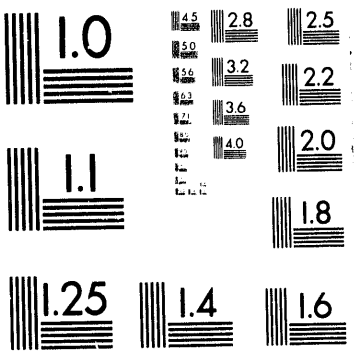

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0

1 of 2

Integrated Computer-Enhanced Remote Viewing System

Quarterly Report Number 2
for
January - March 1993

May 3, 1993

Work Performed under
Contract No.:DE-AC21-92MC29113

For:
U.S. Department of Energy
Morgantown Energy Technology Center
3610 Collins Ferry Road
Morgantown, West Virginia 26507

By:
Mechanical Technology Incorporated
968 Albany-Shaker Road
Latham, New York 12110
Tel No. (518) 785 - 2800

On these grounds, MTI recommends without reservation that the ICERVS project be continued in Phase 2.

6. Appendices

A. Phase 1 Task Descriptions

B. Subsystem Design Report, Phase 1

Appendix A. ICERVS Task Description

The eight tasks in this Phase and the accomplishments involved are described below.

Task 1. National Environmental Policy Act (NEPA) documentation.
MTI will prepare a report containing the environmental, health, and safety information for the NEPA documentation which the DOE must submit for this project. During this Phase, the project efforts are confined to a laboratory where the primary activities involve the use of computers. Hence only the OSHA aspects will be involved.

Task 2. Preliminary design.
MTI will design, analyze, and identify appropriate implementations of the four major subsystems, namely, the Sensor, Data Library, Toolkit, and Computational subsystems. The design of these subsystems will reflect the current understanding of the needs of the user community.

Task 3. Computing Platform and Rapid Prototyping
MTI will provide a workstation to be used as the computer platform for this phase of the effort. A 2D database will be developed as a rapid prototyping tool to validate design approaches for the two subsystems. This task will conclude with a design review.

Task 4. Detailed Design
MTI will detail the preliminary design, providing means to store empirical 3D geometric data (the Volumetric Database), a means to display the data (Display Engine), and the software tools (Model Building) necessary to create and manipulate models of objects. This task will conclude with a design review.

Task 5. Data Library Subsystem
MTI will construct software modules to implement the detailed design of the Data Library subsystem. These modules will store the empirical 3D geometric data in the volumetric database, the wire frame models used in the world model and maintain the associated data in the object files.

Task 6. Toolkit Subsystem
MTI will also construct software modules for the Display Engine, which will retrieve and display the data in the volumetric database and in the world model in a fashion that exploits the speed and efficiency of the octree technology. A set of Model Building modules will provide the operator with the tools to define a region of interest in a scene and create a model of the object s/he perceives there. These models will be stored in the world model and the volumetric database.

Task 7. Demonstrate Data Library and Toolkit Subsystems
MTI will integrate the software modules developed in Tasks 5 and 6 in the Computational subsystem. A demonstration will be made of the basic

functionality incorporated in these two subsystems, namely, the ability to accept, store, and retrieve empirical geometric data rapidly and accurately and to enable an operator to create, store, display and manipulate models of objects that are perceived in the database.

Task 8. Topical Report and Decision Point
MTI will submit a comprehensive report on the technical results achieved during this phase of the project. The Contracting Officer will decide whether to continue on with the next phase.

1. Introduction

The Interactive, Computer-Enhanced, Remote Viewing System (ICERVS) is a system designed to provide a reliable geometric description of a robotic task space in a fashion that enables robotic remediation to be carried out more efficiently and economically that with present systems. The key elements are a faithful way to store empirical data and a friendly user interface that provides an operator with timely access to all that is known about a scene.

ICERVS will help an operator to analyze a scene and generate additional geometric data for automating significant portions of the remediation activity. Features that enable this include the following:

- Storage and display of empirical sensor data,
- Ability to update segments of the geometric description of the task space,
- Side-by-side comparisons of a live TV scene and a computer generated view of the same scene,
- Ability to create and display computer models of perceived objects in the task space, together with textual comments, and
- Easy export of data to robotic world models for robot guidance.

The development of ICERVS is to occur in three phases.

- Phase 1 will focus on the development of the Data Library, which contains the geometric data about the task space and the objects in it, and the Toolkit, which includes the mechanisms for manipulating and displaying both empirical and model data.
- Phase 2 will concentrate on integrating these subsystems with a sensor subsystem into one working system. Some additional functionality will be incorporated in the Data Library and Toolkit subsystems.
- Phase 3 will expand the configuration to meet the needs of a full scale demonstration of the interactive mapping of some waste site to be identified.

We are currently about two thirds of the way through Phase 1.

2. Summary of Progress to Date

The Phase 1 activity consists of eight tasks which are described in Appendix A. The first task was to provide the required National Environmental Policy Act documentation. This was readily accomplished because only computer related activities are involved in this Phase.

The Preliminary Design task developed a conceptual design for the system. This was documented in the System Design Report which was included in the last Quarterly Report.

The Computer Platform and Rapid Prototyping task is essentially a risk reduction effort to validate our approach to critical design issues before embodying them in a detailed design. A two dimensional quadtree was used in lieu of an octree to verify basic data library procedures. A commercial software package was acquired to provide basic user interface functions. This task was completed in the first month of the quarter.

The Detailed Design task began early in the quarter and addressed three major computer software components (CSCs), namely the Octree Engine CSC, the Object Modeling CSC, and the User Interface CSC. Detailed design was done in an object-oriented fashion and the results were documented in a Subsystems Design Report, which is included as Appendix B. The Task and quarter ended with a Critical Design Review which elicited several suggestions for incorporation during Phase 2.

Coding was begun on the Data Library and the Toolkit Tasks as the quarter ended.

3. Progress Report

3.1 Rapid Prototype

This third project Task was aimed at reducing risk in the Detailed Design Task by verifying the workability of certain design choices. One set of these choices was associated with the octree technology and these were validated using a two dimensional equivalent of the octree. The second set of choices pertained to the software for the basic user interface. Class libraries in two software packages from Rogue Wave (Tools.h++ and Views.h++) were validated as providing the necessary basis for the design functionality that was desired.

3.2 Detailed Design

The Detailed Design in Phase 1 focused on three CSCs: Octree Engine, Object Modeling, and User Interface. The design began with an allocation of the requirements listed in the System Design Report to each of the various CSCs. The allocation criterion was to group the functionalities so as to minimize interaction between the groups, in order to expedite debug, integration, and any future modification. Study of the requirements allocated to the User Interface CSC led to the perception that its functions should be restricted solely to the interaction with the operator and to the display of elements prepared by other CSCs. At this time, the separation is not as complete as desired and so there are computer software units (CSUs) in the User Interface CSC associated with the interfaces with the Octree Engine CSC and with the Object Modeling CSC. (It is also recognized that the User Interface cannot be designed without input from real operators. Hence, a goal of the Phase 1 design is a strawman User Interface which can be presented to real operators in order to secure a more realistic definition of the user needs and the interface requirements.)

Study of the requirements allocated to the Octree Engine CSC led to a generalization of the input means, to include data from other dimensional sensors, such as the laser range finder, and from material property sensors, such as temperature probes or magnetometers. A variety of sensors and data structures need to be accommodated, all of which begin to define the functionality to be included in the Data Interface CSC and the Sensor Interface CSC, which are to be designed in the next Phase. From this flowed the realization that it is necessary to define the units to be used for internal representations of the data and to provide means to convert and register different data sets.

In considering the details of the design for the Object Modeling CSC, it became clear that GL (Graphics Language) is much more a set of procedures than it is a format or language. Once this became apparent, the value of GL dropped significantly. This was important when it was discovered that the Rogue Wave software has a number of incompatibilities with GL. Because of the value of the Rogue Wave user interface software, the requirement for GL models was modified, replacing it with other formats such as IGES (Initial Graphics Exchange Specification) or STEP. For Phase 1, the graphics capability of Rogue Wave will be used.

The actual software design was rather straightforward. For each CSCV. the allocated set of requirements was examined and a set of implicit requirements was derived. For this combined set of requirements, a design was created, classes were defined, functionalities enumerated, and included data specified. Interaction between the classes was described and diagrammed. As the design neared completion, a mapping of classes with requirements was created to insure that no requirements were omitted and no extraneous classes were created. It was verified that all pertinent requirements have been satisfied with a minimal set of classes. The result is a detailed design for the three CSCs which has been documented in the Subsystem Design Report.

A Critical Design Review was conducted at the end of the Detailed Design Task. The review team consisted of six people, three of whom were DOE personnel:

Shawn Bohn, Battelle Pacific Northwest Laboratories
Fred DePiero, Oak Ridge National Laboratories
Gary Nelkin, Morgantown Energy Technology Center

and three of whom were MTI personnel:

James Dill
Gordon Hirschman
Thomas Walter.

Shawn represented the Hanford site and Fred represented the Robotics program in OTD. Gordon is a senior investigator who is very skilled in software design. The review was a lively one and many suggestions were offered for Phase 2 improvements. The only Phase 1 comments pertained to

the choice of names; most of the recommended changes have been incorporated in the Subsystem Design Report found in Appendix B.

3.3 Testing of True Solid

One recommendation from the Preliminary Design Review (held December, 1992) was that True Solid, a software package offered by the Octree Corporation, be considered for use in Phase 2. The package offers many attractive features, some of which MTI had planned to develop in Phase 2. However, there are questions about its compatibility with the object modeling requirements, which only testing can resolve. Investigation revealed that the current True Solid configuration will operate on some workstation platforms, but not on the Silicon Graphics Inc. (SGI) products. MTI received authorization to have Octree port the True Solid product to the SGI machine and to provide MTI with an evaluation copy for three months. Octree is currently carrying out the necessary recoding and expects to have the evaluation copy by the latter part of May. Features to be evaluated include its ability to draw and display a wide range of geometric (modeling) objects and the speed with which model updates can be displayed. The evaluation of True Solid will begin in the last part of the present quarter.

4. Plans for Future Activity

Phase 1 of the ICERVS project is on schedule and budget and will be essentially completed by the end of the next quarter. Completion involves the following items.

- Coding and debugging of the Data Library and Toolkit Subsystems.
- Integration into the Phase 1 prototype and debug,
- Completion of the acceptance tests, per the test plan,
- Phase 1 Demonstration,
- Submission of the comprehensive Phase 2 proposal.

In accordance with the contract modification, the Phase 2 proposal will be submitted about the middle of May.

5. Assessment of Prospects

The results from the first two quarters are quite encouraging. Both the Preliminary Design Review and the Critical Design Review concluded that the design was a solid one, with no major risks identified or anticipated. The design has proceeded largely as expected, with only a few minor modifications resulting from the greater insight developed with time. There is high confidence that the Phase 1 demonstration will display all the features and functionality to be delivered in the Phase 1 design.

In addition, discussion with field site personnel has shown that the perceived need for ICERVS is real and is growing . Furthermore, contact with commercial vendors indicates that there is yet no credible competition.

ICERVS
Subsystem Design Report
Phase 1

Prepared for
Department of Energy
Morgantown Energy Technology Center

Prepared under
Contract DE-AC21-92M29113

April 27, 1993

Integrated Computer Enhanced Remote Viewing System (ICERVS)

Prepared for
Department of Energy
Morgantown Enegy Technology Center

Prepared under
Contract DE-AC21-92M29113

Prepared by
Mechanical Technology Inc.
968 Albany-Shaker Road
Latham New York 122110
April 27,1993

Prepared by:	David Smith	4/27/93 Date
Approved by:	John Wagner	4/29/93 Date

REVISION HISTORY

Revision	Date	Comment
A	April 2, 1993	Original Version
B	April 27. 1993	Revised Requirements added

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

2D —
Geometric Object: a geometric element that occupies conceptual space of dimensionality 2. Examples of 2D geometric objects include circles, rectangles, and polygons. A 2D geometric object lies in a plane, occupies surface area, but does not occupy volume.

Polygon: a geometric object characterized by a sequence of connected vertices lying in a plane. The connections between adjacent vertices are edges. A 2D polygon is *convex* if a line segment joining any two interior points is completely contained within the polygon.

2.5 D Surface Map: a mathematical description of a 3D region bounded by a single surface in one dimension. A surface map is typically represented by a set of 3D data points.

3 D —
Geometric Object: a geometric element that occupies conceptual space of dimensionality 3. Examples of 3D geometric objects include spheres, rectangular parallelepipeds, and prisms. A 3D geometric object occupies volume.

Polyhedral Object: a 3D geometric object bounded by plane faces.

Swept Volume: the process of creating a 3D geometric object by projecting a 2D geometric object through a path in 3D space.

Class, C++ —
Base Class: a class which is included in another class(es). A base class typically provides general-purpose characteristics that are tailored by the derived class(es).

Class: a C++ data type defined by the programmer, that aggregates programmer-defined data structures (or member data), member functions, and custom operators.

Derived Class: a class defined such that it includes the member data and member functions from another class. The derived class is said to inherit the characteristics of the other class.
Member Data: the variables and data structures defined within a class (including those inherited from other classes).

Member Function: the functions defined within a class (including those inherited from another class).

CSC: Computer Software Component.

CSCI: Computer Software Configuration Item

GLOSSARY (continued)

CSU: Computer Software Unit

Cut Plane: a plane used to segment 3D space into two regions. Typically cut planes are used in pairs to bound a region of interest.

Dialog Box: an area on the user display for user input/output.

ICERVS: Integrated Computer-Enhanced Remote Viewing System

Level: for a node in a tree, the number of descendants in the direct path from the node to the root node.

Model —
Geometric Model: a computer representation of a physical object.

Task Space: a region in physical space that the ICERVS is being used to model.

World Model: the ICERVS data representations (geometric models and octrees) used to represent a task space.

Node: one data element within a tree data structure

Child Node: a direct descendant of another node.

Leaf Node: a node with no children. Leaf nodes have empty, full, or unknown states.

Node State: for octrees, a flag used to describe the occupancy of the region of space corresponding to a node. The four node states are:

Empty - none of the region is occupied

Partial - some (but not all) of the region is occupied. The occupancy is described in further detail by children nodes

Full - all of the region is occupied

Unknown - the occupancy of the region is not known

GLOSSARY (continued)

Object—
Geometric Object: a computer representation of a physical object. A geometric object typically represents a contiguous region of conceptual space and is defined in a mathematical form(s). Geometric objects can be two dimensional (e.g. circle, rectangle, and polygon) or three dimensional (e.g. sphere, cube, and prism).

Group: a composite set of geometric objects that are treated as a single geometric object.

Octree: an 8-ary tree data structure that represents the spatial occupancy of a 3-dimensional region. The data structure is produced by the recursive subdivision of a finite cubical universe.

OE-CSC: Octree Engine CSC

Polygon: a geometric object characterized by a sequence of connected vertices lying in a plane. The connections between adjacent vertices are edges. A polygon that satisfies the condition that a line segment joining any two interior points is completely contained within the polygon.

