1):52-69.
- [Ohsuga 1989] S. Ohsuga. "Toward intelligent CAD systems. *Computer Aided Design*, 21(5):315-336.
- [Paek 1992] Y. Paek and H. Adeli. "An object space framework for design/construction integration". *Building and Environment*, 10(1):35-48.
- [Ramsey 1994] C. Ramsey and H. Sleeper. *Architectural Graphic Standards*, 9th edition. John Wiley and Sons Inc., New York, NY.
- [Roach 1984] J. Roach. "The rectangle placement language". In *Proceedings of the 21st IEEE Design Automation Conference*, pages 405-411.
- [RCI 1994] *Roof Consultants Institute's Glossary of Terms*, Roof Consultants Institute, Raleigh, NC.
- [Silverman 1992] B. Silverman. *Critiquing Human Error: A Knowledge-Based Human-Computer Collaboration Approach*. Academic Press, New York, NY.
- [Spickelmier 1988] R. Spickelmier and A. Newton. "CRITIC: a knowledge-based program for critiquing circuit designs". In *Proceedings of the 1988 IEEE International Conference of Computer Design: VLSI in Computers and Processors*, pages 324-327.
- [Steinberg 1984] L. Steinberg and T. Mitchell. "A knowledge-based approach to VLSI CAD: the REDESIGN system". In *Proceedings of the 21st IEEE Design Automation Conference*, pages 412-418.

- [Tong 1987] C. Tong. Goal-directed planning of the design process. In *Proceedings of the 1987 IEEE International Conference of Computer Design: VLSI in Computers and Processors*, pages 284-289.
- [USACE 1992] *Roofing Technology: Proponent Sponsored Engineer Corps Training (PROSPECT)*. U.S. Army Corps of Engineers and the American Roofing Consultants, Inc., Spencer, NC.
- [Zamanian 1992] M. Zamanian et al. "Representing spatial abstractions of constructed facilities". *Building and Environment*, 27(2):221-230.
- [Zhou 1989] H. Zhou et al. "CLEER: An AI system developed to assist equipment arrangements on warships". *Naval Engineers Journal*, 101(3):12-137.

Appendix A: Files and Locations

Goldworks III Files

In the *gcl44\sedar* subdirectory:

- cma-fn.lsp
- cma-main.lsp
- cma-rule.lsp
- decomp.lsp
- demons.lsp
- geometry.lsp
- prevdet.lsp
- review.lsp
- snap.lsp
- update.lsp
- violate.lsp

In *gcl44\sedar\kb* subdirectory:

- areadiv.lsp
- assert.lsp
- drains.lsp
- equip.lsp
- expansio.lsp
- frames.lsp
- kb.lsp
- obj-fn.lsp
- obj-rule.lsp
- roof.lsp
- scuppers.lsp
- vents.lsp

AutoCAD Files

In the *sedar* directory:

- Autolisp Files (*.lsp)
 - ac-expl.lsp
 - ac-init.lsp

ac-shell.lsp
attribs.lsp
globals.lsp
handlers.lsp
init.lsp
setup.lsp
slots.lsp

Drawing Files (*.dwg)

ac-curb.dwg
ac-sleep.dwg
ac-unit.dwg
areadiv.dwg
chim.dwg
chimney.dwg
column.dwg
drain.dwg
exh-fan.dwg
expjoint.dwg
fan.dwg
hatch.dwg
hotstack.dwg
hs.dwg
hvac.dwg
od.dwg
odrain.dwg
parwall.dwg
pv.dwg
pvent.dwg
rd.dwg
rdrain.dwg
rh.dwg
roofhatc.dwg
rv.dwg
rvpipe.dwg
scupper.dwg
sump.dwg
vent.dwg

DCL Files (*.dcl)

attribs.dcl
goals2.dcl
objects.dcl

slots.dcl
suggest.dcl
violatns.dcl

Slide Files (*.sld)

1slope.sld
2slope.sld
4slope.sld
ac-curb.sld
ac-sleep.sld
ac-unit.sld
areadiv.sld
chimney.sld
drain.sld
exh-fan.sld
expjoint.sld
fan.sld
hatch.sld
hotstack.sld
hvac.sld
odrain.sld
parwall.sld
pvent.sld
rdrain.sld
rooffoot.sld
roofhatc.sld
rvpipe.sld
scupper.sld
vent.sld
walkway.sld

In the *acadwin* directory:

acad.mnl
acad.mnu
cadreglb.lsp

Appendix B: Function Listings by File

Expert Critiquing Shell Files

File: **cma-main.lsp**

Function

Arguments

user::ac-message

query-id msg-string &rest msg-info

convert-from-string

arg

deep-convert-from-string

arg

do-get-object-slots

lst

do-get-object-slot-values

lst

do-get-object-slot-values*

object-id slot-list

do-get-object-slot-defaults

lst

do-get-object-slot-defaults*

object-id slot-list

do-get-object-children

request lst

inorder-traversal

frame-name

do-get-object-parents

request lst

do-modify-slot-values

lst

do-modify-slot-values*

object-id slot-value-list

do-get-dtm-all

do-get-dtm-activations
 activation
get-all-dtm-activations

do-get-dtm-task-status
 lst
do-get-dtm-tasks
 relation-type task
do-set-dtm-task-activation
 lst
do-get-dtm-task-rules
 lst
do-rule-query
 query-type lst
get-rule-info
 rule-name
do-rule-activation
 activation lst
do-set-review-type
 lst
do-set-critique-type
 critique-type lst
do-reject-critique
 lst
update-kb
 query-id msg-string &optional msg-info
update-tasks
 query-id object-type
recently-activated
 query-num task
recently-activated*
 query-num task depth
task-update-situation-p
 query-num
do-delete-object
 msg-info
do-delete-object*
 obj
delete-assertions
 obj-name
do-review-tasks
 msg-string &optional msg-info

do-select-object
 msg-info
remove-nils
 lst
do-place-object
 msg-info
do-move-object
 msg-info
do-resize-object
 msg-info
get-object-descriptions

File: cma-fn.lsp

Function

Arguments

null?
 x
sqr
 x
minimum
 lst
maximum
 lst
filter
 f lst
filter-mapcar
 filter-fn map-fn lst
clear-all

n-last
 n llist
n-last*
 threshold current llist
violation-name

combine
 f zero list
count
 elt list
count-objects
 the-frame big-instance

set-start-time

print-elapsed-time

detail-list-test

e1 e2

assert-subtask-list

subtasks parent

check-and-activate-tasklist

tasklist

make-object-instance

msg-info

apply-lisp-functions

arg func-list

legal-object?

object-type

frame-ancestor

ancestor descendant

frame-ordered

task1 task2

frame-ordered*

task1 task2 task1-parents task2-parents

frame-ordered**

task task-list

all-frame-instances

frame

File: decomp.lsp

Function

Arguments

maximum-decomposition

rect-obj

horizontal-decomposition

rect-obj

combine-horizontal-areas

area-list

combine-horizontal-areas*

area-list combined-area-list

combine-area-horizontally

extent extent-list

vertical-decomposition
 rect-obj
combine-vertical-areas
 area-list
combine-vertical-areas*
 area-list combined-area-list
combine-area-vertically
 extent extent-list
filter-out-rectangles
 rect-list rect-obj
make-h-slices
 extent h-borders
make-h-slices*
 h-borders extent current-y
make-v-slices
 h-slice-list extent v-borders
make-v-slices*
 h-slice-list current-x v-borders
make-v-slices**
 h-slice current-x v-borders
subtract-area
 start-list subtract-list
subtract-area*
 left subtract-list
remove-rectangle
 left sub-area
one-corner-extent-overlap
 extent1 extent2
two-corner-extent-overlap
 extent1 extent2
one-side-extent-overlap
 extent1 extent2
two-side-extent-overlap
 extent1 extent2
num-intersecting-corners
 extent1 extent2
num-intersecting-corners*
 point-list extent
form-complete-overlap-remainder
 extent1 extent2
form-one-corner-remainder
 extent1 extent2