Polyhedron: a 3D geometric object bounded by plane faces.

Primitive/Geometric Primitive: a 3D geometric object typically defined by analytic description (equations). Examples of primitives include spheres, cylinders, and planes (half-spaces).

Prismoid: a polyhedron that has all of its vertices in two parallel planes, and with the same number of edges in each plane.

Property Data: for octrees, data used to describe the physical characteristics of a region in space corresponding to one node. Example property data include temperature, conductivity, and color.

Quadtree: a 4-ary tree data structure that represents the spatial occupancy of a 2-dimensional region. The data structure is produced by the recursive subdivision of a finite square universe.

Scale: for computer graphic display, the size of the computer-modeled space that is rendered in a display window. For ICERVS, scale is described as a percentage of full scale (At full scale, the entire computer-modeled region is made to exactly fit in the display window.)

Sculpting: for octrees, the process of changing node states to empty along a selected path or within a selected region. Sculpting is typically used to clear regions corresponding to input data provided by line-of-sight sensors.

Translate: for computer graphic display, the apparent position of the display window in the computer-modeled space. Translation provides the means for viewing a subsection of the world model at a magnified scale. For ICERVS, translation is described as screen-horizontal and screen-vertical displacements in internal units.

Tree: a data structure characterized by a hierarchy of elements or nodes descendant from a single or root node.

Tree —
Tree Traversal: the process of retrieving each node from a tree in a prescribed order. Typically each node is retrieved one time. A *depth-first* tree traversal retrieves the descendants of a node before retrieving the siblings of a node.

Units —
External Units: the physical units selected by the ICERVS user for display and data input/output.

Internal Units: the physical units used by the ICERVS User Interface CSC and Object Modeling CSC. For ICERVS, these units are SI (dimensions in meters).

Tree Units: the dimensional units used by the Octree Engine CSC to describe octree space. For ICERVS these dimensions vary between zero and 2,097,152 (2^{21}). This also sets the maximum spatial resolution of the octree.

UI-CSC: User Interface CSC

Wireframe: a method for defining, or generating a graphic display of, a polyhedron as a set of points and connecting edges.

1.0 INTRODUCTION

This ICERVS Subsystem Design Report describes the detailed design of the Software Subsystem for the Phase 1 Integrated Computer-Enhanced Remote Viewing System (ICERVS). The Software Subsystem design is based on the ICERVS system design described in ICERVS System Design Report, December 21, 1993. The Software Subsystem consists of a single Computer Software Configuration Item (CSCI) which is made up of seven Computer Software Components (CSC). The design presented here is restricted to the three CSCs which are needed to meet the contractual commitments of the Phase 1 work effort.

2.0 RELATED DOCUMENTS

MTI technical proposal Q2-030, "*Interactive Computer Enhanced Remote Viewing System*", December 13, 1991.

MTI "*ICERVS System Design Report*", December 21, 1992.

MTI ICERVS "*Software Development Plan*", December 23, 1992.

Rogue Wave Software "*Views.h++, MouseWrapper.h++ Class Libraries for Motif 1.0*"

3.0 PRELIMINARY DESIGN

The Interactive Computer-Enhanced Remote Viewing System (ICERVS) combines volumetric mapping and 3-D computer modeling technologies to provide reliable geometric descriptions of an environment. The purpose of ICERVS is to provide operator assistance in the analysis of remote scenes and in the generation of volumetric data for use in robotic task planning, programming, and execution. The intended use of ICERVS will be in the remediation of hazardous work sites such as underground waste storage tanks, buried waste sites, and contaminated production facilities.

As part of the ICERVS preliminary design, mission profiles were described for three remediation tasks. These profiles are documented in the ICERVS System Design Report (December 21, 1992) Section 3. From these mission profiles, a list of ICERVS primary requirements were developed. These requirements are documented in ICERVS System Design Report (December 21, 1992) Section 3 and summarized in Table 3-1 below. In the course of detailing the subsystem design, it became apparent that some minor revisions in the requirements were necessary in order to clarify points or to reflect the greater insight into the roles of the various subsystem elements. The details of the revisions to the requirements will be found in Appendix A.

The ICERVS system-level design is documented in ICERVS System Design Report (December 21, 1992) Section 5. The following sections are excerpted from that section.

3.1 System Diagrams

The system context diagram shown in figure 3-1 depicts the relationship between the remote viewing subsystem, the computation subsystem and the Data library subsystem.

The system block diagram shown in figure 3-2 shows the relationships between the major system components. There are three subsystems: sensor subsystem, computational subsystem, and software subsystem. The software subsystem is supported by the computational subsystem. The sensor subsystem feeds data into and accepts commands from the computational subsystem.

Table 3-1 ICERVS Requirements

Table 3-1 ICERVS Requirements

NO	SYSTEM REQUIREMENT	PHASE 1 REQ'MENT	NOTES
R1	DATA REPRESENTATION		
R1.01	Octree: spatial data	Full	
R1.02	Octree: property data	None	Design influence
R1.03	Octree: spatial interpolation	None	Design influence
R1.04	Octree: linear res'n 1:512, expandable	Full	
R1.05	Geom: polyhedral objects	Full	
R1.06	Geom: geometric primitives	None	Design influence
R1.07	Geom: associated text each object	Full	
R1.08	Geom: 100 objects, expandable	Full	
R1.09	Geom: enter architectural and robot plans	None	
R2	OBJECT MODELING		
R2.01	Library of primitives/templates	None	
R2.02	Standard templates	None	
R2.03	User-defined templates	None	
R2.04	Automatic waste surface modeling	None	Will demonstrate interactively
R2.05	Synthesize 2D polygons	Full	
R2.06	Synthesize 3D polyhedra	Partial	Straight line sweep path
R2.07	Dimensioning tools	None	Design influence
R2.08	Attach text to objects	Full	
R3	COMPUTER GRAPHICS DISPLAY		
R3.01	Translation and scaling	Full	
R3.02	Display coordinate axes	None	Design influence
R3.03	Parallel cut planes	Partial	1 plane parallel to display
R3.04	Display object text data	Full	
R3.05	Shaded or wire frame polygons	Partial	Wire frame only
R3.06	Update octree as points received as input	Full	
R3.07	Pseudo-color octree data	None	Design influence
R3.08	Property derived color	None	Design influence
R3.09	Text display view parameters	None	
R3.10	Save/Recall view parameter set	None	
R3.11	Multiple windows displaying same data	Full	
R3.12	View tracks sensor station attitude	None	Design influence
R3.13	Display 2.5D surface map	None	
R4	VIDEO DISPLAY		
R4.01	Monitor each camera plus 1 for processing	None	
R4.02	Display wire frame objects over video	None	
R5	MANIPULATION AND ANALYSIS		
R5.01	Copy octree	None	
R5.02	Set region within octree to selected state	None	Design influence
R5.03	Operator select and delete objects	None	

Table 3-1 ICERVS Requirements (cont)

R5.04	Scan objects for consistency with octree	None	
R5.05	Compare octree and object data	None	
R5.06	Compare 2 octrees, compute difference	None	
R5.07	Compute 2.5D surface map from octree	None	
R5.08	Compare 2.5D surface maps, compute diff	None	
R6	MISCELLANEOUS FUNCTIONS		
R6.01	Edit system parameters	Partial	Simple version for phase 1
R6.02	Save/Retrieve models to/from disk	Full	
R6.03	Build octree from backup raw data	None	
R6.04	Maintain operator log	None	Design influence
R6.05	Multiple systems of units	None	Design influence
R6.06	Define disassembly data	None	
R7	DATA INTERFACE		
R7.01	Input: x,y,z position	Full	
R7.02	Input: optional resolution	None	Design influence
R7.03	Input: optional property values	None	Design influence
R7.04	Input: optional sensor location	None	Design influence; assume sensor at infinity
R7.05	Input: station angles during vis. inspection	None	Design influence
R7.06	Output: geometric model data	Partial	Text report
R7.07	Output: 2.5D surface map data	None	
R8	OPERATOR INTERFACE		
R8.01	Graphic tools	Partial	Spaceball or mouse
R9	SENSORS		
R9.01	Tele-operation position and rate commands	None	Design influence
R9.02	Tele-operation display line of sight	None	
R9.03	Tele-operation text display station angles	None	
R9.04	Automatically map surfaces	None	
R9.05	Operator parameters	None	
R9.06	Draw/Display scan paths	None	
R9.07	Continual backup raw data	None	
R9.08	Flexible color TV camera	None	
R9.10	Surface mapping sensor	None	Design influence
R9.11	Sensor performance	None	
R10	SITE ENVIRONMENT		
R10.1	Surface characteristics	None	
R10.2	Illumination and visibility	None	
R10.3	Environmental constraints	None	
R10.4	Design constraints	None	

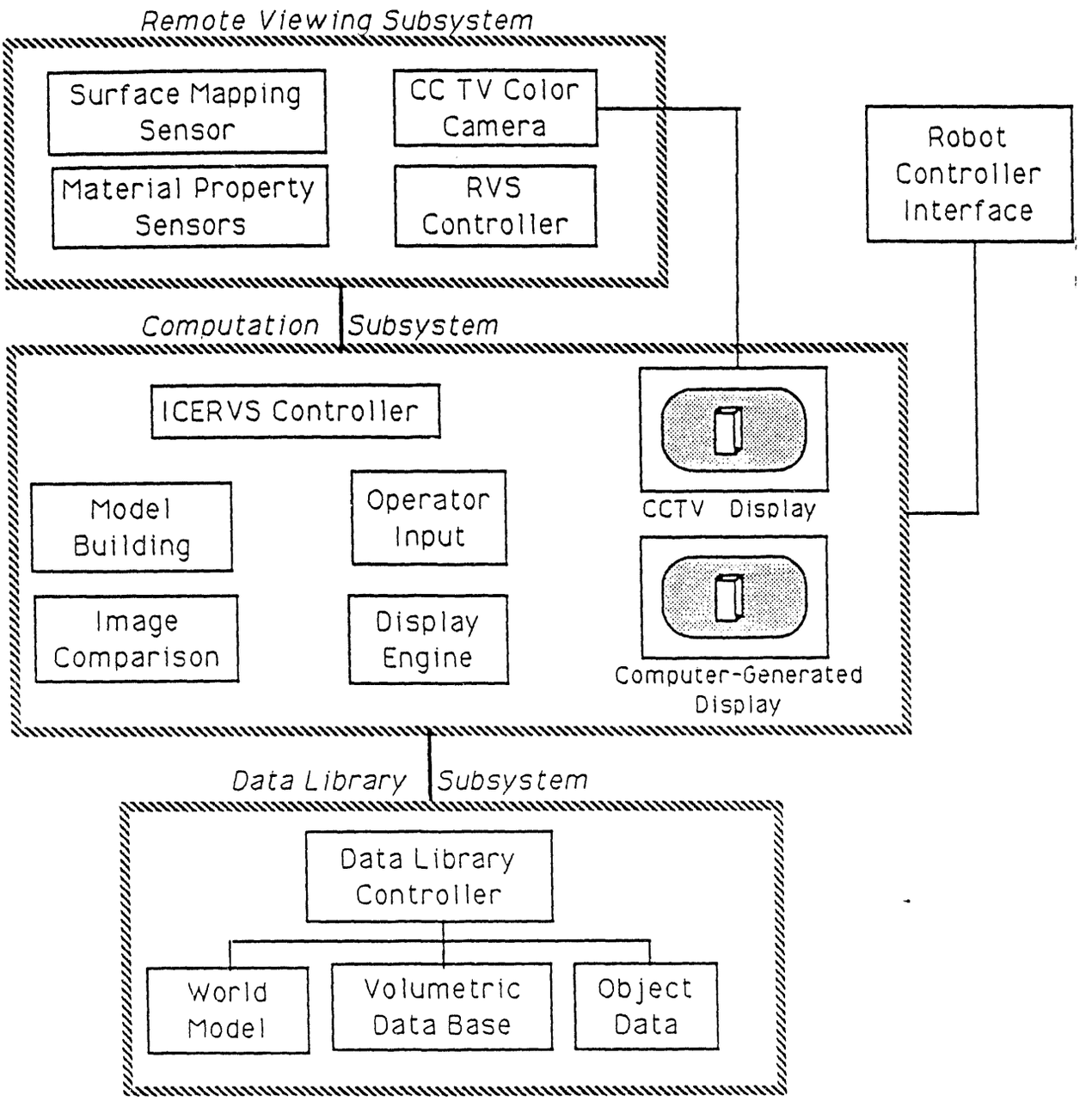


Figure 3-1 — System context diagram

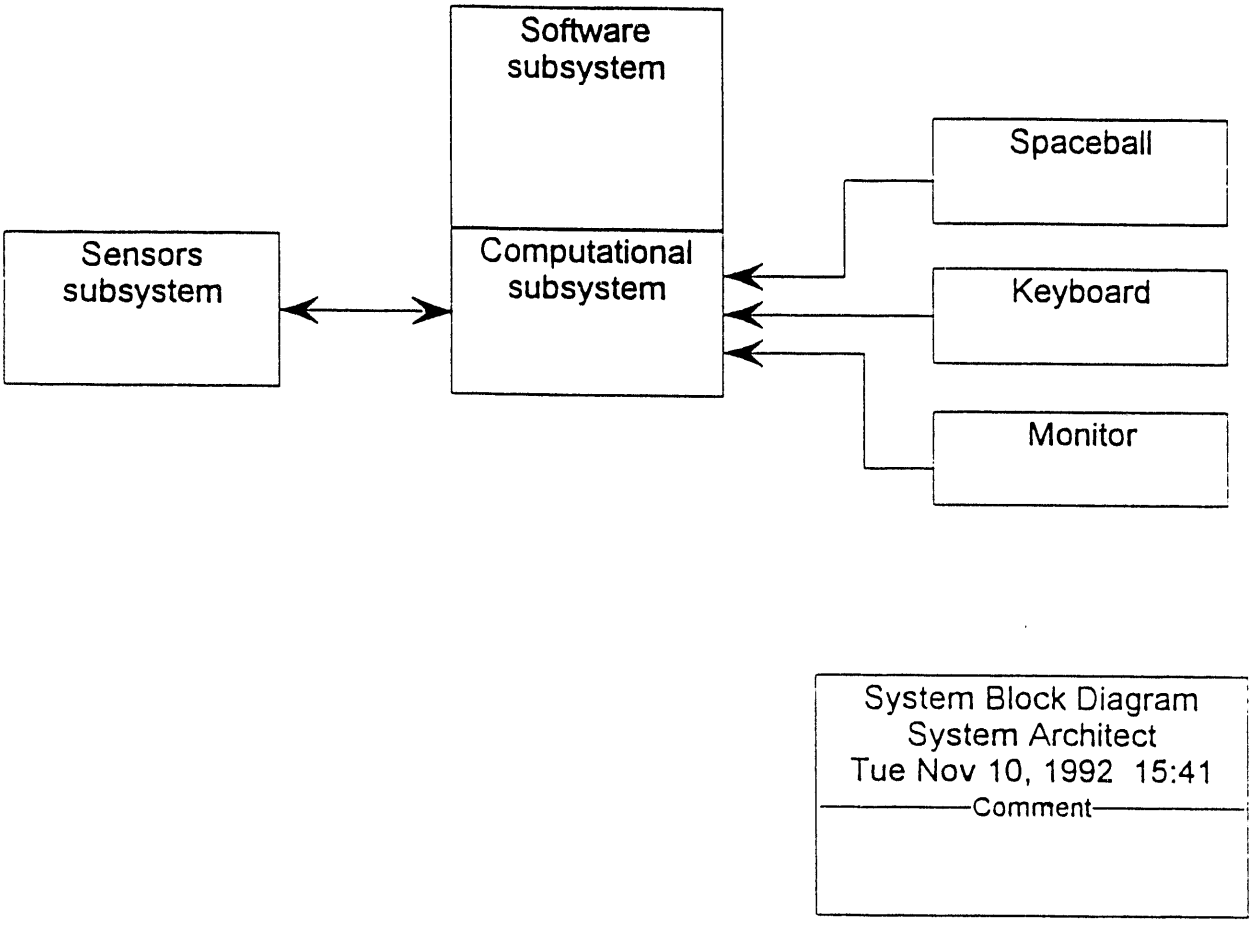


Figure 3-2 — System block diagram

3.2 Sensor subsystem

The sensor subsystem shall contain a color TV camera, a surface mapping sensor, and appropriate illumination means. The equipment will be designed to function for a specified period in the environment of the site. The sensing equipment will be physically located at the site and operated remotely. The TV camera and its mounting shall provide the operator with the ability to visually scan the site and to vary the resolution, as with a zoom lens. The operator shall have control of the lighting and the ability to adjust the focus and iris setting of the lens. The camera and display monitor shall have adequate resolution to show details of the features of interest.

The sensor subsystem is not part of the Phase 1 scope, and is included here for reference only.

3.3 Computational subsystem

The computational subsystem shall:

- o support native software development
- o provide high speed graphical display operations
- o provide keyboard operator input
- o provide a graphical/spatial operator input device
- o provide disk storage for large data models
- o provide network interconnection

The computational subsystem is provided by a Silicon Graphics Indigo workstation.

3.4 Software subsystem

The software subsystem contains all software to support the system. It includes software written for rapid prototyping as well as non-developmental software. This software breakdown comprises the data library and tool kit subsystems described in MTI technical proposal Q2-030.

The software subsystem is a single computer software configuration item (CSCI). The software is divided into the following six CSCs and the requirements for Phase I are assigned.

Table 3-2 Allocation Of ICERVS Requirements To CSCs For Phase I

CSC Name	Full Implementation	Partial Implementation	Design Influence
User Interface	R2.05 R3.01 R3.04 R3.06 R3.11 R6.02	R3.03 R3.05 R6.01 R8.01 R7.01	R2.07 R3.02 R3.07 R3.08 R3.10 R5.03 R6.04 R6.05
Octree Engine	R1.01 R1.04 R3.06 R6.02	R7.01	R1.02 R1.03 R5.02
Objects	R1.05 R1.07 R1.08 R2.05 R2.08 R6.02	R2.06 R3.05 R7.06	R1.06 R2.01 R2.03 R5.03
Octree-Objects Interaction			
Operational Sequencing			
Sensor I/O			

4.0 SOFTWARE SUBSYSTEM DETAILED DESIGN

The ICERVS Software Subsystem is a single Computer Software Configuration Item (CSCI). It is divided into several Computer Software Components (CSC), which in some cases are further divided into Computer Software Units (CSU).

The list below illustrates the organization of the ICERVS Software Subsystem. (Phase I items are in bold type):

ICERVS Software Subsystem

User Interface (UI-CSC)

Octree Engine (OE-CSC)

Object Modeling (OM-CSC)

Octree-Objects Interface (OE/OM-CSC)

Operational Sequencing (OS-CSC)

Sensor I/O (SIO-CSC)

The following sections describe the design of the three Computer Software Components for the ICERVS Phase I system.

4.1 User Interface Computer Software Component (UI-CSC)

The UI-CSC encompasses all functions that display data on the operator screen and take input from the operator. The user interface is graphical with pull-down menus, windows, and dialog boxes.

The UI-CSC relies heavily upon the Rogue Wave **Tools.h++** and **Views.h++** class libraries. The **Tools.h++** library provides a complete toolbox including Smalltalk-like collections, generic collections, string and character manipulation classes, date and time handling, file I/O, virtual I/O streams, virtual arrays, and much more. The **Views.h++** library provides a complete and easy to use encapsulation of the Open Software Foundation's Motif Graphical User Interface. In many cases windowed user interfaces with complex menuing systems can be constructed quickly with little knowledge of Motif.

4.1.1 Preliminary Design Requirements and Functions

The requirements for the UI-CSC are summarized in Table 4-1. For a more detailed description of each requirement, refer to the ICERVS System Design Report. Requirements that apply to Phase I are in bold type. Requirements that not part of Phase I requirements but strongly influence the software design are italicized. Some requirements that span multiple phases have been reworded to clarify the Phase I requirement. These requirements are marked with an asterisk. Requirements that span multiple CSCs are also identified.

Table 4-1 -- ICERVS Requirements for the User Interface CSC

System Requirement Number	Description
R2.05 *	**Define, modify, display and erase 2D convex polygons**
R2.07	*Dimensioning Tools - tick marks, measuring cursors*
R3.01	**Translate and scale**
R3.02	*Display coordinate axes/grid*
R3.03 *	**Pair of parallel cutplanes**
R3.04	**Geometric Objects: display/edit associated text data**
R3.05	**Geometric Objects: wire frame polygons** (shared with OM-CSC)

Table 4-1 -- ICERVS Requirements for the User Interface CSC (continued)

R3.06	**Update octree display as input points received** (shared with OE-CSC)
R3.07	*Pseudo-color octree display based on property or dimension*
R3.08	*Property derived coloring for objects (shared with OM-CSC)*
R3.09	Text display of current view parameters
R3.10	*Save and recall view parameter set*
R3.11	**Multiple view windows displaying the same data**
R3.12	View tracks sensor station viewpoint
R3.13	Display 2.5D surface map data, pseudo color by Z
R5.02	Set Region with octree to selected state (shared with OE-CSC)
R5.03	*Operator delete objects. (shared with OM-CSC)*
R6.01	**Operator edit Phase I system parameters**
R6.02	**Save/retrieve models to/from disk** (Distributed among CSCs)
R6.03	Rebuild octree from backup raw data points (shared with OE-CSC)
R6.04	*Maintain operator log and notebook for observations and other notes*
R6.05	*Support multiple system of units (shared with OE-CSC)*
R6.06	Define disassembly data. (Shared with OM-CSC)
R8.01	**Graphics tools: mouse, pull down menus, dialog boxes**
R9.02	Display sensor line-of-sight
R9.03	Text display of station angles
R9.04	Automatically map surfaces
R9.05	Automatic surface mapping parameters
R9.06	Draw / display scan paths

4.1.2 Derived Requirements and Functions

Derived requirements relate to the specialization of the basic requirements such that basic functions and primitives are readily identifiable. For the UI-CSC, the approach taken is to identify the primary requirement and then to list the individual derived requirements. Several basic requirements that are not part of Phase I have a strong impact on the design of the UI-CSC; these requirements were italicized in Table 4-1 and are also italicized below. For completeness and understandability, these requirements are treated as though they were part of the Phase I requirements.

4.1.2.1 R2.05 Create, modify, and display 2D convex polygons

a. Screen graphics: point, line, rectangle, circle, polygon
b. Draw screen graphic
c. Erase a screen graphic
d. Select a screen graphic
e. Move a screen graphic
f. Resize line, rectangle and circle graphics
g. Reshape polygon graphic (move vertex, drop vertex, add vertex)
h. 2D polygons: point, line, rectangle, circle, general polygon
i. Interactively create 2D polygon
j. Create 2D polygon from list of vertices
k. Draw 2D polygon
l. Erase 2D polygon
m. Select 2D polygon
n. Move 2D polygon
o. Resize 2D polygon
p. Reshape 2D general polygon
q. Associate screen graphic with a 2D polygon
r. Convert 2D polygon into screen graphic
s. Convert screen graphic into 2D polygon
t. Notify OM-CSC when polygon has been moved or changed

4.1.2.2 *R2.07 Dimensioning Tools - tick marks, measuring cursors*

a. *Readout for view scale slider*
b. *Readouts for view translation scroll bars*
c. *Readouts for view cutplanes*

4.1.2.3 R3.01 Translate and Scale

a. Use slider bars for translation controls
b. Use slider bar for scaling control
c. Convert translation slider position into view window offset factors
d. Convert scale slider position into view window scaling factors
e. Notify OE-CSC when translation or scaling has changed
f. Redisplay the view after slider stops.

4.1.2.4 *R3.02 Display coordinate axes/grid.*

a. *Enable / disable coordinate axes*
b. *Compute grid scaling from tree display level*

4.1.2.5 **R3.03 Pair of parallel cutplanes**

a. Enable / disable cutplanes
b. Use icon (arrow/line) for each cutplane
c. Convert icon position into cutplane description parameters
d. Notify OE-CSC when cutplane position has changed
e. Redisplay all views after cutplane icon stops

4.1.2.6 **R3.04 Geometric Objects - display/edit associated text data**

a. Select a 2D geometric object
b. Determine associated 3D geometric object
c. Call to OM-CSC to get associated text
d. Use general purpose text editing window as editor
e. Call to OM-CSC to update text

4.1.2.7 **R3.05 Geometric Objects - wire frame polygons**

a. Interface with OM-CSC
b. Assemble 3D geometric object from set of 2D polygons
c. Convert 3D geometric object into set of 2D polygons
d. Associate 2D polygon with 3D geometric object

4.1.2.8 **R3.06 Update octree display as input points received**

a. Use dialog box to get new point data from user
b. Use file selection box to get file of points to add
c. Interface with OE-CSC to add points/list of points

4.1.2.9 *R3.07 Pseudo-color octree display based on property or dimension*

a. View window menu option for type of display
b. Display routine supports color mapping

4.1.2.10 *R3.08 Property derived coloring for objects (shared with OM-CSC)*

a. User setable geometric object and object category colors
b. Display routine supports color mapping

4.1.2.11 *R3.10 Save and recall view parameter set*

a. *Read / write saved view files*
b. *Read / write saved view index files*
c. *Save a view*
d. *Restore a saved view*

4.1.2.12 **R3.11 Multiple windows displaying the same data**

a. Interface with OE-CSC
b. Create, clear and delete view windows
c. Select view window
d. Position / resize view windows
e. Support x,y,z orthogonal views
f. Color code the windows for each type of view
g. Display (DrawPoint) routine for octree data
h. Associate set of view windows with octree
i. Update shared octree attributes when view window attributes are updated
j. Compute size of smallest displayable area
k. View window graphic drawing primitives: DrawPoint, DrawLine

4.1.2.13 *R5.03 Operator delete objects*

a. *Select 2D object to delete*
b. *Find associated 3D object index*
c. *Interface with OM-CSC to delete 3D geometric object*

4.1.2.14 **R6.01 Operator edit phase I system parameters**

a. Use general purpose text editing window as editor
b. Verify consistency/completeness of parameters
c. Print parameters and parameter file

4.1.2.15 **R6.02 Save/retrieve models to/from disk**

a. Read / write model parameter file
b. Select a model to make active
c. Create new model
d. Delete existing model

4.1.2.16 *R6.04 Maintain operator log and notebook for observations and other notes*

a. *Read / write history log file*
b. *Edit history log file*
c. *Append log entry to log file*
d. *Print log file.*

4.1.2.17 *R6.05 Support Multiple System Of Units*

a. *Internal -- The fixed system of units used internally by ICERVS will be SI units.*
b. *External -- A user specified system of units for data input and display*
c. *Convert to/from internal units when adding data / displaying data*
d. *Convert to/from internal units when adding objects / displaying objects*

4.1.2.18 R8.01 Graphics tools - space ball/mouse, pull-down menus, dialog boxes

a. Main Window Menu Bar Functions: Model, View, Window, Help

b. Main Window MODEL Functions: Select, Create, Delete, Edit Notebook, Print Notebook, Edit Parameters, Print Parameters

c. Main Window VIEW Functions: Select, Create, Close, Restore, Save, Save All

d. Main Window WINDOW Functions: Select

e. Main Window HELP Functions: Index, Extended, About ICERVS

f. View Window Menu Bar Functions: Data, Add, Display, Objects, Debug

g. View Window DATA Menu Functions: New, Open, Save

h. View Window ADD Menu Functions: Point, List

i. View Window OBJECTS Menu Functions: Show, Create, Delete, Edit, Print, Print All

j. View Window DEBUG Menu Functions: Show Tree, Statistics, Refresh

4.1.3 Class Descriptions

The UI-CSC software is objected oriented, implemented in C++ and consists of approximately thirty (30) classes. The C++ class provides a mechanism for combining the data and the manipulation procedures related to a high-level entity into a single construct. Classes facilitate abstraction (ignoring details of processes and how data is represented), promote encapsulation (hiding of the internal workings of entities) and support inheritance (defining new entities as specializations of other entities). The first step in object-oriented design is to identify the classes. Later steps involve assignment of attributes and behavior, identification of relationships between classes and arrangement of the classes into hierarchies. This section identifies the UI-CSC classes and discusses their general characteristics (attributes, behavior and relationships). The last subsection (4.1.3.7) will describe the relationship of the classes to the basic and derived requirements defined in previous sections (4.1.1 and 4.1.2). Section 4.1.4 will discuss the major functions assigned to the UI-CSC and describe how the software classes implement the functions.

To simplify the discussion of the UI-CSC software, the classes have been organized into six (6) groups, as follows:

1. Utility Classes
2. Main Window Related Classes
3. System Parameters and Work Volume Related Classes
4. View Window Related Classes
5. Object Modeling Interface Classes
6. Octree Engine Interface Classes

The following sections describe the purpose and function of each group and each class within the group. Figures 4-1 through 4-3 provide illustrations of the interactions among the UI-CSC classes.

4.1.3.1 Utility Classes

Utility classes provide a set of common services that other software components may use when needed. These classes are included as part of the UI-CSC only as a convenience. In general, the requirement for the utility class was first identified during the UI-CSC design. The window classes (CBrowseWindow, CEditorWindow, etc.) are ICEKVS application shells (another level of abstraction) that surround several elements of the Rogue Wave **Views.h++** library.

CBrowseWindow: A general purpose read-only window class for viewing and optionally printing an ASCII text file. This class is a higher level abstraction of a Rogue Wave RWScrolledTextWindow.

CConfigManager: A class that manages configuration parameter files. These files contain groups of key/value pairs that may be read/written in random order. The parameter file itself is ASCII and may be maintained with any general purpose text editor such as a CEditorWindow.