form-two-corner-remainder

extent1 extent2

form-one-side-remainder

extent1 extent2

form-two-side-remainder

extent1 extent2

File: geometry.lsp

Function

Arguments

border-order

e1 e2

make-vertical-borders

coord-list

make-vertical-borders*

coord-list

make-horizontal-borders

coord-list

make-horizontal-borders*

coord-list

legal-composition

coord-list

legal-composition*

current-coord rest-list

make-extent

coord-list

make-extent*

coord-list min-x min-y max-x max-y

make-coord-info

extent

point-distance

point1 point2

point-in-rect

point rect

point-in-rect1

point v-borders

on-horizontal-border

x-val y-val horizontal-borders

on-vertical-border

x-val y-val vertical-borders

num-right-crossings
 x-val y-val vertical-borders

num-border-crossings
 point1 point2 border-list direction

complete-extent-overlap
 extent1 extent2

no-extent-overlap
 extent1 extent2

point-in-extent
 point extent

point-strictly-in-extent
 point extent

complete-overlap
 object1 object2

complete-overlap-cc
 circle1 circle2

complete-overlap-rr
 rect1 rect2

complete-overlap-rr*
 rect1 coordlist1 coordlist2 v-borders h-borders

check-all-borders
 coordlist1 v-borders h-borders

complete-overlap-rc
 rect1 circle1

complete-overlap-rc*
 coordlist center radius rect

segment-within-distance
 endpt1 endpt2 point distance

segment-within-distance*
 pt1 pt2 point distance direction

no-overlap
 object1 object2

no-overlap-cc
 circle1 circle2

no-overlap-rr
 rect1 rect2

no-overlap-rr*
 rect1 coordlist1 coordlist2 v-borders h-borders

no-overlap-rc
 rect1 circle1

no-overlap-rc*
 coordlist center radius rect

next-to-outside
 object1 object2

next-to-inside
 object1 object2

simple-span-extent
 extent1 extent2 tolerance

simple-span-rr
 rect-obj1 rect-obj2 tolerance

simple-span-extent
 extent1 extent2 tolerance

simple-span-extent-rr
 extent1 extent2 tolerance

spans-roof
 obj roof-obj

spans-roof*
 obj extent-list

next-to-cc
 circle1 circle2

next-to-rc
 rect circle

adjacent
 object1 object2

adjacent-cc
 circle-obj1 circle-obj2

adjacent-rc
 rect-obj circ-obj

rect-segments-touch-circle
 h-borders v-borders center radius

get-first-coord
 border

get-second-coord
 border

rect-points-touch-circle
 coord-list center radius

adjacent-rr
 rect-obj1 rect-obj2

check-colinearity-overlap-segments
 borders1 borders2

check-colinearity-overlap-segments*
 border border-list

intersection
 object1 object2

adjacent-on-edge
 object1 object2 object2-side

intersect-on-edge
 object1 object2 object2-side

aligned
 object1 object2 tolerance

area-of
 object

area-of-c
 circle

area-of-rc
 rect

area-of-rc*
 h-borders v-borders left-x right-x area

area-of-rc**
 y-coord delta-y right-x v-border-list

pop-border-list
 h border-list

pop-border-list*
 h last-elt rest-list

filter-heights
 h1 h2 v-border-list

traversable
 obj1 obj2 path-obj

connected?
 obj1 obj2

check-walkway-objects
 obj1 obj2 walkway

object-connected
 object-list target-obj

connected?*
 obj1 obj2 current-walkway

get-relative-distance
 object border-list border-type

get-relative-distance-circle
 center radius border-list border-type

get-relative-distance-rect
 coordinate-info extent border-list border-type

compute-distance
 obj1 obj2

compute-distance-cc
 circle-obj1 circle-obj2

compute-distance-rc
 rect-obj circle-obj

distance-to-corners
 corner-list center radius

distance-to-vert-borders
 border-list center radius

distance-to-horiz-borders
 border-list center radius

compute-distance-rr
 rect-obj1 rect-obj2

rect-comps-dist
 rect-comp1 rect-comp2

rect-comps-dist-aux
 rect-comp1 rect-comp2

rect-comp-point-dist
 rect-comp point

draw-point
 point

edge-point-dist
 point edge

vert-edge-point-dist
 point edge

horiz-edge-point-dist
 point edge

pointx
 point

pointy
 point

edgex1
 edge

edgey1
 edge

edgex2
 edge

edgey2
 edge

distance
 point1 point2

opposite-orientation
 orientation

distance-to-line
 point line

get-edge
 simple-rectangle edge
 horizontal?
 line
 vertical?
 line
 north-of
 rect1 rect2
 north-of-extent
 rect1-extent rect2-extent
 south-of
 rect1 rect2
 south-of-extent
 rect1-extent rect2-extent
 west-of
 rect1 rect2
 west-of-extent
 rect1-extent rect2-extent
 east-of
 rect1 rect2
 east-of-extent
 rect1-extent rect2-extent
 determine-alignment
 rect1 rect2
 determine-alignment-extent
 rect1-extent rect2-extent
 line-distance
 line1 line2
 exceeds-max-distance-p
 obj1 obj2 maxd

File: prevdet.lsp

Function

Arguments

build-rule-object-type-list
 agent rule-name
 first-highest-priority-task
 task-list
 max-priority
 a-task a-priority

get-active-rules
 agent
get-active-rules-for-object-place
 agent
build-rule-task-list
 agent rule
check-intersection
 list1 list2
make-object-place-ruleset
 active-rule-list
make-object-select-ruleset
 active-rule-list
make-od-rule
 new-rule-name old-rule-name trigger-name object-list bindable-list
make-new-consequent
 old-consequent bindings rule-name
make-new-antecedent
 old-antecedent bindings
instantiate
 s-exp binding-list
instantiate-binding
 s-exp binding quote-p
form-binding
 object-list bindable-list
form-binding*
 object-list prev-bindable-list rest-bindable-list

File: review.lsp

Function

Arguments

make-all-review-rules
 agent
make-all-focus-review-rules
 agent
make-review-rules-for-task-subtree
 agent task
make-review-rules
 agent task-list
activate-rules-by-tasks
 agent task-list

File: snap.lsp

Function

Arguments

snap-to-fit-circle-point

circ-obj point

snap-one-slope-das

one-slope-das

snap-four-slope-das

four-slope-das

snap-drainage-area

one-slope-das tolerance

equal-extents

extent1 extent2

align-adjust-width-rr

snapper snappee

find-nearest-adjacent-edge

*obj1 obj2***File: update.lsp**

Function

Arguments

update-assert

object-type current-agent

filter-deactivated-tasks

task-list

get-tasks

rule-list current-agent

get-interfering-tasks

task-list current-agent

get-last-tasks

filter-recently-activated

*task-assertion-list***File: violate.lsp**

Function

Arguments

order-by-tasks

rule-task-pair-compare
 rule-task-pair1 rule-task-pair2

task-compare
 task1 task2

sort-by-priority-reverse
 rule-list

order-by-tasks-select

make-violation-action
 violation-name rule-name task-name level rule-type var-bindings
 violation-action explanation

make-select-violation-action
 constraint-area-name obj-type rule-num task level obj-id bindable-list
 binding-list explanation violation-action

get-binding
 var binding-list

make-explanation
 explanation var-bindings

cat-explanation-terms
 inst-expl

make-explanation-list
 explanation var-bindings

cat-explanation-terms-2
 inst-expl

make-object-list
 var-bindings

make-object-list
 explanation var-bindings

object-variable-p
 str

get-object-string-assoc
 obj-type binding-list

get-other-object-string-assoc
 obj-id binding-list bindable-list

get-var-name
 obj-id binding-list

get-other-object-string-assoc*
 var-name bindable-list

get-partial-action
 var violation-action

get-partial-action*
 var action-list

string-member
 sym lst
atomic-listp
 lst
var-obj-instantiate
 var obj-id lst

Flat and Low-Slope Roof Knowledge Base Files

File: kb.lsp

Function

Arguments

binding-list-match
 binding-list1 binding-list2
checked-before-dual
 rulenum &rest binding-list
checked-before
 rulenum &rest binding-list

File: obj-fn.lsp

Function

Arguments

make-assoc-id

make-penetration-id

make-slice-id
 direction
make-wall-segment-id

make-roof-drains-id

make-column-id

make-roof-walls
 roof-footprints-id
make-roof-walls*
 roof-footprints-id coord-info width point1 point2 list-length

get-bounding-points
 pt1 pt2 pt3 pt4

classify-corner
 pt1 pt2 pt3

make-roof-edges
 roof-footprints-id

make-expansion-joint-points
 roof-footprints-id

make-joint-points*
 roof-id area-list direction

make-joint-points-h**
 roof-id area area-list

make-joint-points-v**
 roof-id area area-list

complete-expansion-joint-slots
 joint-obj

clip-or-extend-to-roof
 obj roof-obj

clip-or-extend-to-roof*
 obj extent-list

clip-or-extend-rc-objects
 obj extent

make-footprint-slices
 roof-obj

make-footprint-slices*
 area-list roof-obj slice-type

test-cricket
 low-point edge-point1 edge-point2 obj

make-line
 point1 point2

check-corners
 line coord-list

which-cricket
 four-slope-da obj

make-wall-segments
 wall

make-wall-segments*
 endpoint1 endpoint2 endpoint-list half-width wall-id segment-count

assess-spacing
 slice distance-interval

assess-spacing*
 slice distance-interval

check-distance-intervals
 offset-list interval list-length
 create-low-point-drain
 four-slope-da
 make-columns
 column-list
 do-vertical-column-lines
 column-list endpt1 endpt2 height
 do-horizontal-column-lines
 column-list endpt1 endpt2 height