CEditorWindow: A general purpose read/write window class for viewing and editing an ASCII text file. No attempt is made to assess the validity or correctness of the edited information. This class is a higher level abstraction of a Rogue Wave RWScrolledTextWindow.

CFileListWindow: A general purpose dialog box class for presenting the operator with a selection list of filenames. The operator will be allowed to select one filename from the list. This class is derived from a Rogue Wave RWFileListDialog.

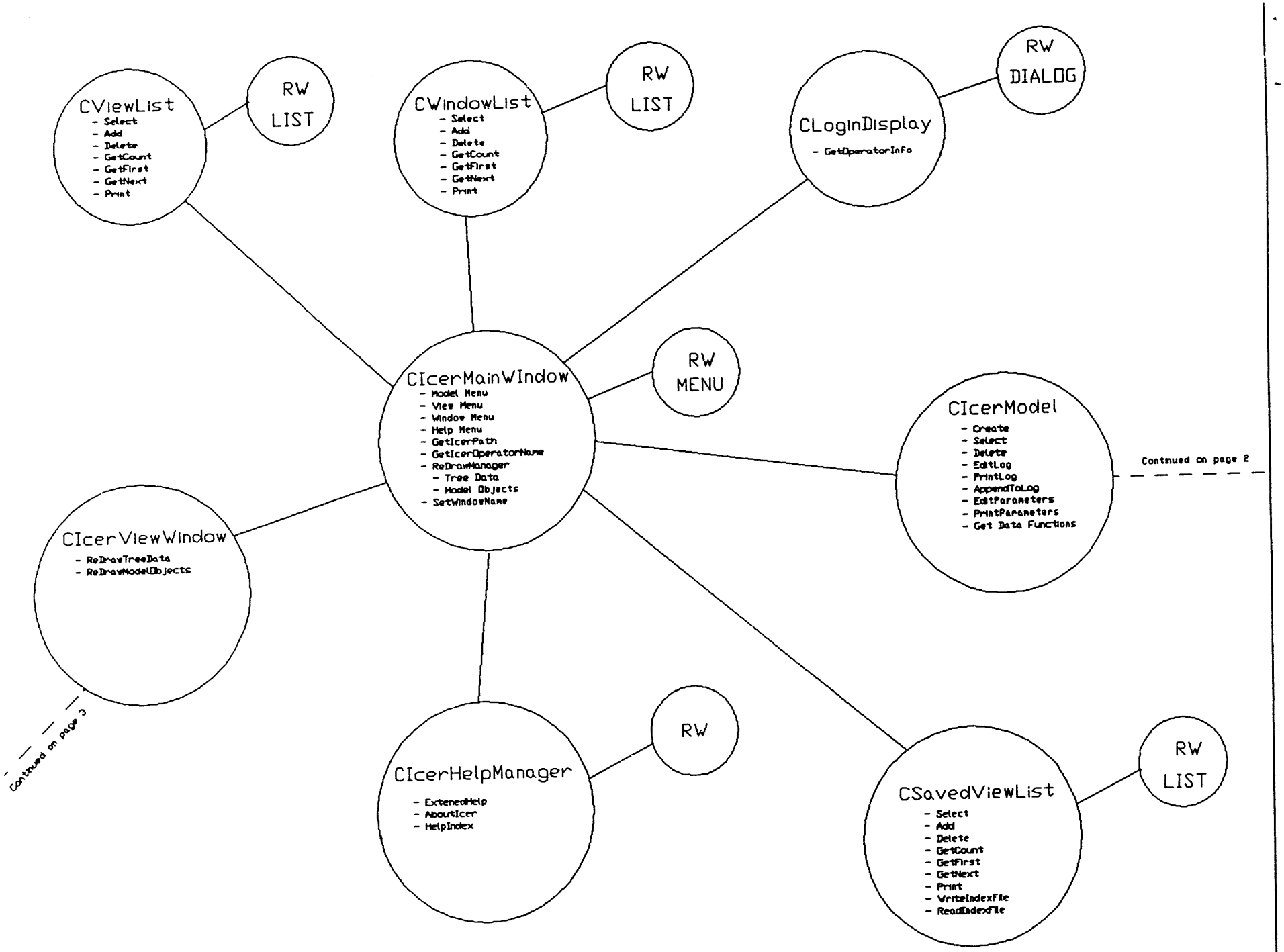


Figure 4-1 -- ICERVS Main Window Class Interactions

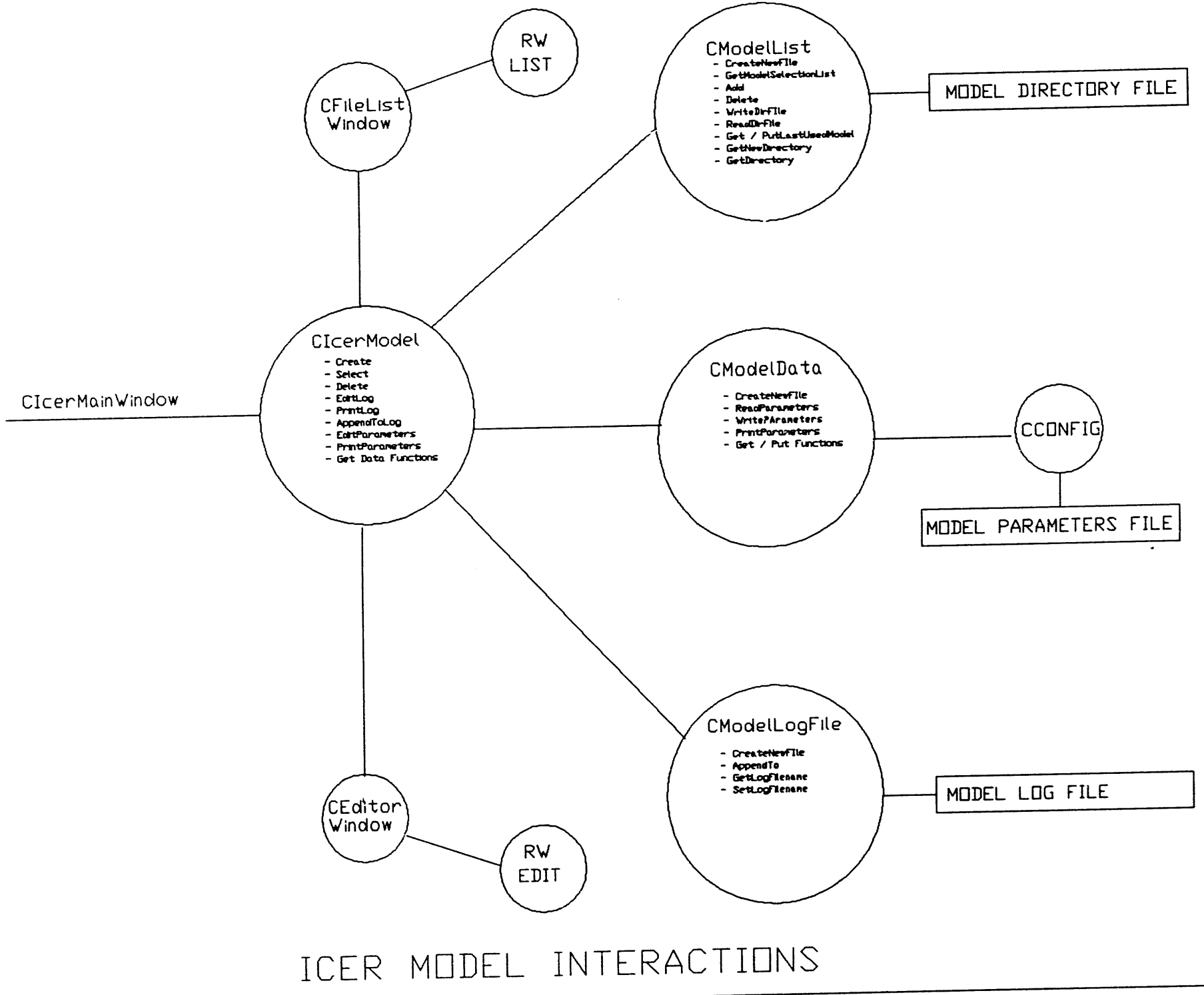


Figure 4-2 -- ICERVS World Model (work space) Class Interactions

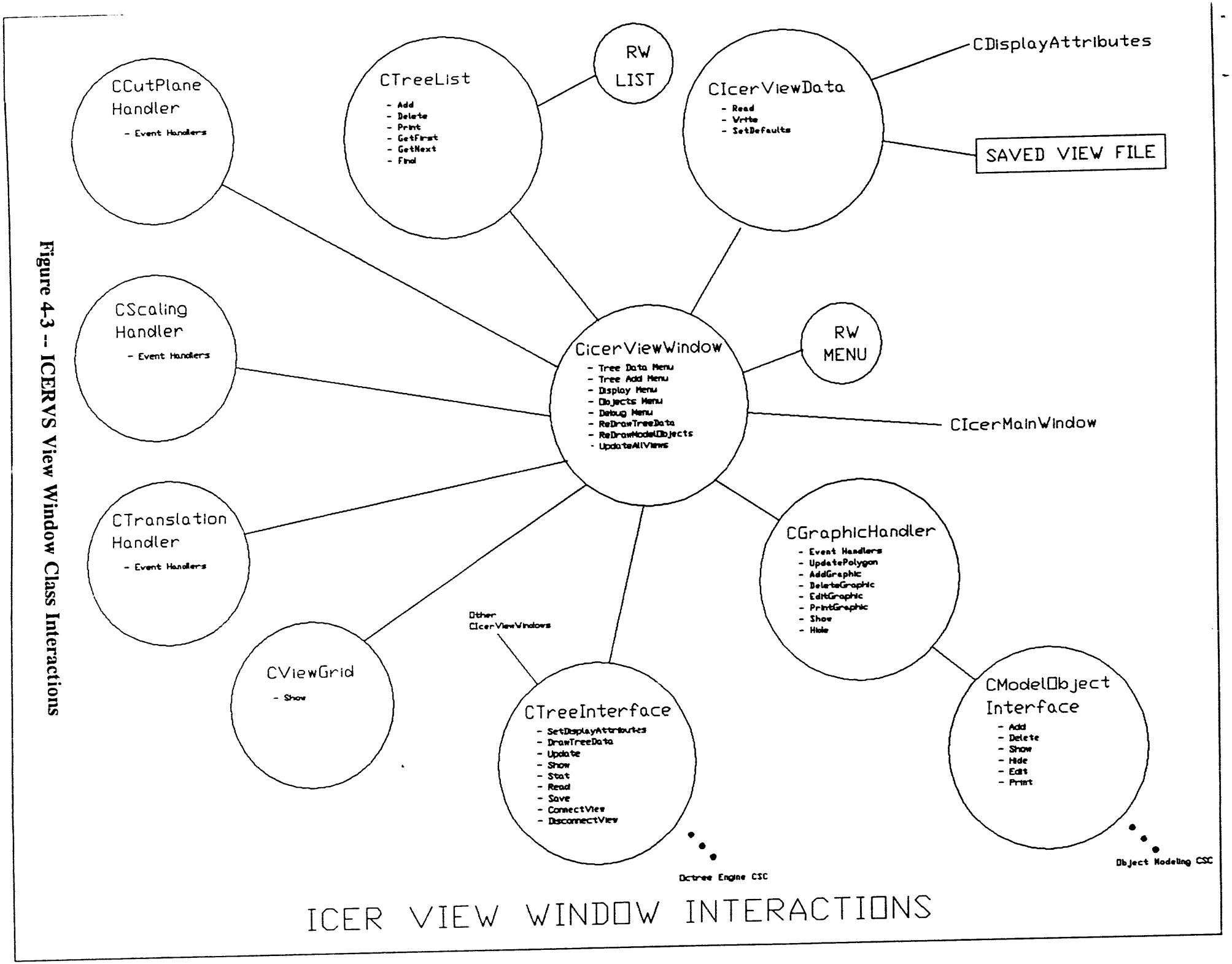


Figure 4-3 -- ICERVS View Window Class Interactions

CIcerWindow: The base class for all ICERVS window classes (except CIcerMainWindow, CBrowseWindow and CEditorWindow). This class provides all the common functionality for ICERVS windows and is derived from the Rogue Wave RWAuxView class.

COrderedStringList: An ordered list of string items. Entries are sequenced by insertion order. Once added to the list, an item may be accessed by index number. Deleting an item does not alter the index position of other items. Items in the list must be a Rogue Wave RWCollectableString class.

C3dPoint: An ordered triplet of values (x,y,z) that represents the coordinates of a point in space. Methods are provided for defining and examining individual coordinate values.

CVertex: A vertex of a three-dimensional polyhedra. The CVertex class is derived from the C3dPoint class.

4.1.3.2 Main Window Related Classes

Main window related classes are all concerned with the topmost interface with the operator. This level of the user interface manages the main menus, establishes the work volume context, accepts and dispatches top-level commands, launches the view windows, maintains lists of other active windows, and sequences the orderly shutdown of the software system.

CIcerHelpManager: The class that implements the ICERVS help system. This class is not implemented in Phase I.

CIcerMainWindow: A class that implements the ICERVS main window and its menus. The CIcerMainWindow class is derived from the Rogue Wave RWMainView class. Exactly one instance of CIcerMainWindow exists while the ICERVS software subsystem is active.

CLoginDisplay: A class that encapsulates all aspects of logging into the ICERVS system. An instance of this class is created, used and destroyed during the start-up of the ICERVS software subsystem.

CSavedViewList: A class that examines the disk, creates a list of saved view files, and allows the operator to select one of the files for restoration. Instances of this class are usually created, used and destroyed whenever needed. This class is derived from the utility class CFileListWindow.

CViewList: A class that creates and maintains a list of all active view windows. There is only one instance of CViewList and it is owned by the ICERVS main window.

CWindowList: A class that creates and maintains a list of all active windows (other than the main window). There is only one instance of CWindowList and it is owned by the ICERVS main window.

4.1.3.3 System Parameters and Work Volume Related Classes

The system parameters and work volume related classes provide/control access to the world model parameter and log files. The CIcerModel class contains the other classes and the CIcerModel object will in turn be owned by the CIcerMainWindow object. All access to the model's data will be controlled by the CIcerModel object.

CIcerModel: A class that encapsulates all the world model (work space) specific data parameters. Only one instance of this class exists at any one time and it is owned by the ICERVS main window. The CIcerModel class is derived from CModelData class. When a new world model is selected, the old CIcerModel object is destroyed and a new instance created.

CModelData: A class that encapsulates all aspects of the world model parameter file and the data items contained in the file. Read, write and edit capabilities are provided.

CModelList: A class that maintains a list of ICERVS world models stored on disk. This class is used exclusively by the CIcerModel object.

CModelLog: A class that encapsulates all aspects of the world model log file.

4.1.3.4 View Window Related Classes

The view window related classes handle all aspects of the user's view of an octree and the geometric objects. Several view windows may exist at one time and display different portions of the same octree and set of geometric objects.