User Interface (AutoLisp) Files

File: ac-expl.lsp

Function

Arguments

draw-explanation-box-and-text
 object-list explanation-list violation-level / lower-left
 lower-left-of-exp-box
 object-list
 in-middle-third
 object-name / shape center radius rc y-maxmin top bottom
 in-bottom-third
 object-name / shape center radius rc y-min
 in-top-third
 object-name / shape center radius rc y-max
 in-all-3-regions
 object-name
 draw-explanation-box
 the-point
 str-to-sym
 string
 get-first-word
 string / whtspc
 get-first-word-aux
 string
 trim-leading-whitespace
 string
 get-leading-whitespace
 string

all-spaces

string

fits

str-test left-x right-x ht

my-textbox

string height

show-text

string start-location left-margin right-margin line-ht char-ht tlw end-pt /

str-fits

str-test remainder done next-word new-end-pt

object-name-p

string

process-explanation-list

*explanation-list violation-type color-num start-location left-margin
right-margin line-ht char-ht tlw / string new-color-num*

draw-explanation-text

lower-left explanation-list level

erase-shadow-layer

draw-message-text

string

draw-message-text-at-line

string line-num

draw-message-text-at-location

string start-location

erase-message-window

/ selset

make-message-window-blue

File: ac-init.lsp

Function

Arguments

init-log-file

File: ac-shell.lsp

Function

Arguments

filter

f lst

position
 item the-list
position-aux
 item the-list n
violation-types
 violations-list / tmp
violation-message
 the-string / viol-types
save-roof-layout

save-globals
 pathname
open-roof-layout

toggle-influencer-mode

toggle-debiaser-mode

delete-object-callback

resize-object-callback

c:done

change-object-slot-values

move-object-callback

get-object-constraint-layers
 object-name
get-object-constraint-layers*
 actions object-name
offset-layers-bounds
 layers delta-x delta-y
new-object-callback

get-new-object
 / viol-types
create-new-objects
 objects
create-new-object
 object

add-if-not-null

x

get-new-object-type

object-list-click-callback

hierarchic-stringify

l

hierarchic-stringify*

l prefix-string

hierarchic-stringify-children

children prefix-string

active-state-string

task

get-active-state

tasks task

on-state-string

task

add-task

task

tasks-callback

/ tasks-orig

activate-task

deactivate-task

turn-task-on

/ old-task new-task

turn-task-off

/ old-task new-task

perform-task-activations

perform-task-activations-aux

changes tasks-list-click-callback

mk_list

readlist displist / count item retlist

violations-callback

/ true-violations-list

suggestions-callback

/ suggestions-list

get-some-violations

violations-list violation-type rule-type

get-some-violations
 violations-list violation-type
physical-violations-list-click-callback

specification-violations-list-click-callback

preference-violations-list-click-callback

view-violation-callback

shift-coordinates
 deltax deltay coords
forget-object-constraints
 object
forget-object-constraints-aux
 action-layers object
forget-object-constraint
 action-layer object
action-depends-on-object
 action object
subactions-depend-on-object
 subactions object
get-object-shape
 obj-type
get-dwg-object-center
 object-name
get-dwg-object-radius
 object-name
get-dwg-object-entity
 object-name
get-dwg-object-type
 object-name
get-dwg-object-vertices
 object-name
delete-from-dwg-object
 object-name
delete-from-dwg-object*
 obj-name obj-list
get-dwg-object-from-entity
 ent
get-dwg-object-from-entity*
 ent obj-list

call-gcl
 msg msg-info
stringify
 x
perform-review-actions
 msg-string msg-info / viol-types
perform-critique-actions
 critique / lower-left
get-violation-level
 critique
get-rule-type
 critique
get-critique-action
 critique
get-critique-explanation
 critique
get-object-list
 critique
get-explanation-list
 critique
show-constraints
 constraints object-list
draw-constraint-actions
 object-list
draw-constraint-action
 constraint-action object-list / action layer-name
get-color-from-violation-type
 violation-type
create-and-color-constraint-layers
 constraints object-list
create-and-color-constraint-layer
 constraint object-list
thaw-critique-layers
 critique-list
freeze-critique-layers
 critique-list
build-critique-layer-list
 critique-list
thaw-layers
 layer-list
thaw-layers*
 layer-list

freeze-layers
 layer-list

freeze-layers*
 layer-list

generate-constraint-block
 constraints block-name object center

draw-constraint-layers
 constraints

build-constraint-block
 layers block-name object center

build-constraint-block-filter
 layers

draw-layers-bounding-box
 layers

find-layers-bounding-box
 layers

draw-critique-explanation
 critique-explanation

draw-attention-text
 text

draw-thinking-text

draw-critique
 critique-action object-list

show-critique
 critique-action critique

draw-outline
 object-name shape

draw-arrow
 source-action dest-action object-list

map-shadow-points
 points

map-shadow-point
 point

find-nearest-point
 point points

find-nearest-point-aux
 point points nearest second-nearest

draw-shadow-object
 object-type location

draw-exterior-circle-constraint
 object-name size

draw-interior-circle-constraint
 object-name size

draw-circle-constraint
 object-name hatch

draw-rc-constraint
 object-name hatch

draw-exterior-rc-constraint
 object-name size

draw-interior-rc-constraint
 object-name size

draw-boundary-area
 object-name shape boundary-type size

clear-violation

clear-violation-aux
 constraint-layers

upto
 elt lst

first
 lst

second
 lst

third
 lst

fourth
 lst

fifth
 lst

sixth
 lst

seventh
 lst

eighth
 lst

ninth
 lst

get-x-maxmin
 coord-list

get-x-maxmin*
 coord-list maxval minval

get-y-maxmin
 coord-list

get-y-maxmin*
 coord-list max min

gensym
 object-type

draw-rect-comp
 rect-comp

draw-rect-comp-aux
 rect-comp

draw-outside-rect-comp
 rect-comp radius

draw-outside-rect-comp-aux
 rect-comp radius direction

draw-right
 x y next-direction radius

draw-left
 x y next-direction radius

draw-up
 x y next-direction radius

draw-down
 x y next-direction radius

draw-inside-rect-comp
 rect-comp radius

draw-inside-rect-comp-aux
 rect-comp radius direction

draw-inside-right
 x y next-direction radius

draw-inside-left
 x y next-direction radius

draw-inside-up
 x y next-direction radius

draw-inside-down
 x y next-direction radius

File: globals.lsp

Function

Arguments

id

x

File: attribs.lsp

Function

Arguments

change-attribs

object-name

stringify-elements

l

stringify-pairs

l

attribs-list-click-callback

File: handlers.lsp

Function

Arguments

get-attic-vents

create-attic-vents

object-info

get-hot-stacks

create-hot-stacks

object-info

get-overflow-drains

create-overflow-drains

object-info

get-roof-drains

create-roof-drains

object-info

get-roof-vent-pipes

create-roof-vent-pipes

object-info

get-fans

create-fans

object-info

get-circular-object
 radius
create-circular-object
 object-info block-filename
get-ac-units-curbed

create-ac-units-curbed
 object-info
get-ac-units-sleeps

create-ac-units-sleeps
 object-info
get-exhaust-fans

create-exhaust-fans
 object-info
get-mech-units

create-mech-units
 object-info
get-power-vents

create-power-vents
 object-info
get-hatches

create-hatches
 object-info
get-masonry-chims

create-masonry-chims
 object-info
create-columns
 object-info
get-rectangular-object
 width height
create-rectangular-object
 object-info block-filename
get-walls

create-wall-segments
 object-info
get-scuppers

create-scuppers
 object-info
get-walkways

create-walkways
 object-info
get-roof-footprints

create-roof-footprints
 object-info
get-exp-joints

create-exp-joints
 object-info
get-area-dividers

create-area-dividers
 object-info
get-column-lines

create-column-lines
 object-info
get-four-slope-das

create-four-slope-das
 object-info
get-two-slope-das

create-two-slope-das
 object-info
get-one-slope-das

create-one-slope-das
 object-info
convert-da-to-rc
 da
get-rc

get-ortho-pline
 prompt
input-pline
 layer-name
get-ww

normalize-rc
 points
make-rc-start-right
 points
clean-up-rc
 points
clean-up-rc*
 points cleaned-points
clean-up-segment
 point1 point2
offset
 points xoffset yoffset
compute-ww-rc
 path radius
get-direction
 this-point next-point
compute-ww-side-1
 path radius
continue-path-1
 path last-direction radius
find-next-ww-point-1
 this-x this-y this-direction last-direction radius
compute-ww-side-2
 path radius
continue-path-2
 path last-direction radius
find-next-ww-point-2
 this-x this-y this-direction last-direction radius
make-clockwise
 path
is-clockwise
 path
is-clockwise*
 path total
turn-value
 last-direction this-direction
convert-pline-to-rc
 layer-name
get-pline-vertices
 entity
get-da
 drawing-fn
find-max-min

pin-mouse-point

draw-4slope

draw-2slope

draw-1slope

wrap-rc-coords
rect-comp

File: setup.lsp

Function
Arguments

S::STARTUP

File: acad.mnl

Function
Arguments

move-object

new-object

resize-object

delete-object

ai_tiledvp_chk

ai_tiledvp
num ori / ai_tiles_g ai_tiles_cmde

ai_tab1

ai_tab2

ai_tab3

ai_tab4

merr

msg

merrmsg

msg

c:rectang

/ cmde pt1 pt2

c:ai_peditm

/ m:p0 m:p1

m:p0

/ m:s1 m:e1 m:e2 m:e3

m:p1

/ m:a

ai_rootmenus

Appendix C: Rules and Rule Set Listings by File

Expert Critiquing Shell Files

File: cma-rule.lsp

Rule Set Definitions

Rule Set Rules

clear-cache-information

remove-cache-info1

remove-cache-info2

remove-cache-info3

remove-relation-info1

remove-relation-info2

remove-relation-info3

remove-relation-info4

remove-relation-info5

remove-relation-info6

mark-active-rules

deactivate-active-rules

Rule Definitions

remove-cache-info1

remove-cache-info2

remove-cache-info3

remove-relation-info1

remove-relation-info2

remove-relation-info3

remove-relation-info4

remove-relation-info5

remove-relation-info6

deactivate-active-rules

File: demons.lsp

Rule Set Definitions

Rule Set Rules

all-demons

clear-all-shadow-objects1
clear-all-shadow-objects2
clear-all-active-rules
clear-all-temporary-rules
clear-all-temporary-rules-prime
clear-all-violations
clear-all-violations-prime
clear-constraint-rule-set
clear-all-interact-rules
clear-all-interact-rules-prime
clear-task-list
mark-task-list
mark-task-list-prime
unmark-task-list
unmark-task-list-prime
clear-all

Rule Definitions

clear-all-shadow-objects1
clear-all-shadow-objects2
clear-all-active-rules
clear-task-list
clear-all
mark-task-list
mark-task-list-prime
unmark-task-list
unmark-task-list-prime
clear-all-temporary-rules
clear-all-temporary-rules
clear-all-violations
clear-all-violations-prime
clear-constraint-rule-set
clear-all-interact-rules
clear-all-interact-rules-prime

File: update.lsp

Rule Set Definitions

Rule Set Rules

update-tasks-frame

*update-tasks-frame1**update-tasks-frame2*

update-tasks-copy

*update-tasks-copy1**update-tasks-copy2*

Rule Definitions

update-tasks-frame1

update-tasks-frame2

update-tasks-copy1

update-tasks-copy2

Flat and Low-Slope Roof Knowledge Base Files**File: areadiv.lsp**

Rule Definitions

ruleF-1-trigger

ruleF-1-condition

ruleF-2-a-trigger

ruleF-2-a-condition

ruleF-2-b-trigger

ruleF-2-b-condition

ruleF-2-c-trigger

ruleF-2-c-condition

ruleF-3-trigger

ruleF-3-condition

ruleF-1-1-2-trigger

ruleF-1-1-2-condition

ruleF-1-3-1-a-1-trigger

ruleF-1-3-1-a-1-condition

ruleF-1-3-1-a-2-trigger

ruleF-1-3-1-a-2-condition

ruleF-1-3-1-b-trigger

ruleF-1-3-1-b-condition
ruleF-1-3-1-c-trigger
ruleF-1-3-1-c-condition

File: drains.lsp

Rule Definitions

rule1-trigger
rule1-condition
rule2-trigger
rule2-condition
rule3-trigger
rule3-condition
rule4-trigger
rule4-condition
rule17-trigger
rule17-condition
rule22-trigger
rule22-condition
ruleO-4-trigger
ruleO-4-condition
ruleO-5-trigger
ruleO-5-condition
ruleO-6-trigger
ruleO-6-condition
ruleO-7-trigger
ruleO-7-condition
ruleO-8-trigger
ruleO-8-condition
ruleO-10-a-trigger
ruleO-10-a-condition
ruleO-10-b-trigger
ruleO-10-b-condition
ruleO-14-trigger
ruleO-14-condition
ruleO-15-trigger
ruleO-15-condition
ruleO-16-trigger
ruleO-16-condition
ruleO-17-trigger
ruleO-17-condition
ruleO-18-trigger

ruleO-18-condition
ruleO-19-a-trigger
ruleO-19-a-condition
ruleO-19-b-trigger
ruleO-19-b-condition
ruleO-19-c-trigger
ruleO-19-c-condition
ruleO-19-d-trigger
ruleO-19-d-condition
ruleO-20-trigger
ruleO-20-condition
ruleO-22-trigger
ruleO-22-condition
ruleO-23-trigger
ruleO-23-condition
ruleO-24-trigger
ruleO-24-condition
ruleO-25-a-trigger
ruleO-25-a-condition
ruleO-25-b-trigger
ruleO-25-b-condition
ruleO-26-a-trigger
ruleO-26-a-condition
ruleO-26-b-trigger
ruleO-26-b-condition
ruleO-26-c-trigger
ruleO-26-c-condition
rule5-trigger
rule5-condition
rule9-trigger
rule9-condition

File: equip.lsp

Rule Definitions

rule6-trigger
rule6-condition
rule7-trigger
rule7-condition
rule8-trigger
rule8-condition
rule12-trigger

rule12-condition
rule13-trigger
rule13-condition
rule14-trigger
rule14-condition
rule15-trigger
rule15-condition
rule18-trigger
rule18-condition
rule19-trigger
rule19-condition
rule20-trigger
rule20-condition
rule21-trigger
rule21-condition
rule23-trigger
rule23-condition

File: expansio.lsp

Rule Definitions

ruleE-1-trigger
ruleE-1-condition
ruleE-2-a-trigger
ruleE-2-a-condition
ruleE-2-b-trigger
ruleE-2-b-condition
ruleE-2-c-trigger
ruleE-2-c-condition
ruleE-3-trigger
ruleE-3-condition
ruleE-1-1-1-a-trigger
ruleE-1-1-1-a-condition
ruleE-1-1-1-b-trigger
ruleE-1-1-1-b-condition
ruleE-1-1-1-c-trigger
ruleE-1-1-1-c-condition
ruleE-1-1-1-d-trigger
ruleE-1-1-1-d-condition
ruleE-1-1-1-e-trigger
ruleE-1-1-1-e-condition
ruleE-1-1-2-trigger

ruleE-1-1-2-condition
ruleE-1-3-1-a-1-trigger
ruleE-1-3-1-a-1-condition
ruleE-1-3-1-a-2-trigger
ruleE-1-3-1-a-2-condition
ruleE-1-3-1-b-trigger
ruleE-1-3-1-b-condition
ruleE-1-3-1-c-trigger
ruleE-1-3-1-c-condition

File: kb.lsp

Rule Set Definitions

Rule Set Rules

constraint-rules

rule1-trigger
rule1-condition
rule2-trigger
rule2-condition
rule3-trigger
rule3-condition
rule4-trigger
rule4-condition
rule5-trigger
rule5-condition
rule6-trigger
rule6-condition
rule7-trigger
rule7-condition
rule8-trigger
rule8-condition
rule9-trigger
rule9-condition
rule11-trigger
rule11-condition
rule12-trigger
rule12-condition
rule13-trigger
rule13-condition
rule14-trigger
rule14-condition

rule15-trigger
rule15-condition
rule16-trigger
rule16-condition
rule17-trigger
rule17-condition
rule18-trigger
rule18-condition
rule19-trigger
rule19-condition
ruleH111a-trigger
ruleH111a-condition
ruleH111b-trigger
ruleH111b-condition
ruleH121-trigger
ruleH121-condition
ruleK221-trigger
ruleK221-condition
ruleK222-trigger
ruleK222-condition
ruleK223-trigger
ruleK223-condition
ruleK224-trigger
ruleK224-condition
ruleO-6-trigger
ruleO-6-condition
ruleO-7-trigger
ruleO-7-condition
ruleO-8-trigger
ruleO-8-condition
ruleO-10-a-trigger
ruleO-10-a-condition
ruleO-10-b-trigger
ruleO-10-b-condition
ruleO-14-trigger
ruleO-14-condition
ruleO-15-trigger
ruleO-15-condition
ruleO-16-trigger
ruleO-16-condition
ruleO-17-trigger
ruleO-17-condition
ruleO-18-trigger