CIcerViewWindow: The ICERVS octree view window class. A unique instance of this class is created for each view window requested by the operator. Each view window manages all aspects of its octree view and has menu functions to allow the operator to interact with the view and the octree. Several view windows may be built on the same octree data set. The view window also provides an interface to the object modeling CSC. Polygons and other 2D figures may be drawn in a set of view windows and combined to represent a 3D geometric object. A 3D geometric object may be selected, translated, scaled, modified or deleted by manipulating the associated 2D polygons in the view windows.

CIcerViewData: A class that encapsulates the data (size, position, colors, etc.) that describes a view window. This class is also capable of creating saved view files that can later be restored.

CCutPlaneHandler: A class that encapsulates all aspects of the view window's cutplanes. Cutplanes are represented by two vertical lines in the view window. The cutplanes may be disabled, enabled and moved independently in each view window. The position of all cutplanes in all of the view windows affects the data displayed in each of the view windows.

CGraphicHandler: A class that encapsulates all aspects of the creation, manipulation, and destruction of 2D geometric object graphics. This class relies upon the Rogue Wave Mouse Wrapper portion of **Views.h++**. Most of the ability to position, resize, reshape, and select 2D objects is directly performed by the Rogue Wave software. The CGraphicHandler receives notification after the Rogue Wave operations are completed and will then update the necessary ICERVS geometric objects.

CScalingHandler: A class that encapsulates all aspects of scaling the view window. Most of the ability to rescale is directly performed by the Rogue Wave software. CScalingHandler receives notification after the Rogue Wave operations are complete and the user has stopped moving the scaling slider bar. The view window will then be commanded to redraw itself.

CTranslateHandler: A class that encapsulates all aspects of translating the view window. Most of the ability to translate is directly performed by the Rogue Wave software. CTranslateHandler receives notification after the Rogue Wave operations are complete and the user has stopped moving the translation slider bars. The view window will then be commanded to redraw itself.

CTreeList: A class for maintaining a list of CTreeInterface instances. This list is part of the mechanism for allowing multiple view to display the same octree. There is only one instance of a CTreeList and it is owned by all active views. The CTreeList object is a static member of CIcerViewWindow class.

CViewGrid: A class for creating and displaying a grid on a view window. If an instance of CViewGrid exists, then the grid is visible. If no instance exists, the grid is not visible.

4.1.3.5 Object Modeling Interface Classes

These interface classes establish and control the communications between the UI-CSC and the OM-CSC. This is done to simplify the software design, to permit parallel design and implementation of the software, and to promote the modularity of the software.

CModelObjectInterface: A class that implements a controlled interface between the User Interface and the Object Modeling CSC. Its primary function is to isolate the UI-CSC and the OM-CSC by ensuring that implementation details and dependencies of one CSC will never affect the other CSC. Only one instance of CModelObjectInterface will exist. All view windows will share this one interface instance.

CObjectDisplayAttribute: A class that defines the display attributes that apply to all geometric objects such as visibility by category, color by category, etc.

CIcerGraphic: The abstract base class for all 2D displayable graphic object classes. This class provides all the common behavior for 2D graphic objects. All other 2D graphic object classes must be derived from CIcerGraphic. Dimensional information for CIcerGraphic objects is always expressed in ICERVS internal units. For Phase I, only CIcerRectangle and CIcerPolygon are implemented. Future derived types will include CIcerPoint, CIcerLine, CIcerCircle, and others.

CIcerRectangle: A class that encapsulates a drawable 2D rectangle.

CIcerPolygon: A class that encapsulates a drawable 2D polygon.

4.1.3.6 Octree Engine Interface Classes

These interface classes establish and control the communications between the UI-CSC and the OE-CSC. This is done to simplify the software design, to permit parallel design and implementation of the software, and to promote the modularity of the software.

CTreeInterface:: A class that implements a controlled interface between the User Interface and the Octree Engine CSC. Its primary function is to isolate the UI-CSC and the OE-CSC by ensuring that implementation details and dependencies of one CSC will never affect the other CSC. Multiple instances of CTreeInterface may exist. All view windows that share the same octree data will share a common interface instance.

CDisplayAttribute: A class that provides an encapsulation for defining how octree data is displayed. It contains CCutplane and CColorMap objects.

CCutplane: A class that defines and implements a half-space. The volume in the half-space is visible, the rest is not visible.

CColorMap: A class for converting an octree CNodeData to a color value.

4.1.3.7 Relationship Between UI-CSC Requirements and Classes

Previous sections have enumerated the ICERVS basic requirements for the UI-CSC, defined a set of derived requirements for each basic requirement and identified a set of software classes for the UI-CSC. This section will define the relationships between the requirements and the classes.

Table 4-2 identifies the classes that implement each ICERVS basic requirement. Requirements that apply to Phase I are in bold type. Requirements that not part of Phase I requirements but strongly influence the software design are italicized. Some requirements that span multiple phases have been reworded to clarify the Phase I requirement. These requirements are marked with a asterisk. The order in which the classes are listed for each requirement (except R8.01) is significant. The first class listed is either CIcerMainWindow or CIcerViewWindow and depends upon which ICERVS menu function the operator would select to exercise the requirement. The remaining classes are ordered roughly in the sequence that would be encountered during execution of the selected menu function. Note that the list of classes for each requirement generally ends with a class that interfaces to another CSC or with a utility class.

Table 4-3 identifies the ICERVS basic requirements related to each UI-CSC class.

Table 4-4 identifies the detailed relationships between the derived requirements and the UI-CSC classes. Most of the derived requirements relate to one data member or one function member of a single class. These relationships are denoted by the expression *class_name*::*data_member_name* or *class_name*::*function_name*(). In many cases, a derived requirement is implemented by a Rogue Wave class. This is indicated by listing the name(s) of the Rogue Wave class. All Rogue Wave class names start with "RW" (e.g. RWScale, RWScrollBar, etc.).

Table 4-2 -- UI-CSC Classes For Each ICERVS Requirement

Requirement Number	Requirement Description and List Of Related Classes
R2.05	**Define, modify, display and erase 2D convex polygons** CIcerViewWindow CGraphicHandler C3dPoint CVertex CIcerGraphic, CIcerRectangle, CIcerPolygon CModelObjectInterface
R2.07	*Dimensioning Tools - tick marks, measuring cursors* CIcerViewWindow CCutplaneHandler CTranslateHandler CScalingHandler
R3.01	**Translate and scale** CIcerViewWindow CTranslateHandler CScalingHandler CTreeInterface
R3.02	*Display coordinate axes/grid* CIcerViewWindow CViewGrid
R3.03	**Pair of parallel cutplanes** CIcerViewWindow CCutplaneHandler CTreeInterface
R3.04 *	**Geometric Objects: display/edit associated text data** CIcerViewWindow CModelObjectInterface CBrowseWindow CEditorWindow COrderedStringList

Table 4-2 -- UI-CSC Classes For Each ICERVS Requirement (continued)

R3.05 *	**Geometric Objects: wire frame polygons** CIcerViewWindow CGraphicHandler CIcerGraphic, CIcerRectangle, CIcerPolygon C3dPoint CVertex CModelObjectInterface
R3.06 *	**Update octree display as input points received** CIcerViewWindow CTreeInterface
R3.07	*Pseudo-color octree display based on property or dimension* CIcerViewWindow CColorMap
R3.08	*Property derived coloring for objects (shared with OM-CSC)* CIcerViewWindow CObjectDisplayAttribute
R3.10	*Save and recall view parameter set* CIcerMainWindow CSavedViewList CIcerViewWindow CIcerViewData
R3.11	**Multiple view windows displaying the same data** CIcerMainWindow CViewList CIcerViewWindow CTreeList CTreeInterface
R5.03	*Operator delete objects* CIcerViewWindow CGraphicHandler CModelObjectInterface

Table 4-2 -- UI-CSC Classes For Each ICERVS Requirement (continued)

R6.01 *	**Operator edit Phase I system parameters** CIcerMainWindow CModelData CEditorWindow
R6.02	**Save/retrieve models to/from disk** CIcerMainWindow CIcerModel CModelList CModelData CConfigManager
R6.04	*Maintain operator log and notebook for observations and other notes* CIcerMainWindow CModelLogFile CEditorWindow CBrowseWindow COrderStringList
R6.05	*Support multiple system of units* CIcerMainWindow CModelData CIcerViewWindow CIcerViewData CTreeInterface
R8.01 *	**Graphics tools: mouse, pull down menus, dialog boxes** CIcerMainWindow, CIcerWindow, CIcerViewWindow, CBrowseWindow, CEditorWindow, CFileListWindow, CViewList, CWindowList, CSavedViewList, CModelList, CTreeList, CLoginDisplay, CIcerHelpInterface

Table 4-3 -- ICERVS Requirements For Each UI-CSC Class

Class	Related Requirements
Utility Classes	
CBrowseWindow	R3.04, R6.04
CConfigManager	R6.02
CEditorWindow	R3.04, R6.01, R6.04
CFileListWindow	R8.01
CIcerWindow	R8.01
COrderedStringList	R3.04, R6.04
C3dPoint	R2.05, R3.05
CVertex	R2.05, R3.05
Main Window Classes	
CIcerHelpManager	R8.01
CIcerMainWindow	R3.10, R3.11, R6.01, R6.02, R6.04, R6.05, R8.01
CLoginDisplay	R8.01
CSavedViewList	R3.10, R8.01
CViewList	R3.11, R8.01
CWindowList	R8.01
Work Volume Classes	
CIcerModel	R6.02
CModelData	R6.01, R6.02, R6.05
CModelList	R6.02
CModelLogFile	R6.04
View Window Classes	
CIcerViewWindow	R2.05, R2.07, R3.01, R3.02, R3.03, R3.04, R3.05, R3.06, R3.07, R3.08, R3.10, R3.11, R5.03, R6.05,R8.01
CIcerViewData	R3.10, R6.05
CCutplaneHandler	R2.07, R3.03
CGraphicHandler	R2.05, R2.07, R3.05, R5.03
CScalingHandler	R2.07, R3.01
CTranslateHandler	R2.07, R3.01
CTreeList	R3.11
CViewGrid	R3.02

Table 4-3 -- ICERVS Requirements For Each UI-CSC Class (continued)

Class	Related Requirements
Modeling Interface Classes	
CModelObjectInterface	R2.05, R3.04, R3.05, R5.03
CObjectDisplayAttribute	R2.05, R3.05, R3.08
CIcerGraphic	R2.05, R3.05
CIcerRectangle	R2.05, R3.05
CIcerPolygon	R2.05, R3.05
Octree Interface Classes	
CTreeInterface	R3.01, R3.03, R3.06, R3.11, R6.05
CDisplayAttribute	R3.06
CCutplane	R3.03
CColorMap	R3.06, R3.07

Table 4-4 -- UI-CSC Detailed Requirements / Class Relationships

Requirement Number	Requirement Description and Detailed Class Relationships
R2.05	**Define, modify, display and erase 2D convex polygons**
a.	RWGraphic, RWXCanvas
b.	RWXCanvas
c.	RWXCanvas
d.	RWXCanvas
e.	RWXCanvas
f.	RWXCanvas
g.	RWXCanvas
h.	CIcerGraphic. CIcerRectangle, CIcerPolygon constructors
i.	CGraphicHandler::AddGraphic()
j.	RWXCanvas
k.	CGraphic::ShowGraphic()
l.	CGraphic::EraseGraphic()
m.	CGraphic::SelectGraphic()
n.	RWXCanvas, CGraphicHandler::MovedCallBack()
o.	RWXCanvas, CGraphicHandler::MovedCallBack()
p.	RWXCanvas, CGraphicHandler::MovedCallBack()
q.	CGraphicHandler::graphicList
r.	CGraphicHandler::AddGraphic()
s.	CGraphic::MovedCallBack()
t.	CModelObjectInterface::Update()
R2.07	*Dimensioning Tools - tick marks, measuring cursors*
a.	CScalingHandler::MovedCallBack()
b.	CTranslationHandler::MovedCallBack()
c.	CCutPlaneHandler::MovedCallBack()
R3.01	**Translate and scale**
a.	CTranslateHandler constructor, RWScale
b.	CScalingHandler constructor, RWScale
c.	CTranslateHandler::MovedCallBack()
d.	CScalingHandler::MovedCallBack()
e.	CTranslateHandler::MovedCallBack(), CScalingHandler::MovedCallback()
f.	CTranslateHandler::MovedCallBack(), CScalingHandler::MovedCallback()

Table 4-4 -- UI-CSC Detailed Required/Class Relationships (continued)

R3.02 a. b.	*Display coordinate axes / grid* CViewGrid constructor CViewGrid::GridSize()
R3.03 a. b. c. d. e.	**Pair of parallel cutplanes** CIcerViewWindow::DisplayCutPlaneON/OFF() CCutPlaneHandler constructor CCutPlaneHandler::MovedCallBack() CCutPlaneHandler::MovedCallBack() CCutPlaneHandler::MovedCallBack()
R3.04 * a. b. c. d. e.	**Geometric Objects: display/edit associated text data** RWXCanvas CGraphicHandler::FindGraphic() CModelObjectInterface::Edit() CEditorWindow constructor CModelObjectList::Replace()
R3.05 * a. b. c. d.	**Geometric Objects: wire frame polygons** CModelObjectInterface::ConnectView() CModelObjectInterface::AddObject() CModelEntity::ConvertObjectTo2D() CGraphicHandler::graphicList
R3.06 * a. b. c.	**Update octree display as input points received** CIcerViewWindow::Add(), RWDialog CIcerViewWindow::AddList(), RWDialog CTreeInterface::AddPoint(), AddList()
R3.10 a. b. c. d.	*Save and recall view parameter set* CICerViewData::ReadSavedViewFile(), WriteSavedVIewFile() CSavedViewList::ReadSavedViewIndex(), WriteSavedViewIndex() CIcerMainWindow::ViewSave(), ViewSaveAll() CIcerMainWindow::ViewRestore()

Table 4-4 -- UI-CSC Detailed Required/Class Relationships (continued)

R3.07 a. b.	*Pseudo-color octree display based on property or dimension* CIcerViewWindow::DisplayColor(); CIcerViewWindow::DrawPoint()
R3.08 a. b.	*Property derived coloring for objects (shared with OM-CSC)* CIcerViewWindow::DisplayColor(); CIcerViewWindow::DrawLine()
R3.11 a. b. c. d. e. f. g. h. i. j. k.	**Multiple view windows displaying the same data** CTreeInterface constructor CIcerViewWindow constructor, destructor CIcerWindow::ToTop(), RWView RWView CIcerViewWindow::DisplayOrthogonalView() CIcerViewWindow::DisplayOrthogonalView() CTreeInterface::DrawPoint() CTreeList::Add(), Find() CTreeInterface::UpdateAttributes() CViewData::SmallestDisplayableRectangle() CIcerViewWindow::DrawPoint(), DrawLine()
R5.03 a. b. c.	*Operator delete objects* RWXCanvas CGraphicHandler::FindGraphic() CModelObjectInterface::Delete()
R6.01 * a. b. c.	**Operator edit Phase I system parameters** CEditorWindow constructor CModelData::Read() CIcerModel::PrintParameters(), CModelData::Print()

Table 4-4 -- UI-CSC Detailed Required/Class Relationships (continued)

R6.02	**Save/retrieve models to/from disk**
a.	CConfigManager constructor, RWCString
b.	CIcerMainWindow::ModelSelect(), CIcerModel::Select()
c.	CIcerMainWindow::ModelCreate(), CIcerModel::Create()
d.	CIcerMainWindow::ModelDelete(), CIcerModel::Delete()
R6.04	*Maintain operator log and notebook for observations and other notes*
a.	RWCString
b.	CIcerModel::EditModelLog(), CModelLogFile::EditLog()
c.	CModelLogFile::AppendLog()
d.	CIcerModel::PrintModelLog(), CModelLogFile::PrintLog()
R6.05	*Support multiple system of units*
a.	CModelData::Get/SetTankInternalUnits(),Get/SetTankInternalUnits()
b.	CModelData::Get/SetTankExternalUnits, Get/SetUnitsMultiplier(), Get/SetUnitsOffset
c.	CTreeInterface::AddPoint(), AddList(), DisplayPoint()
d.	CModelObjectInterface::Add(), Display()
R8.01 *	**Graphics tools: mouse, pull down menus, dialog boxes**
a.	CIcerMainWindow::modelMenu, viewMenu, windowMenu, helpMenu
b.	CIcerMainWindow::ModelSelect(), ModelCreate(), ModelDelete(), ModelEditNotebook(), ModelPrintNotebook(), ModelEditParameters(), ModelPrintParameters()
c.	CIcerMainWindow::ViewSelect(), ViewCreate(), ViewClose, ViewRestore(), ViewSave(), ViewSaveAll()
d.	CIcerMainWindow::WindowSelect()
e.	CIcerMainWindow::HelpIndex(), HelpExtended(), HelpAboutIcervs()
f.	CIcerViewWindow::dataMenu, addMenu, displayMenu, objectsMenu, debugMenu
g.	CIcerViewWindow::DataNew(), DataOpen(), DataSave()
h.	CIcerViewWindow::AddPoint(), AddList()
i.	CIcerViewWindow::ObjectsShow(), ObjectsCreate(), ObjectsDelete(), ObjectsEdit(), ObjectsPrint(), ObjectsPrintAll()
j.	CIcerViewWindow::DebufShowTree(), DebugStatistics(), DebugRefresh()

4.1.4 Major Function Descriptions

The UI-CSC software's major functions are implemented via pull-down menus which are available from both the main window and the view windows. These functions are organized into two groups, as follows:

Main Window Menu Functions: MODEL, VIEW, WINDOW, HELP

View Window Menu Functions: DATA, ADD, DISPLAY, OBJECT, DEBUG

In order to understand the functions of the UI-CSC and their implementation, it is necessary understand the structure and operation of the ICERVS main program, the ICerMainWindow class menus, and the ICerViewWindow class menus. The ICERVS main program code is very short and is listed below:

```
void main(int argc, char **argv) {
   CIcerMainWindow mainWindow(argc,argv,"ICERVS MAIN WINDOW");      1
   mainWindow.start();                                              2
   return; }
```

This code shows that the main program 1) creates an instance of a CIcerMainWindow as its main window and 2) starts the windowing system. All subsequent actions are performed as a consequence of the operator selecting some menu function.

The key to understanding how the menus operate (and how user commands are initiated) is the constructor function for the CIcerMainWindow and CIcerViewWindow classes. After Rogue Wave **Views.h++** has created the window, the most important action of these constructor functions is to build the window menu system. Using the **Views.h++** library makes building the menus very simple. All it takes is code similar to the following:

```
menu.addOption("Select",                    //Name for menu item
               'S',                         //Hot key for menu item
               this,                        //Owner of the menu
               &CIcerModel::Select,         //Callback function
               0,                           //Callback function data
               some value);                 //Callback function data

.                                           //Define other menu items
.
.
menu.attachTo(*menubar, "MODEL", 'M');      //Attach menu to menu bar
```

This code creates a menu item called "Select" and attaches it to the CIcerMainWindow menu bar at a position called MODEL. The callback function CIcerModel::Select is defined as the

function to be called when the user selects this menu item. Using similar code, the constructor for CIcerMainWindow completes its menu system and then returns control to the main program.

The main program turns over control to the Rogue Wave function *start()* to initiate the main window input loop. From this point on, the Rogue Wave **Views.h++** will perform all necessary actions to permit the operator to select a menu item. Control will be returned to the ICERVS software via the callback function associated with the menu item after the operator has selected a menu item.

The key point to bear in mind during the discussion of the UI-CSC functions is that there is not a single entry point in the software. Each menu item has its own separate entry point (via its callback function) into the ICERVS software. Consequently, the first step in establishing the relationships between the UI-CSC functions and the UI-CSC classes is to determine the associated callback function for the menu function being discussed. In general, this will be a unique member function of a particular class (generally, CIcerMainWindow or CIcerViewWindow).

4.1.4.1 Main Window Menu Functions

The main window is the top level window which appears as soon as the program is started and remains on the screen until the program is exited. See Figure 4-4 for an example of this screen. All other windows created are displayed on top of this window.

When the ICERVS program is started, the operator will be automatically prompted to logon the ICERVS system (another operation of the CIcerMainWindow constructor). See Figure 4-5 for an example logon screen. When the operator has successfully logged in, the last used world model from the previous ICERVS session will become the current working model. The operator may change the current working model by clicking on the MODEL-SELECT menu function and selecting another world model to be the current working model. The main window menu functions and their subfunctions are listed below:

MODEL		**VIEW**	**WINDOW**	**HELP**
SELECT		CREATE	DISPLAY	EXTENDED
CREATE		SELECT		INDEX
DELETE		CLOSE		ABOUT
NOTEBOOK.....	EDIT	RESTORE VIEW		
	PRINT	SAVE VIEW		
PARAMETERS...	EDIT	SAVE ALL		
	PRINT			
QUIT				

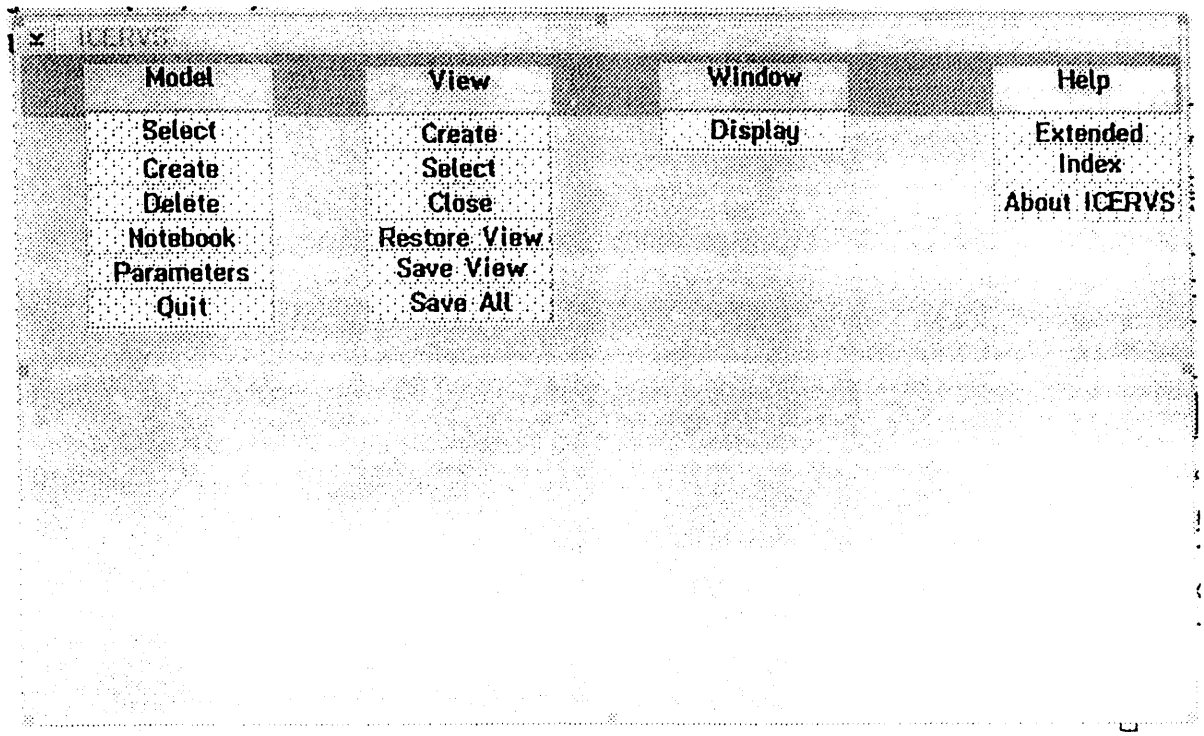


Figure 4-4 - ICERVS Main Window

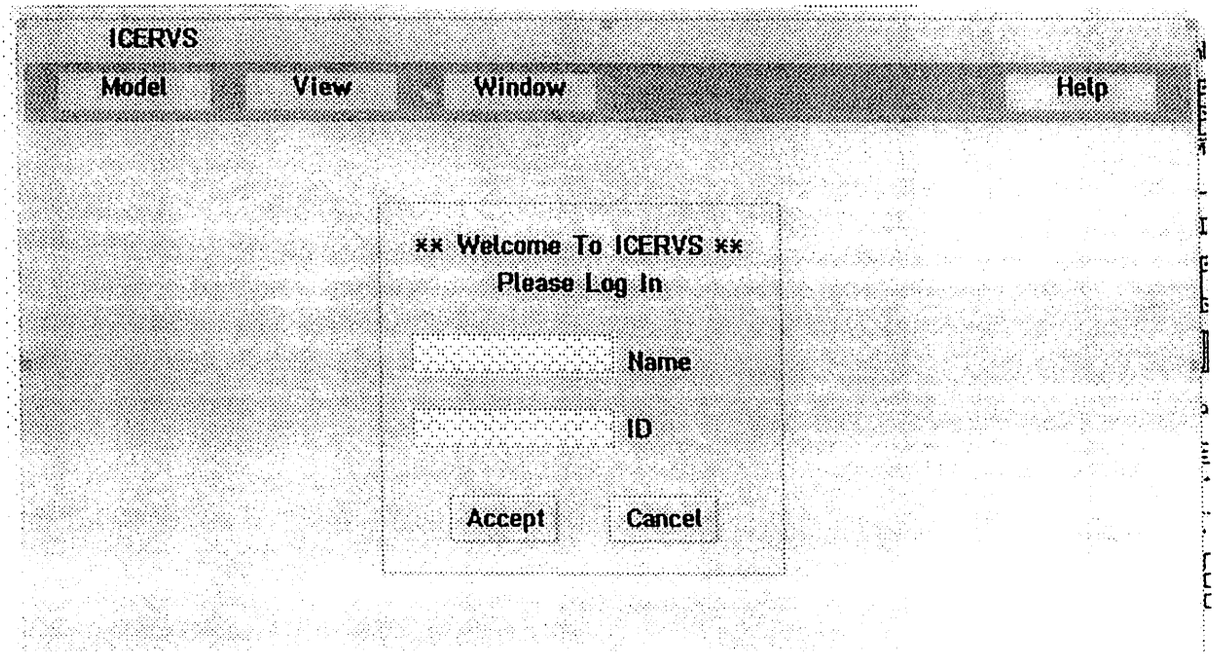


Figure 4-5 - ICERVS Logon Dialog

4.1.4.1.1 MODEL Sub-Menu Functions: MODEL sub-menu functions consist of all the functionality needed to maintain world models in the ICERVS system. A world model can be thought of as a single workspace (tank) configuration. There can be several world models defined in the ICERVS system; however, there is only one current model defined at any time. Each model is contained in a model subdirectory located under the main model directory: ...\MODELS. They are named MODEL1, MODEL2, etc. Each world model subdirectory contains the following files necessary to describe the world model:

Model Parameter File: This file contains all the parameters that pertain to the world model (i.e. tank description, system defaults, etc.). It is initially created by the MODEL-CREATE menu function from a default Model Parameter file and can be subsequently edited by the MODEL-PARAMETERS EDIT menu function. The file is an ASCII file. Each parameter in the file will have a keyword for identification in the file for ease of operator editing. There is one Model Parameter File per world model.

Model Log File: This file contains the logbook history of the world model and is written to by the operator over time. It is initially created by the MODEL-CREATE menu function from a default Log file and can be subsequently edited by the MODEL-NOTEBOOK EDIT menu function. The file is an ASCII file. Each time a operator logs onto the system or creates/selects a world model, an entry is written to the log file containing the operator's name, date and time. There is one Model Log File per world model.

Geometric Objects List File: This file contains information about all the 3D polyhedral objects defined for the world model. It is initially created by the MODEL-CREATE menu function from a default file and can be subsequently edited by the OBJECT-INFO EDIT menu function. There is one Geometric Objects List File per world model. A similar file, called the *Geometric Objects Library* File, contains standard and user defined geometric object templates. These templates may be recalled and inserted into a view window. There is only one library file in the ICERVS system and all world models share that one file.

Tree File: This file contains the volumetric data (either physical or property) describing the mapped storage tank. It is stored as an octree data structure. The word *tree* is used throughout this manual to refer to the volumetric data described as an octree data structure. It is initially created by the DATA-SAVE menu function. There may be several Tree Files per world model.

Saved View File: This file contains all the information necessary to describe one view window. It is initially created by the VIEW-SAVE VIEW or VIEW-SAVE ALL menu functions. The Saved View Files are managed by a Saved View Index File which

contains the names of all the Saved View Files which are to be stored together, allowing multiple views to be saved and restored as a group. There may several Saved View Files per world model.

The following describes the MODEL sub-menu functions in detail:

MODEL-SELECT (*Select a current working model*): This function allows selection from a list of available world models (workspace configurations) via a list selection dialog window. The selected world model will become the working model in the system. The callback function is CIcerMainWindow::ModelSelect().

MODEL-CREATE: (*Create a new model*): This function creates a new model directory with default world model files in it. The operator is prompted for a model name to identify the model being created. The newly created model will become the working model in the system. The operator may wish to edit the model parameter file at this time by clicking on the MODEL-PARAMETERS EDIT menu function to customize the tank configuration parameters. The operator may also wish to edit the model log file at this time by clicking on the MODEL-NOTEBOOK EDIT menu function to make an entry in the model historical log. The callback function is CIcerMainWindow::ModelCreate().

MODEL-DELETE: (*Delete an existing model*): This function removes an existing world model directory and its files. The operator is prompted to confirm verification before deletion occurs. If the current working directory is the one being deleted, the MODEL-SELECT function will be automatically invoked for the operator to select a new working model. The callback function is CIcerMainWindow::ModelDelete().

MODEL-NOTEBOOK EDIT: (*Edit the model notebook log file*): This function allows editing of the current model's notebook log file via a scrollable multi-line edit dialog window. The notebook log file contains automatic entries when the model is created or selected. The file is meant to be used as a logbook of historical information concerning the model (tank configuration). The callback function is CIcerMainWindow::ModelNotebookEdit().