ruleO-18-condition
ruleO-19-a-trigger
ruleO-19-a-condition
ruleO-19-b-trigger
ruleO-19-b-condition
ruleO-19-c-trigger
ruleO-19-c-condition
ruleO-19-d-trigger
ruleO-19-d-condition
ruleO-20-trigger
ruleO-20-condition
ruleO-22-trigger
ruleO-22-condition
ruleO-23-trigger
ruleO-23-condition
ruleO-24-trigger
ruleO-24-condition
ruleO-25-a-trigger
ruleO-25-a-condition
ruleO-25-b-trigger
ruleO-25-b-condition
ruleO-26-a-trigger
ruleO-26-a-condition
ruleO-26-b-trigger
ruleO-26-b-condition
ruleO-26-c-trigger
ruleO-26-c-condition
ruleE-1-trigger
ruleE-1-condition
ruleE-2-a-trigger
ruleE-2-a-condition
ruleE-2-b-trigger
ruleE-2-b-condition
ruleE-2-c-trigger
ruleE-2-c-condition
ruleE-3-trigger
ruleE-3-condition
ruleE-1-1-1-a-trigger
ruleE-1-1-1-a-condition
ruleE-1-1-1-b-trigger
ruleE-1-1-1-b-condition
ruleE-1-1-1-c-trigger
ruleE-1-1-1-c-condition

ruleE-1-1-1-d-trigger
ruleE-1-1-1-d-condition
ruleE-1-1-1-e-trigger
ruleE-1-1-1-e-condition
ruleE-1-1-2-trigger
ruleE-1-1-2-condition
ruleE-1-3-1-a-1-trigger
ruleE-1-3-1-a-1-condition
ruleE-1-3-1-a-2-trigger
ruleE-1-3-1-a-2-condition
ruleE-1-3-1-b-trigger
ruleE-1-3-1-b-condition
ruleE-1-3-1-c-trigger
ruleE-1-3-1-c-condition
ruleV-1-trigger
ruleV-1-condition
ruleV-2-trigger
ruleV-2-condition
ruleV-3-trigger
ruleV-3-condition
ruleV-4-trigger
ruleV-4-condition
ruleV-5-trigger
ruleV-5-condition
ruleV-6-trigger
ruleV-6-condition
ruleV-7-trigger
ruleV-7-condition
ruleV-8-trigger
ruleV-8-condition
ruleV-9-trigger
ruleV-9-condition
ruleV-10-trigger
ruleV-10-condition
ruleV-11-trigger
ruleV-11-condition
ruleV-12-trigger
ruleV-12-condition
ruleV-13-trigger
ruleV-13-condition
ruleV-14-trigger
ruleV-14-condition
ruleV-15-trigger

ruleV-15-condition
ruleV-17-trigger
ruleV-17-condition
ruleI-1-5-13-trigger
ruleI-1-5-13-condition
ruleI-1-5-16-trigger
ruleI-1-5-16-condition
ruleR-2-a-trigger
ruleR-2-a-condition
ruleR-2-b-trigger
ruleR-2-b-condition
ruleR-3-trigger
ruleR-3-condition
ruleF-1-trigger
ruleF-1-condition
ruleF-2-a-trigger
ruleF-2-a-condition
ruleF-2-b-trigger
ruleF-2-b-condition
ruleF-2-c-trigger
ruleF-2-c-condition
ruleF-3-trigger
ruleF-3-condition
ruleF-1-1-2-trigger
ruleF-1-1-2-condition
ruleF-1-3-1-a-1-trigger
ruleF-1-3-1-a-1-condition
ruleF-1-3-1-a-2-trigger
ruleF-1-3-1-a-2-condition
ruleF-1-3-1-b-trigger
ruleF-1-3-1-b-condition
ruleF-1-3-1-c-trigger
ruleF-1-3-1-c-condition
rule20-trigger
rule20-condition
rule21-trigger
rule21-condition
ruleO-4-trigger
ruleO-4-condition
ruleO-5-trigger
ruleO-5-condition
ruleV-16-a-trigger
ruleV-16-a-condition

ruleV-16-b-trigger
ruleV-16-b-condition
ruleN-1-trigger
ruleN-1-condition
ruleN-2-trigger
ruleN-2-condition
rule22-trigger
rule22-condition
rule23-trigger
rule23-condition
remove-duals
remove-duplicates1
remove-duplicates2
perform-subsumption-physical1
perform-subsumption-physical2
perform-subsumption-specification
perform-subsumption-or-rules1
perform-subsumption-or-rules2
cache-failed-check-conditions

Rule Definitions

ruleH111a-trigger
ruleH111a-condition
ruleH111b-trigger
ruleH111b-condition
ruleH121-trigger
ruleH121-condition
ruleK221-trigger
ruleK221-condition
ruleK222-trigger
ruleK222-condition
ruleK223-trigger
ruleK223-condition
ruleK224-trigger
ruleK224-condition
remove-duals
remove-duplicates1
remove-duplicates2
perform-subsumption-physical1
perform-subsumption-physical2
perform-subsumption-specification
perform-subsumption-or-rules1

perform-subsumption-or-rules2
cache-failed-check-conditions

File: obj-rule.lsp

Rule Set Definitions

Rule Set Rules

drain-rules

form-penetration-assertion1
add-to-drain-number1
add-to-drain-number2
assert-drainage-area-drain-overlap
form-roof-overflow-drain-assoc1
scupper-drain-assoc2

overflow-drain-rules

form-roof-overflow-drain-assoc2

vent-shaft-rules

form-penetration-assertion1

sump-rules

associate-drain-object

expansion-joint-ruleset

clip-to-roof-footprints
cover-structural-exp-joints

area-divider-ruleset

clip-to-roof-footprints

structural-ruleset

find-support-for-beams
find-center-for-columns
find-support-for-joists1
find-support-for-joists2
find-end-points-for-joists1
find-end-points-for-joists2

roof-footprints-ruleset

initialize-drain-number
initialize-roof-drainage-coverage-area

form-penetration-assertion2

form-penetration-assertion3

two-slope-das-ruleset

assert-complete-overlap-for-drainage-areas1

form-equipment-da-assertions1

one-slope-das-ruleset

subtract-drainage-area-from-roof-coverage1

subtract-drainage-area-from-roof-coverage2

assert-complete-overlap-for-drainage-areas2

assert-drainage-area-drain-overlap

four-slope-das-ruleset

subtract-drainage-area-from-roof-coverage1

subtract-drainage-area-from-roof-coverage2

assert-drainage-area-drain-overlap

walkway-rules

form-close-to-walkway-assertions2

form-adjacent-walkway-assertions

equipment-rules

form-equipment-da-assertions2

form-close-to-walkway-assertions1

scupper-ruleset

make-scupper-drain-assoc1

delete-roof-footprints-ruleset

delete-assoc-footprint-slices

delete-assoc-edges

delete-assoc-wall-segments

delete-roof-footprint-slices-ruleset

delete-roof-edge-ruleset

delete-wall-segment-ruleset

delete-wall-assoc1

delete-wall-assoc2

delete-wall-assoc3

deleted-column-line-ruleset
delete-col-line

Rule Definitions

delete-col-line
make-scupper-drain-assoc1
make-scupper-drain-assoc2
delete-wall-assoc1
delete-wall-assoc2
delete-wall-assoc3
delete-assoc-footprint-slices
delete-assoc-edges
delete-assoc-wall-segments
form-equipment-da-assertions1
form-equipment-da-assertions2
form-close-to-walkway-assertions1
form-close-to-walkway-assertions2
form-adjacent-walkway-assertions
find-center-for-columns
find-end-points-for-joists1
find-end-points-for-joists2
find-support-for-beams
find-support-for-joists1
find-support-for-joists2
form-penetration-assertion1
form-penetration-assertion2
form-penetration-assertion3
associate-drain-object
assert-complete-overlap-for-drainage-areas1
assert-complete-overlap-for-drainage-areas2
assert-drainage-area-drain-overlap
initialize-roof-drainage-coverage-area
subtract-drainage-area-from-roof-coverage1
subtract-drainage-area-from-roof-coverage2
initialize-drain-number
add-to-drain-number1
add-to-drain-number2
clip-to-roof-footprints
cover-structural-exp-joints
form-roof-overflow-drain-assoc1
form-roof-overflow-drain-assoc2

File: roof.lsp

Rule Definitions

ruleR-2-a-trigger
ruleR-2-a-condition
ruleR-2-b-trigger
ruleR-2-b-condition
ruleR-3-trigger
ruleR-3-condition

File: scuppers.lsp

ruleN-1-trigger
ruleN-1-condition
ruleN-2-trigger
ruleN-2-condition

File: vents.lsp

rule16-trigger
rule16-condition
rule11-trigger
rule11-condition
ruleV-1-trigger
ruleV-1-condition
ruleV-2-trigger
ruleV-2-condition
ruleV-3-trigger
ruleV-3-condition
ruleV-4-trigger
ruleV-4-condition
ruleV-5-trigger
ruleV-5-condition
ruleV-6-trigger
ruleV-6-condition
ruleV-7-trigger
ruleV-7-condition
ruleV-8-trigger
ruleV-8-condition
ruleV-9-trigger
ruleV-9-condition
ruleV-10-trigger

ruleV-10-condition
ruleV-11-trigger
ruleV-11-condition
ruleV-12-trigger
ruleV-12-condition
ruleV-13-trigger
ruleV-13-condition
ruleV-14-trigger
ruleV-14-condition
ruleV-15-trigger
ruleV-15-condition
ruleV-16-a-trigger
ruleV-16-a-condition
ruleV-16-b-trigger
ruleV-16-b-condition
ruleV-17-trigger
ruleV-17-condition
ruleI-1-5-13-trigger
ruleI-1-5-13-condition
ruleI-1-5-16-trigger
ruleI-1-5-16-condition

Appendix D: Alphabetical Listing of Goldworks III Lisp Functions

	Function Name	File Name
A	activate-rules-by-tasks	review.lsp
	adjacent	geometry.lsp
	adjacent-cc	geometry.lsp
	adjacent-on-edge	geometry.lsp
	adjacent-rc	geometry.lsp
	adjacent-rr	geometry.lsp
	align-adjust-width-rr	snap.lsp
	aligned	geometry.lsp
	all-frame-instances	cma-fn.lsp
	apply-lisp-functions	cma-fn.lsp
	area-of	geometry.lsp
	area-of-c	geometry.lsp
	area-of-rc	geometry.lsp
	area-of-rc*	geometry.lsp
	area-of-rc**	geometry.lsp
	assert-subtask-list	cma-fn.lsp
	assess-spacing	obj-fn.lsp
	assess-spacing*	obj-fn.lsp
	atomic-listp	violate.lsp
B	binding-list-match	kb.lsp
	border-order	geometry.lsp
	build-rule-object-type-list	prevdet.lsp
	build-rule-task-list	prevdet.lsp
C	cat-explanation-terms	violate.lsp
	cat-explanation-terms-2	violate.lsp
	check-all-borders	geometry.lsp

check-and-activate-tasklist	cma-fn.lsp
check-colinearity-overlap-segments	geometry.lsp
check-colinearity-overlap-segments*	geometry.lsp
check-corners	obj-fn.lsp
check-distance-intervals	obj-fn.lsp
check-intersection	prevdet.lsp
check-walkway-objects	geometry.lsp
checked-before	kb.lsp
checked-before-dual	kb.lsp
classify-corner	obj-fn.lsp
clear-all	cma-fn.lsp
clip-or-extend-rc-objects	obj-fn.lsp
clip-or-extend-to-roof	obj-fn.lsp
clip-or-extend-to-roof*	obj-fn.lsp
combine	cma-fn.lsp
combine-area-horizontally	decomp.lsp
combine-area-vertically	decomp.lsp
combine-horizontal-areas	decomp.lsp
combine-horizontal-areas*	decomp.lsp
combine-vertical-areas	decomp.lsp
combine-vertical-areas*	decomp.lsp
complete-expansion-joint-slots	obj-fn.lsp
complete-extent-overlap	geometry.lsp
complete-overlap	geometry.lsp
complete-overlap-cc	geometry.lsp
complete-overlap-rc	geometry.lsp
complete-overlap-rc*	geometry.lsp
complete-overlap-rr	geometry.lsp
complete-overlap-rr*	geometry.lsp
compute-distance	geometry.lsp
compute-distance-cc	geometry.lsp
compute-distance-rc	geometry.lsp
compute-distance-rr	geometry.lsp
connected?	geometry.lsp
connected?*	geometry.lsp
convert-from-string	cma-main.lsp
count	cma-fn.lsp
count-objects	cma-fn.lsp
create-low-point-drain	obj-fn.lsp

D	deep-convert-from-string	cma-main.lsp
	delete-assertions	cma-main.lsp
	detail-list-test	cma-fn.lsp
	determine-alignment	geometry.lsp
	determine-alignment-extent	geometry.lsp
	distance	geometry.lsp
	distance-to-corners	geometry.lsp
	distance-to-horiz-borders	geometry.lsp
	distance-to-line	geometry.lsp
	distance-to-vert-borders	geometry.lsp
	do-delete-object	cma-main.lsp
	do-delete-object*	cma-main.lsp
	do-get-dtm-activations	cma-main.lsp
	do-get-dtm-all	cma-main.lsp
	do-get-dtm-task-rules	cma-main.lsp
	do-get-dtm-task-status	cma-main.lsp
	do-get-dtm-tasks	cma-main.lsp
	do-get-object-children	cma-main.lsp
	do-get-object-parents	cma-main.lsp
	do-get-object-slot-defaults	cma-main.lsp
	do-get-object-slot-defaults*	cma-main.lsp
	do-get-object-slot-values	cma-main.lsp
	do-get-object-slot-values*	cma-main.lsp
	do-get-object-slots	cma-main.lsp
	do-horizontal-column-lines	obj-fn.lsp
	do-modify-slot-values	cma-main.lsp
	do-modify-slot-values*	cma-main.lsp
	do-move-object	cma-main.lsp
	do-place-object	cma-main.lsp
	do-reject-critique	cma-main.lsp
	do-resize-object	cma-main.lsp
	do-review-tasks	cma-main.lsp
	do-rule-activation	cma-main.lsp
	do-rule-query	cma-main.lsp
	do-select-object	cma-main.lsp
	do-set-critique-type	cma-main.lsp
	do-set-dtm-task-activation	cma-main.lsp
	do-set-review-type	cma-main.lsp
	do-vertical-column-lines	obj-fn.lsp

	draw-point	geometry.lsp
E	east-of	geometry.lsp
	east-of-extent	geometry.lsp
	edge-point-dist	geometry.lsp
	edgex1	geometry.lsp
	edgex2	geometry.lsp
	edgey1	geometry.lsp
	edgey2	geometry.lsp
	equal-extents	snap.lsp
	exceeds-max-distance-p	geometry.lsp
F	filter	cma-fn.lsp
	filter-deactivated-tasks	update.lsp
	filter-heights	geometry.lsp
	filter-mapcar	cma-fn.lsp
	filter-out-rectangles	decomp.lsp
	filter-recently-activated	update.lsp
	find-nearest-adjacent-edge	snap.lsp
	first-highest-priority-task	prevdet.lsp
	form-binding	prevdet.lsp
	form-binding*	prevdet.lsp
	form-complete-overlap-remainder	decomp.lsp
	form-one-corner-remainder	decomp.lsp
	form-one-side-remainder	decomp.lsp
	form-two-corner-remainder	decomp.lsp
	form-two-side-remainder	decomp.lsp
	frame-ancestor	cma-fn.lsp
	frame-ordered	cma-fn.lsp
	frame-ordered*	cma-fn.lsp
	frame-ordered**	cma-fn.lsp
G	get-active-rules	prevdet.lsp
	get-active-rules-for-object-place	prevdet.lsp
	get-all-dtm-activations	cma-main.lsp
	get-binding	violate.lsp
	get-bounding-points	obj-fn.lsp
	get-edge	geometry.lsp

	get-first-coord	geometry.lsp
	get-interfering-tasks	update.lsp
	get-last-tasks	update.lsp
	get-object-descriptions	cma-main.lsp
	get-object-string-assoc	violate.lsp
	get-other-object-string-assoc	violate.lsp
	get-other-object-string-assoc*	violate.lsp
	get-partial-action	violate.lsp
	get-partial-action*	violate.lsp
	get-relative-distance	geometry.lsp
	get-relative-distance-circle	geometry.lsp
	get-relative-distance-rect	geometry.lsp
	get-rule-info	cma-main.lsp
	get-second-coord	geometry.lsp
	get-tasks	update.lsp
	get-var-name	violate.lsp
H		
	horiz-edge-point-dist	geometry.lsp
	horizontal-decomposition	decomp.lsp
	horizontal?	geometry.lsp
I		
	inorder-traversal	cma-main.lsp
	instantiate	prevdet.lsp
	instantiate-binding	prevdet.lsp
	intersect-on-edge	geometry.lsp
	intersection	geometry.lsp
J		
K		
L		
	legal-composition	geometry.lsp
	legal-composition*	geometry.lsp
	legal-object?	cma-fn.lsp
	line-distance	geometry.lsp
M		
	make-all-focus-review-rules	review.lsp
	make-all-review-rules	review.lsp
	make-assoc-id	obj-fn.lsp