MODEL-NOTEBOOK PRINT: (*Print the model notebook log file*): This function allows printing of the current model's notebook log file. The notebook log file contains automatic entries when the model is created or selected. The file is meant to be used as a notebook of historical information concerning the world model (tank configuration). The callback function is CIcerMainWindow::ModelNotebookPrint().

MODEL-PARAMETERS EDIT: (*Edit the model parameter file*): This function allows editing of the current model's parameter file via a scrollable multi-line edit dialog window. The model parameter file contains all the necessary information pertaining to

the world model (tank configuration) such as tank size, units, and default settings. The file being edited is an ASCII file so caution should be taken when editing to maintain the proper format of the file. The callback function is CIcerMainWindow::ModelParametersEdit().

MODEL-PARAMETERS PRINT: (*Print the model parameter file*): This function allows printing of the current model's parameter file. The model parameter file contains all the necessary information pertaining to the world model (tank configuration) such as tank size, units, and default system settings. The callback function is CIcerMainWindow::ModelParametersPrint().

MODEL-QUIT: (*Exit the ICERVS system*): This function closes all windows and exits the ICERVS system. The callback function is CIcerMainWindow::QuitIcervs().

4.1.4.1.2 **VIEW Sub-Menu Functions**: VIEW sub-menu functions consist of all the functionality needed to manage a view window in the ICERVS system. A view window contains a 2D display of the volumetric data as well as a 2D representation of the 3D polyhedral geometric objects. There can be as many view windows as desired in the ICERVS system. The following describes the VIEW sub-menu functions in detail:

VIEW-CREATE: (*Create a new view window*): This function creates a new view window and allows the operator to select an existing tree file or a new tree file (named NEW.TRE) to be displayed in the newly created view via a filelist selection dialog window. If the operator selects to display a new tree file, a name will be prompted for to identify the new tree file; however, a tree file will not be created until the operator explicitly saves the tree with the DATA-SAVE view menu function. The callback function is CIcerMainWindow::ViewCreate().

VIEW-SELECT: (*Select a current view window*): This function allows selection of a view window from a list of existing views via a list selection dialog window. A view automatically becomes the current view when the mouse is moved over it; however, if a view window becomes completely hidden behind another view, the VIEW-SELECT menu function will cause the selected view to pop to the top. The callback function is CIcerMainWindow::ViewSelect().

VIEW-CLOSE: (*Close a view window*): This function closes a selected view window. A list selection dialog window is displayed for the operator to select the view window to be closed. If the tree in the selected view window has changed, the operator will be prompted to save the tree to a file before closing the view window via a filelist dialog window. The callback function is CIcerMainWindow::ViewClose().

VIEW-RESTORE VIEW: (*Open a previously saved view*): This function allows selection of a previously saved view file from a list of existing view files via a filelist selection dialog window. Saved view files contain tree information (i.e. tree file used) as well as view context information (i.e. current display options). A saved view file may contain one or many previously saved view windows. When a saved view file has been selected, the view window(s) will be displayed on the screen exactly as they were when they were originally saved with the VIEW-SAVE VIEW menu function. The callback function is CIcerMainWindow::ViewRestore().

VIEW-SAVE VIEW: (*Save a current view*): This function saves a selected view window to a saved view file. A list selection dialog window is displayed for the operator to select the view window to be saved. A filelist selection dialog window is then displayed for the operator to enter a saved view file name in which to save the selected view. Saved view files contain tree information (i.e. tree file used) as well as view context information (i.e. current display options). The saved view can later be restored with the VIEW-RESTORE VIEW menu function. The callback function is CIcerMainWindow::ViewSave().

VIEW-SAVE ALL: (*Save all views*): This function saves all displayed views to a saved view file. A filelist selection dialog window is displayed for the operator to enter a saved view file name in which to save the views. Saved view files contain tree information (i.e. tree file used) as well as view context information (i.e. current display options) for each saved view. The saved views can later be restored with the VIEW-RESTORE VIEW menu function. The callback function is CIcerMainWindow::ViewSaveAll().

4.1.4.1.3 <u>WINDOW Sub-Menu Functions</u>: WINDOW sub-menu functions consist of all the functionality needed to manage the windows in the ICERVS system. There is always one main window in the system. There may be one or more view windows as well as some other extraneous windows at any point in time. The following describes the WINDOW sub-menu functions in detail:

WINDOW-DISPLAY: (*Display all windows*): This function allows selection of a windows from a list of existing windows via a list selection dialog box. The list of window contains all view windows and any other window (i.e.: sensor window, interface window, etc.) that may be currently displayed. A window automatically becomes the current window when the mouse is moved over it; however, if a window becomes completely hidden behind another window, the WINDOW-DISPLAY menu function will cause the selected window to pop to the top. The callback function is CIcerMainWindow::WindowSelect().

4.1.4.1.4 **HELP Sub-Menu Functions**: HELP sub-menu functions consist of all the functionality needed to get HELP for the ICERVS system. The following describes the HELP sub-menu functions in detail:

HELP-EXTENDED: (*Display extended HELP for the system*): This function displays a scrollable multi-line browse dialog window containing system wide HELP information. The callback function is CIcerMainWindow::HelpExtended().

HELP-INDEX: (*Display HELP index*): This function displays an index of HELP information in a list selection dialog window. When the operator selects the desired HELP index item, all information about that item will be displayed in a scrollable multi-line browse dialog window. The callback function is CIcerMainWindow::HelpIndex().

HELP-ABOUT: (*Display ABOUT ICERVS information*): This function displays an ABOUT message for the ICERVS system in an about dialog window. The callback function is CIcerMainWindow::HelpAboutIcervs().

4.1.4.2 View Window Menu Functions

The view windows are created from the main window's VIEW-CREATE or VIEW-RESTORE menu functions. See Figure 4-6 for an example of a View Window. View windows contain volumetric data that represent the 3D surface of the storage tank. As many view windows as desired can be created; however, only 4 view windows will fit on the screen before window overlapping occurs and window resizing becomes necessary.

Each view can contain unique volumetric data or the same volumetric data as another view. Different views that contain the same volumetric data are "linked" in the sense that certain display attributes of one view will be echoed in all other linked views. The display attributes that are linked are: data colors and cutplanes. Also, when any ADD menu function is invoked, the function will also be automatically performed on all linked views.

All view windows contain scrolling controls on the right and bottom of the window which are used for panning or translating the data. They also contain a slider bar on the left of the window which is used for zooming or scaling the image. When the data is first displayed on the view, the data is displayed at full scale and no translation is necessary. However, as the data is scaled, the data may need to be translated in order to view the region of interest.

Each view window's title bar contains a view name (view 1, view 2, etc.), as well as a volumetric data name (the name of the tree file which is currently displayed in the view. Each view's title bar is color coded to represent the type of orthogonal view that is being displayed (x, y, z, etc.).

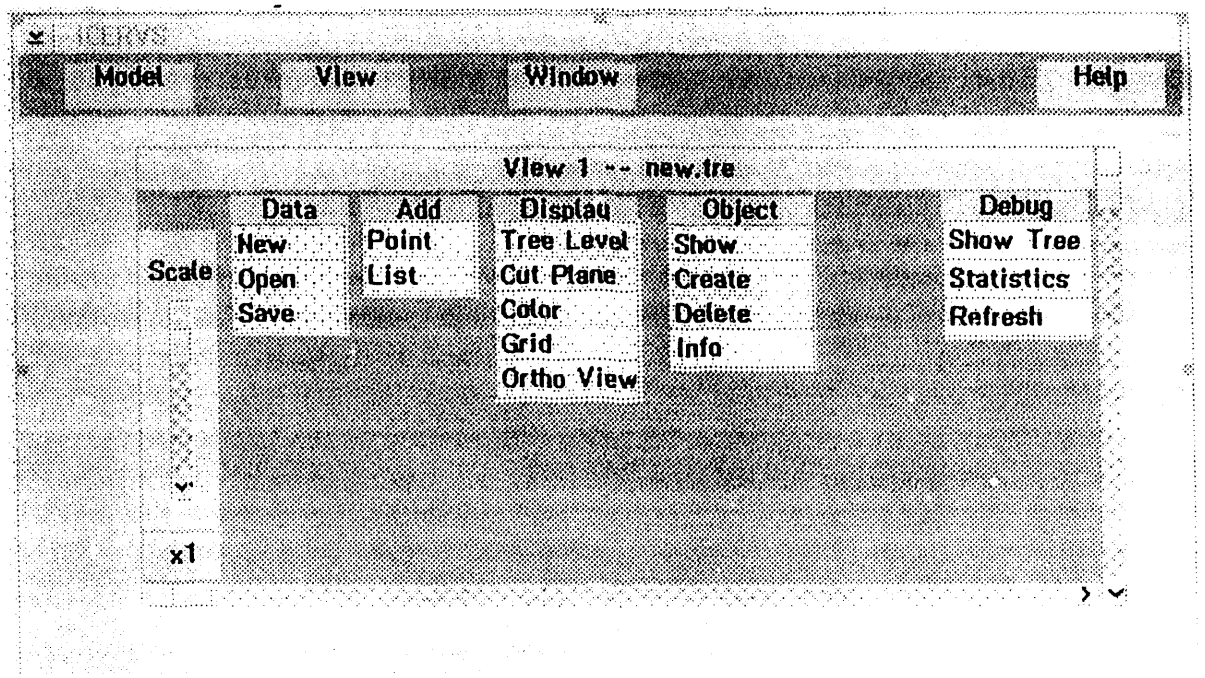


Figure 4-6 - ICERVS View Window

The view window menu functions and their subfunctions are listed below:

DATA	ADD	DISPLAY	OBJECT	DEBUG
NEW	POINT	TREE LEVEL	SHOW...ON	SHOW TREE
OPEN	LIST	CUTPLANE...ON	OFF	STATISTICS
SAVE		OFF	CREATE	REFRESH
		COLOR	DELETE	
		GRID.....ON	INFO...EDIT	
		OFF	PRINT	
		ORTHO VIEW	PRINT ALL	

4.1.4.2.1 DATA Sub-Menu Functions: DATA sub-menu functions consist of all the functionality needed to manage the tree files which contain volumetric data. The following describes the DATA sub-menu functions in detail:

DATA-NEW: (*Open a new tree file*): This function clears the view window of any currently displayed volumetric data and replaces it with a new tree file which contains no volumetric data. It also resets all display attributes back to the system defaults. If a tree file is displayed in the view before this function is invoked, the operator will be asked if the current volumetric data should be saved to a tree file first. When the function is invoked, a name will be prompted for to identify the new tree file, however, a tree file will not be created until the operator explicitly saves the tree with the DATA-SAVE menu function. This name will be displayed in the view window's title bar along with the view name. The callback function is CIcerViewWindow::DataNew().

DATA-OPEN: (*Open an existing tree file*): This function allows the operator to select via a filelist selection dialog window the tree file (which contains volumetric data) to be displayed in the view. It also resets all display attributes back to the system defaults. If a tree file is displayed in the view before this function is invoked, the operator will be asked if the current volumetric data should be saved to a tree file first. The selected tree filename will be displayed in the view window's title bar along with the view name. The callback function is CIcerViewWindow::DataOpen().

DATA-SAVE: (*Save the volumetric data to a tree file*): This function saves the view's volumetric data to a tree file. A filelist selection dialog window is displayed for the operator to select the tree file in which to save the data. The callback function is CIcerViewWindow::DataSave().

4.1.4.2.2 ADD Sub-Menu Functions: ADD sub-menu functions consist of all the functionality needed to add volumetric data to the ICERVS system. The following describes the ADD sub-menu functions in detail:

ADD-POINT: (*Add a point to the volumetric data*): This function allows the operator to interactively add a point to the volumetric data. An input dialog window is displayed for the operator to enter the x,y,z and property values for the data point to be added. The entered point is then displayed on the view and all other linked views. The callback function is CIcerViewWindow::AddPoint().

ADD-LIST: (*Add list of points to the volumetric data*): This function allows the operator to add a list of points contained in an ASCII file to the volumetric data. A filelist selection dialog window is displayed for the operator to enter the name of the ASCII file containing the x,y,z and property values of each data point to be added. The entered points are then displayed on the view and all other linked views. The callback function is CIcerViewWindow::AddList().

4.1.4.2.3 DISPLAY Sub-Menu Functions: DISPLAY sub-menu functions consist of all the functionality needed to control the display attributes of the volumetric data in the view windows. The following describes the DISPLAY sub-menu functions in detail:

DISPLAY-TREE LEVEL: (*Get a new volumetric data level*): This function allows the operator to get the data level to be used when displaying the volumetric data. This data level pertains to the resolution of the view generated by the volumetric data. A list selection dialog window is displayed for the operator to select the level (1-9) which correlates to the number of pixels used to represent the data (i.e. level 5 = 32 pixels, level 9 = 512 pixels). This selection effects the resolution of the view generated, not the size of the display window. The data is then redisplayed using the new level. The callback function is CIcerViewWindow::DisplayTreeLevel().

DISPLAY-CUTPLANE: (*Enable / Disable a cutplane*): This function allows the operator to turn ON or OFF a set of cutplanes. When turned on, a default cutplane will be displayed at the left and right edges of the view window. They will appear as vertical lines attached to a readout of the axis position. The operator will move the lines to their desired positions by dragging the appropriate line across the view window with the mouse and dropping it at the desired cutplane location. The data will be redisplayed with only the data between the 2 cutplanes visible. If the 2 cutplanes should cross as they are being defined, then only the data outside the cutplanes will be visible. A cutplane's position will affect the displayed data in all linked views; however, the act of turning on or off cutplanes is view specific. The callback function is CIcerViewWindow::DisplayCutPlane().

DISPLAY-COLOR: (*Define the volumetric data colors*): This function allows the operator to define the colors used to display the volumetric data (i.e. filled nodes, empty nodes, partial nodes, unknown nodes). This will be done via radio buttons in a dialog window. The volumetric data is then redisplayed in the view using the new colors, as well as in all other linked views. Note that colors having to do with the windows are changeable only from the MODEL-PARAMETERS EDIT menu function. The callback function is CIcerViewWindow::DisplayColor().

DISPLAY-GRID: (*Set the grid*): This function allows the operator to turn a grid ON or OFF via a 2nd level menu. The callback function is CIcerViewWindow::DisplayGrid().

DISPLAY-ORTHOGONAL VIEW: (*Set the orthogonal view*): This function allows the operator to select the orthogonal view to be used when displaying the volumetric data in the view window. A list selection dialog window will be displayed which contains the view choices (x,y,z,etc.). Each view type will have a different color title bar to allow easy identification of the view type. When the orthogonal view has been selected, the volumetric data will be redisplayed using the new view type. The callback function is CIcerViewWindow::DisplayOrthoginalView().