make-column-id	obj-fn.lsp
make-columns	obj-fn.lsp
make-coord-info	geometry.lsp
make-expansion-joint-points	obj-fn.lsp
make-explanation	violate.lsp
make-explanation-list	violate.lsp
make-extent	geometry.lsp
make-extent*	geometry.lsp
make-footprint-slices	obj-fn.lsp
make-footprint-slices*	obj-fn.lsp
make-h-slices	decomp.lsp
make-h-slices*	decomp.lsp
make-horizontal-borders	geometry.lsp
make-horizontal-borders*	geometry.lsp
make-joint-points*	obj-fn.lsp
make-joint-points-h**	obj-fn.lsp
make-joint-points-v**	obj-fn.lsp
make-line	obj-fn.lsp
make-new-antecedent	prevdet.lsp
make-new-consequent	prevdet.lsp
make-object-instance	cma-fn.lsp
make-object-list	violate.lsp
make-object-list	violate.lsp
make-object-place-ruleset	prevdet.lsp
make-object-select-ruleset	prevdet.lsp
make-od-rule	prevdet.lsp
make-penetration-id	obj-fn.lsp
make-review-rules	review.lsp
make-review-rules-for-task-subtree	review.lsp
make-roof-drains-id	obj-fn.lsp
make-roof-edges	obj-fn.lsp
make-roof-walls	obj-fn.lsp
make-roof-walls*	obj-fn.lsp
make-select-violation-action	violate.lsp
make-slice-id	obj-fn.lsp
make-v-slices	decomp.lsp
make-v-slices*	decomp.lsp
make-v-slices**	decomp.lsp
make-vertical-borders	geometry.lsp

make-vertical-borders*
 make-violation-action
 make-wall-segment-id
 make-wall-segments
 make-wall-segments*
 max-priority
 maximum
 maximum-decomposition
 minimum

geometry.lsp
 violate.lsp
 obj-fn.lsp
 obj-fn.lsp
 obj-fn.lsp
 prevdet.lsp
 cma-fn.lsp
 decomp.lsp
 cma-fn.lsp

N

n-last
 n-last*
 next-to-cc
 next-to-inside
 next-to-outside
 next-to-rc
 no-extent-overlap
 no-overlap
 no-overlap-cc
 no-overlap-rc
 no-overlap-rc*
 no-overlap-rr
 no-overlap-rr*
 north-of
 north-of-extent
 null?
 num-border-crossings
 num-intersecting-corners
 num-intersecting-corners*
 num-right-crossings

cma-fn.lsp
 cma-fn.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 geometry.lsp
 cma-fn.lsp
 geometry.lsp
 decomp.lsp
 decomp.lsp
 geometry.lsp

O

object-connected
 object-variable-p
 on-horizontal-border
 on-vertical-border
 one-corner-extent-overlap
 one-side-extent-overlap
 opposite-orientation
 order-by-tasks

geometry.lsp
 violate.lsp
 geometry.lsp
 geometry.lsp
 decomp.lsp
 decomp.lsp
 geometry.lsp
 violate.lsp

	order-by-tasks-select	violate.lsp
P	point-distance	geometry.lsp
	point-in-extent	geometry.lsp
	point-in-rect	geometry.lsp
	point-in-rect1	geometry.lsp
	point-strictly-in-extent	geometry.lsp
	pointx	geometry.lsp
	pointy	geometry.lsp
	pop-border-list	geometry.lsp
	pop-border-list*	geometry.lsp
	print-elapsed-time	cma-fn.lsp
Q		
R	recently-activated	cma-main.lsp
	recently-activated*	cma-main.lsp
	rect-comp-point-dist	geometry.lsp
	rect-comps-dist	geometry.lsp
	rect-comps-dist-aux	geometry.lsp
	rect-points-touch-circle	geometry.lsp
	rect-segments-touch-circle	geometry.lsp
	remove-nils	cma-main.lsp
	remove-rectangle	decomp.lsp
	rule-task-pair-compare	violate.lsp
S		
	segment-within-distance	geometry.lsp
	segment-within-distance*	geometry.lsp
	set-start-time	cma-fn.lsp
	simple-span-extent	geometry.lsp
	simple-span-extent	geometry.lsp
	simple-span-extent-rr	geometry.lsp
	simple-span-rr	geometry.lsp
	snap-drainage-area	snap.lsp
	snap-four-slope-das	snap.lsp
	snap-one-slope-das	snap.lsp
	snap-to-fit-circle-point	snap.lsp
	sort-by-priority-reverse	violate.lsp

	south-of	geometry.lsp
	south-of-extent	geometry.lsp
	spans-roof	geometry.lsp
	spans-roof*	geometry.lsp
	sqr	cma-fn.lsp
	string-member	violate.lsp
	subtract-area	decomp.lsp
	subtract-area*	decomp.lsp
T	task-compare	violate.lsp
	task-update-situation-p	cma-main.lsp
	test-cricket	obj-fn.lsp
	traversable	geometry.lsp
	two-corner-extent-overlap	decomp.lsp
	two-side-extent-overlap	decomp.lsp
U	update-assert	update.lsp
	update-kb	cma-main.lsp
	update-tasks	cma-main.lsp
	user::ac-message	cma-main.lsp
V	var-obj-instantiate	violate.lsp
	vert-edge-point-dist	geometry.lsp
	vertical-decomposition	decomp.lsp
	vertical?	geometry.lsp
	violation-name	cma-fn.lsp
W	west-of	geometry.lsp
	west-of-extent	geometry.lsp
	which-cricket	obj-fn.lsp

X**Y****Z**

Appendix E: Alphabetical Listing of Autolisp Functions

	Function	File
A	action-depends-on-object	ac-shell.lsp
	activate-task	ac-shell.lsp
	active-state-string	ac-shell.lsp
	add-if-not-null	ac-shell.lsp
	add-task	ac-shell.lsp
	all-spaces	ac-expl.lsp
	attribs-list-click-callback	attribs.lsp
B	build-constraint-block	ac-shell.lsp
	build-constraint-block-filter	ac-shell.lsp
	build-critique-layer-list	ac-shell.lsp
C	c:done	ac-shell.lsp
	call-gcl	ac-shell.lsp
	change-attribs	attribs.lsp
	change-object-slot-values	ac-shell.lsp
	change-slots	slots.lsp
	change-slots*	slots.lsp
	clean-up-rc	handlers.lsp
	clean-up-rc*	handlers.lsp
	clean-up-segment	handlers.lsp
	clear-violation	ac-shell.lsp
	clear-violation-aux	ac-shell.lsp
	compute-ww-rc	handlers.lsp
	compute-ww-side-1	handlers.lsp
	compute-ww-side-2	handlers.lsp
	continue-path-1	handlers.lsp
	continue-path-2	handlers.lsp

convert-da-to-rc	handlers.lsp
convert-pline-to-rc	handlers.lsp
create-ac-units-curbed	handlers.lsp
create-ac-units-sleeps	handlers.lsp
create-and-color-constraint-layer	ac-shell.lsp
create-and-color-constraint-layers	ac-shell.lsp
create-area-dividers	handlers.lsp
create-attic-vents	handlers.lsp
create-circular-object	handlers.lsp
create-column-lines	handlers.lsp
create-columns	handlers.lsp
create-exhaust-fans	handlers.lsp
create-exp-joints	handlers.lsp
create-fans	handlers.lsp
create-four-slope-das	handlers.lsp
create-hatches	handlers.lsp
create-hot-stacks	handlers.lsp
create-masonry-chims	handlers.lsp
create-mech-units	handlers.lsp
create-new-object	ac-shell.lsp
create-new-objects	ac-shell.lsp
create-one-slope-das	handlers.lsp
create-overflow-drains	handlers.lsp
create-power-vents	handlers.lsp
create-rectangular-object	handlers.lsp
create-roof-drains	handlers.lsp
create-roof-footprints	handlers.lsp
create-roof-vent-pipes	handlers.lsp
create-scuppers	handlers.lsp
create-two-slope-das	handlers.lsp
create-walkways	handlers.lsp
create-wall-segments	handlers.lsp

D deactivate-task	ac-shell.lsp
delete-from-dwg-object	ac-shell.lsp
delete-from-dwg-object*	ac-shell.lsp
delete-object	acad.mnl
delete-object-callback	ac-shell.lsp
draw-1slope	handlers.lsp

draw-2slope	handlers.lsp
draw-4slope	handlers.lsp
draw-arrow	ac-shell.lsp
draw-attention-text	ac-shell.lsp
draw-boundary-area	ac-shell.lsp
draw-circle-constraint	ac-shell.lsp
draw-constraint-action	ac-shell.lsp
draw-constraint-actions	ac-shell.lsp
draw-constraint-layers	ac-shell.lsp
draw-critique	ac-shell.lsp
draw-critique-explanation	ac-shell.lsp
draw-down	ac-shell.lsp
draw-explanation-box	ac-expl.lsp
draw-explanation-box-and-text	ac-expl.lsp
draw-explanation-text	ac-expl.lsp
draw-exterior-circle-constraint	ac-shell.lsp
draw-exterior-rc-constraint	ac-shell.lsp
draw-inside-down	ac-shell.lsp
draw-inside-left	ac-shell.lsp
draw-inside-rect-comp	ac-shell.lsp
draw-inside-rect-comp-aux	ac-shell.lsp
draw-inside-right	ac-shell.lsp
draw-inside-up	ac-shell.lsp
draw-interior-circle-constraint	ac-shell.lsp
draw-interior-rc-constraint	ac-shell.lsp
draw-layers-bounding-box	ac-shell.lsp
draw-left	ac-shell.lsp
draw-message-text	ac-expl.lsp
draw-message-text-at-line	ac-expl.lsp
draw-message-text-at-location	ac-expl.lsp
draw-outline	ac-shell.lsp
draw-outside-rect-comp	ac-shell.lsp
draw-outside-rect-comp-aux	ac-shell.lsp
draw-rc-constraint	ac-shell.lsp
draw-rect-comp	ac-shell.lsp
draw-rect-comp-aux	ac-shell.lsp
draw-right	ac-shell.lsp
draw-shadow-object	ac-shell.lsp
draw-thinking-text	ac-shell.lsp

	draw-up	ac-shell.lsp
E	eighth	ac-shell.lsp
	erase-message-window	ac-expl.lsp
	erase-shadow-layer	ac-expl.lsp
F	fifth	ac-shell.lsp
	filter	ac-shell.lsp
	find-layers-bounding-box	ac-shell.lsp
	find-max-min	handlers.lsp
	find-nearest-point	ac-shell.lsp
	find-nearest-point-aux	ac-shell.lsp
	find-next-ww-point-1	handlers.lsp
	find-next-ww-point-2	handlers.lsp
	first	ac-shell.lsp
	fits	ac-expl.lsp
	forget-object-constraint	ac-shell.lsp
	forget-object-constraints	ac-shell.lsp
	forget-object-constraints-aux	ac-shell.lsp
	fourth	ac-shell.lsp
	freeze-critique-layers	ac-shell.lsp
	freeze-layers	ac-shell.lsp
	freeze-layers*	ac-shell.lsp
G	generate-constraint-block	ac-shell.lsp
	gensym	ac-shell.lsp
	get-active-state	ac-shell.lsp
	get-ac-units-curbed	handlers.lsp
	get-ac-units-sleeps	handlers.lsp
	get-area-dividers	handlers.lsp
	get-attic-vents	handlers.lsp
	get-circular-object	handlers.lsp
	get-color-from-violation-type	ac-shell.lsp
	get-column-lines	handlers.lsp
	get-critique-action	ac-shell.lsp
	get-critique-explanation	ac-shell.lsp
	get-da	handlers.lsp
	get-direction	handlers.lsp

get-dwg-object-center	ac-shell.lsp
get-dwg-object-entity	ac-shell.lsp
get-dwg-object-from-entity	ac-shell.lsp
get-dwg-object-from-entity*	ac-shell.lsp
get-dwg-object-radius	ac-shell.lsp
get-dwg-object-type	ac-shell.lsp
get-dwg-object-vertices	ac-shell.lsp
get-exhaust-fans	handlers.lsp
get-exp-joints	handlers.lsp
get-explanation-list	ac-shell.lsp
get-fans	handlers.lsp
get-first-word	ac-expl.lsp
get-first-word-aux	ac-expl.lsp
get-four-slope-das	handlers.lsp
get-hatches	handlers.lsp
get-hot-stacks	handlers.lsp
get-leading-whitespace	ac-expl.lsp
get-masonry-chims	handlers.lsp
get-mech-units	handlers.lsp
get-new-object	ac-shell.lsp
get-new-object-type	ac-shell.lsp
get-object-constraint-layers	ac-shell.lsp
get-object-constraint-layers*	ac-shell.lsp
get-object-list	ac-shell.lsp
get-object-shape	ac-shell.lsp
get-one-slope-das	handlers.lsp
get-ortho-pline	handlers.lsp
get-overflow-drains	handlers.lsp
get-pline-vertices	handlers.lsp
get-power-vents	handlers.lsp
get-rc	handlers.lsp
get-rectangular-object	handlers.lsp
get-roof-drains	handlers.lsp
get-roof-footprints	handlers.lsp
get-roof-vent-pipes	handlers.lsp
get-rule-type	ac-shell.lsp
get-scuppers	handlers.lsp
get-some-violations	ac-shell.lsp
get-some-violations	ac-shell.lsp

	get-two-slope-das	handlers.lsp
	get-violation-level	ac-shell.lsp
	get-walkways	handlers.lsp
	get-walls	handlers.lsp
	get-ww	handlers.lsp
	get-x-maxmin	ac-shell.lsp
	get-x-maxmin*	ac-shell.lsp
	get-y-maxmin	ac-shell.lsp
	get-y-maxmin*	ac-shell.lsp
H	hierarchic-stringify	ac-shell.lsp
	hierarchic-stringify*	ac-shell.lsp
	hierarchic-stringify-children	ac-shell.lsp
I	id	globals.lsp
	in-all-3-regions	ac-expl.lsp
	in-bottom-third	ac-expl.lsp
	init-log-file	ac-init.lsp
	in-middle-third	ac-expl.lsp
	input-pline	handlers.lsp
	in-top-third	ac-expl.lsp
	is-clockwise	handlers.lsp
	is-clockwise*	handlers.lsp
J		
K		
L	lower-left-of-exp-box	ac-expl.lsp
M	make-clockwise	handlers.lsp
	make-message-window-blue	ac-expl.lsp
	make-rc-start-right	handlers.lsp
	map-shadow-point	ac-shell.lsp
	map-shadow-points	ac-shell.lsp
	mk_list	ac-shell.lsp
	move-object	acad.mnl
	move-object-callback	ac-shell.lsp
	my-textbox	ac-expl.lsp

N	new-object	acad.mnl
	new-object-callback	ac-shell.lsp
	ninth	ac-shell.lsp
	normalize-rc	handlers.lsp
O	object-list-click-callback	ac-shell.lsp
	object-name-p	ac-expl.lsp
	offset	handlers.lsp
	offset-layers-bounds	ac-shell.lsp
	on-state-string	ac-shell.lsp
	open-roof-layout	ac-shell.lsp
P	perform-critique-actions	ac-shell.lsp
	perform-review-actions	ac-shell.lsp
	perform-task-activations	ac-shell.lsp
	perform-task-activations-aux	ac-shell.lsp
	physical-violations-list-click-callback	ac-shell.lsp
	pin-mouse-point	handlers.lsp
	position	ac-shell.lsp
	position-aux	ac-shell.lsp
	preference-violations-list-click-callback	ac-shell.lsp
Q	process-explanation-list	ac-expl.lsp
R	resize-object	acad.mnl
	resize-object-callback	ac-shell.lsp
S	S::STARTUP	setup.lsp
	save-globals	ac-shell.lsp
	save-roof-layout	ac-shell.lsp
	second	ac-shell.lsp
	seventh	ac-shell.lsp
	shift-coordinates	ac-shell.lsp
	show-constraints	ac-shell.lsp
	show-critique	ac-shell.lsp
	show-text	ac-expl.lsp
	sixth	ac-shell.lsp

	slots-list-click-callback	slots.lsp
	specification-violations-list-click-callback	ac-shell.lsp
	stringify	ac-shell.lsp
	stringify-elements	attribs.lsp
	stringify-elements	slots.lsp
	stringify-pairs	attribs.lsp
	stringify-pairs	slots.lsp
	str-to-sym	ac-expl.lsp
	subactions-depend-on-object	ac-shell.lsp
	suggestions-callback	ac-shell.lsp
T	tasks-callback	ac-shell.lsp
	tasks-list-click-callback	ac-shell.lsp
	thaw-critique-layers	ac-shell.lsp
	thaw-layers	ac-shell.lsp
	thaw-layers*	ac-shell.lsp
	third	ac-shell.lsp
	toggle-debiasser-mode	ac-shell.lsp
	toggle-influencer-mode	ac-shell.lsp
	trim-leading-whitespace	ac-expl.lsp
	turn-task-off	ac-shell.lsp
	turn-task-on	ac-shell.lsp
	turn-value	handlers.lsp
U	update-current-slot	slots.lsp
	upto	ac-shell.lsp
V	view-violation-callback	ac-shell.lsp
	violation-message	ac-shell.lsp
	violations-callback	ac-shell.lsp
	violation-types	ac-shell.lsp
W	wrap-rc-coords	handlers.lsp
X		
Y		
Z		

USACERL DISTRIBUTION

Chief of Engineers

ATTN: CEHEC-IM-LH (2)

ATTN: CEHEC-IM-LP (2)

ATTN: CECC-R

ATTN: CEMP-C

ATTN: CEMP-CE (2)

ATTN: CEMP-E

ATTN: CEMP-ES (2)

ATTN: CERD-L

US Army Engr District

ATTN: Library (40)

ATTN: Civil Engineers (40)

US Army Engr Division

ATTN: Library (11)

ATTN: Civil Engineers (11)

ATTN: Civil Construction/Civil Con-Ops (11)

Defense Tech Info Center 22060-6218

ATTN: DTIC-O (2)

127
7/96