4.1.4.2.4 OBJECT Sub-Menu Functions: OBJECT sub-menu functions consist of all the functionality needed to control the volumetric geometric objects in the view window. The following describes the OBJECT sub-menu functions in detail:

OBJECT-SHOW: (*Show the geometric objects*): This function allows the operator to turn ON or OFF the geometric objects which pertain to the current model (workspace configuration). Only those objects which are visible given the data display configuration are displayed. The callback function is CIcerViewWindow::ObjectShow().

OBJECT-CREATE: (*Create a geometric object*): This function allows the operator to create a geometric object on the view window. A list selection dialog window will be displayed for the operator to select the type of object to create (i.e. point, line, rectangle, circle, polygon, etc.). When the type of object has been selected, the object will appear on the view window and can be moved or resized on the view by selecting the sides and/or vertices of the object with the mouse. The OBJECT-INFO EDIT menu function will be automatically invoked to allow the operator to identify the newly created object. All other linked views will also show the newly created object. The object can be created in any orthogonal view, and then extruded in any of the other two views to give it dimension. For example, if the object is created in the X orthogonal view, the object in either of the other views (Y and Z) is shown as a collapsed rectangle which can be stretched to give it dimension. Each time the object is moved or sized, its effects will also be seen in the other linked views. This function is

disabled until the OBJECT-SHOW menu item is turned ON. The callback function is CIcerViewWindow::ObjectCreate()

OBJECT-DELETE: (*Delete a geometric object*): This function allows the operator to delete a geometric object currently displayed on the view window. The object must have been first selected with the mouse to identify the object to be deleted. The object will be deleted in all views which are currently showing the objects. This function is disabled until the OBJECT-SHOW menu item is turned ON. The callback function is CIcerViewWindow::ObjectDelete()

OBJECT-INFO EDIT: (*Display geometric object information*): This function allows the operator to edit an object's information via a scrollable multi-line edit dialog window. The object must have been first selected with the mouse to identify the object to be edited. The object information includes such things as name, color, description, category, and data vertices. When an object is created, this function is automatically invoked to allow the operator to identify the newly created object. This function is disabled until the OBJECT-SHOW menu item is turned ON. The callback function is CIcerViewWindow::ObjectEdit()

OBJECT-INFO PRINT: (*Print selected geometric object info*): This function allows the operator to display and optionally print an object's information via a scrollable multi-line browse dialog window. The object must have been first selected with the mouse to identify the object to be printed. The object information includes such things as name, color, description, category, and data vertices. This function is disabled until the OBJECT-SHOW menu item is turned ON. The callback function is CIcerViewWindow::ObjectPrint()

OBJECT-INFO PRINT ALL: (*Print all geometric objects info*): This function allows the operator to display and optionally print each object's information via a scrollable multi-line browse dialog window. The object information includes such things as name, color, description, category, and data vertices. The callback function is CIcerViewWindow::ObjectPrintAll()

4.1.4.2.5 **DEBUG Sub-Menu Functions**: DEBUG sub-menu functions consist of all the functionality needed for debugging the octree data in the view window. The following describes the DEBUG sub-menu functions in detail:

DEBUG-SHOW TREE: (*Show the tree nodes*): This function allows the operator to view the tree node information via a scrollable multi-line browse dialog window. This function is used for debug purposes only. The callback function is CIcerViewWindow::DebugShowTree().

DEBUG-STATISTICS: (*Show the tree statistics*): This function allows the operator to view the tree statistics information via a scrollable multi-line browse dialog window. This function is used for debug purposes only. The callback function is CIcerViewWindow::DebugStatistics().

DEBUG-REFRESH: (*Refresh the view*): This function allows the operator to redraw everything in the view window. This function is used for debug purposes only. The callback function is CIcerViewWindow::DebugRefresh().

4.2 Octree Engine Computer Software Component (OE-CSC)

The OE-CSC encompasses all functions that relate to the representation, storage and management of volumetric data. Each world model (workspace) in the ICERVS system will contain one or more octree based data sets which represents some volumetric aspect of the model.

4.2.1 Preliminary Design Requirements and Functions

The requirements for the OE-CSC are summarized in Table 4-5. For a more detailed description of each requirement, refer to the ICERVS System Design Report. Requirements that apply to Phase I are in bold type. Requirements that not part of Phase I requirements but strongly influence the software design are italicized. Some requirements that span multiple phases have been reworded to clarify the Phase I requirement. These requirements are marked with a asterisk. Requirements that span multiple CSCs are also identified.

Table 4-5 – ICERVS Requirements for the Octree Engine CSC

System Requirement Number	Description
R1.01 *	**Octree representation, spatial data**
R1.02 *	*Octree representation, property data*
R1.03	*Octree representation, spatial interpolation*
R1.04	**Octree representation: linear resolution of 1:512**
R3.06	**Update octree representation as input points received** (shared with UI-SCS)
R5.01	Copy octree
R5.02	*Set region within octree to selected state (shared with UI-CSC)*
R5.04	Scan model for consistency - no suspended objects (shared with OM-CSC)
R5.05	Compute volumetric difference between octree and object (shared with OM-CSC)
R5.06	Compute volumetric difference between two octrees

Table 4-5 -- ICERVS Requirements for the Octree Engine CSC (continued)

R5.07	Compute 2.5D surface map as projection of 3D data
R5.08	Compute absolute and difference 2.5D surface maps
R6.02	**Save/retrieve models to/from disk (Distributed among CSCs)**
R6.03	Rebuild octree from backup raw data points (shared with UI-CSC)
R7.07	Output: waste surface maps, absolute and difference

4.2.2 Derived Requirements and Functions

Derived requirements relate to the specialization of the basic requirements such that basic functions and primitives are readily identifiable. For the OE-CSC, the approach taken is to identify the primary requirement and then to list the individual derived requirements. Several basic requirements that are not part of Phase I have a strong impact on the design of the OE-CSC; these requirements were italicized in Table 4-5 and are also italicized below. For completeness and understandability, those requirements are treated as though they were part of the Phase I requirements.

4.2.2.1 R1.01 Octree representation - spatial data

a. Create and delete an octree
b. Set/Get octree volumetric dimensions (i.e. the tank size)
c. Create/Delete/Maintain tree nodes
d. Link nodes together to form a tree
e. Conversion of ICERVS internal engineering units to/from tree units
f. Perform depth first tree traversals for display, printing, etc.

4.2.2.2 *R1.02 Octree representation - property data*

a. *Store/recover property data in/from tree*
b. *Store/recover property data in/from tree nodes*

4.2.2.3 *R1.03 Octree representation - spatial interpolation*

a. *Identify undefined regions of tree*
b. *Perform an interpolation to fill in tree gaps*

4.2.2.4 R1.04 Octree representation - linear resolution of 1:512

a. Set/Get the number of levels in the tree
b. Default level for adding points / display tree is 9

4.2.2.5 R3.06 Update octree representation as input points received

a. Add single data points as real data
b. Add list of data points as real data
c. Add and sculpt single data points as real data
d. Add and sculpt list of data points as real data
e. Add display attributes (i.e. property values) at data points

4.2.2.6 *R5.02 Set region within octree to selected state (shared with UI-CSC)*

a. *Convert region to list of tree nodes*
b. *Traverse node list and redefine node state*

4.2.2.7 R6.02 Save/retrieve models to/from disk (Distributed among CSCs)

a. Read/write octree data from/to disk
b. Read/write octree node data from/to disk

4.2.3 Class Descriptions

The OE-CSC is implemented in C++ and consists of approximately eleven (11) classes. The C++ class provides a mechanism for combining the data and the manipulation procedures related to a high-level entity into a single construct. Classes facilitate abstraction (ignoring details of processes and how data is represented), promote encapsulation (hiding of the internal workings of entities) and support inheritance (defining new entities as specializations of other entities). The first step in object-oriented design is to identify the classes. Later steps involve assignment of attributes and behavior, identification of relationships between classes and arrangement of the classes into hierarchies. This section identifies the OE-CSC classes and discusses their general characteristics (attributes, behavior and relationships). The last subsection (4.2.3.4) will describe the relationship of the classes to the basic and derived requirements defined in previous sections (4.2.1 and 4.2.2). Section 4.2.4 will discuss the major functions assigned to the OE-CSC and describe how the software classes implement the functions.

To simplify the discussion of the OE-CSC software, these classes have been organized into three (3) groups.

1. Tree related classes
2. Scaling class
3. Tree traversal classes

Two classes are included in the Tree Traversal classes that support octree debugging. These classes are COctStats and COctPrint.

The following sections describe the purpose and function of each group and each class within the group. Figure 4-7 provides illustrations of the interactions among the OE-CSC classes.

4.2.3.1 Tree Related Classes

Tree related classes are all concerned with the overall structure and management of trees. That is creation/deletion of trees; creation/deletion/maintenance of tree nodes, and addition of data to the tree.

COctNode: This class implements an octree node object. Each node of the tree contains eight (8) octants and the methods to access and add child nodes.

CNodeData: This class defines the data for a node of an octree. This allows a method of storing the node data on any medium, and the data may be modified at anytime. The node data could be stored on disk, for example. All accesses of the data must be done by access method.

CNodeNext: This class implements the pointer to the next element in the tree. This is implemented as a class so that no other object will rely on the next node being in memory. This separation allows the tree to reside partially on disk.

CCube: This class implements a cube region that knows how to divide itself into octants. This facilitates the process of subdividing when data is added at high resolutions. A CCube object stores the geometric position associated with a COctNode.

COctree: This is a class derived from COctNode. It is used to define the root of an octree. When COctree is constructed it contains the first level of octants to which all new nodes are attached.

4.2.3.2 Scaling Class

The Scaling class performs conversions of ICERVS internal units to tree units. This is necessary because the octree units are always positive integers, and the ICERVS internal units may be any real units (e.g. meters, feet, etc.). The internal units chosen for ICERVS are SI units. The conversion to/from ICERVS internal units to external user units is the concern of the UI-CSC.

CTreeScaling: This class implements a linear scaling transformation which converts between internal and tree units. Scaling is performed as data is added to the tree and it is unscaled when output from the tree. The octree space is an integer space of dimension 2,097,152 (21 bits). The ICERVS internal coordinate system uses SI units and has its origin at (0,0,0). CTreeScaling implements these conventions and provides the necessary transformations between coordinate spaces. The conversion between ICERVS internal units and the users external units is performed in the the UI-CSC.

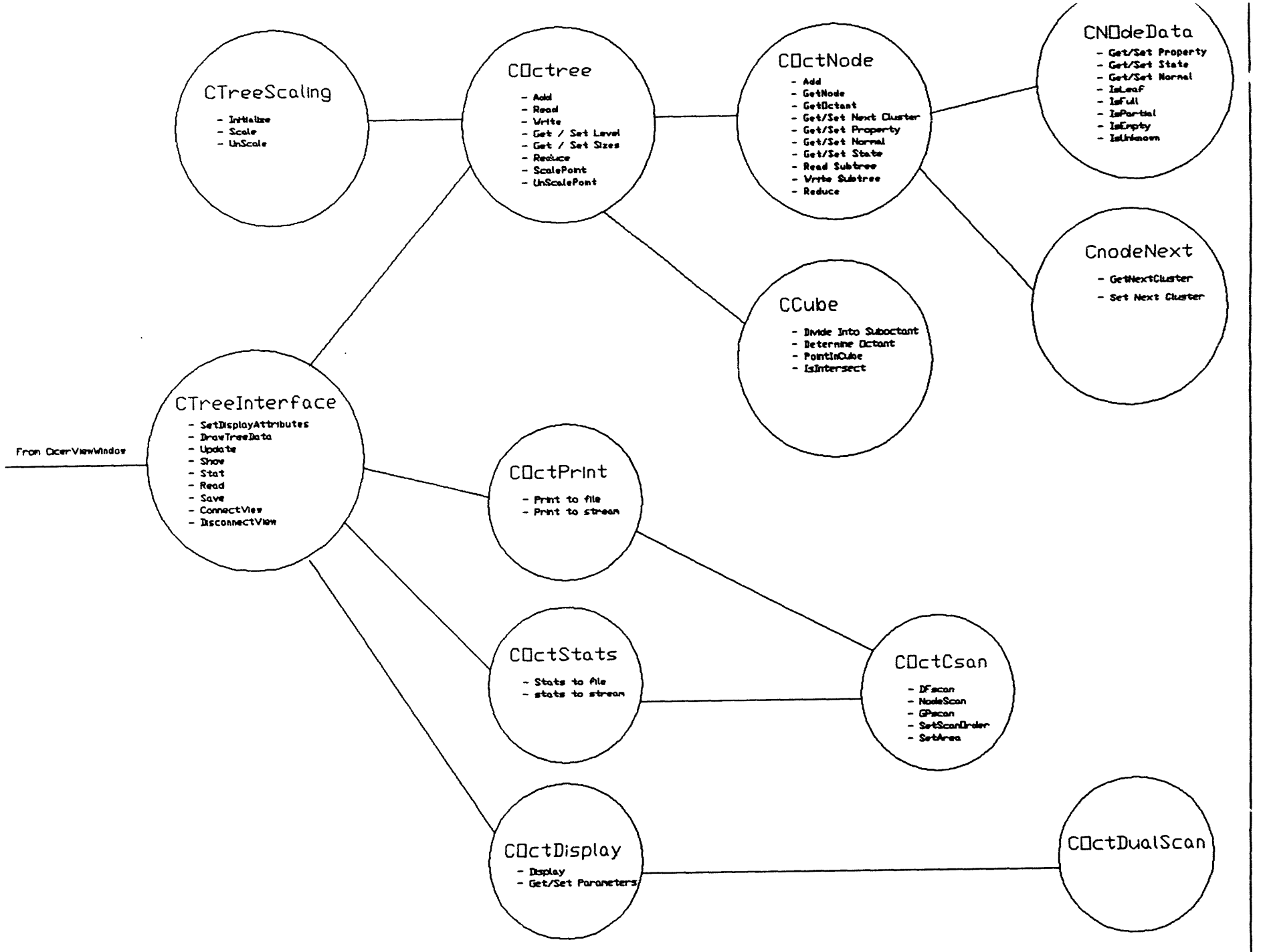


Figure 4-7 -- Octree Engine Class Interactions

4.2.3.3 Tree Traversal Classes

The tree traversal classes provide the basis for accessing data in the tree.

COctScan: This is an abstract class that is the basis for traversing an octree. The scanning procedure traverses the tree in a depth-first order and calls a method (pScanProc) for each octant encountered. A derived class must override this method to implement the desired function. The octants and suboctants are scanned in specified order to account for a different point of view or other requirements.

COctDualScan: This class scans two trees in synchronization. The octants of each tree can be scanned in a specified order which may be different for each tree. The scanner can optionally extend one or both trees so that their topology matches. The initial use for this class is to support COctDisplay. In later phases, this class will support comparisons among multiple trees.

COctPrint: This class is derived from COctScan and is used to print a graphical representation of the tree.

COctStats: This class is derived from COctScan and is used to accumulate and print statistics about the tree.

COctDisplay: This class is derived from COctDualScan and is used to traverse the tree and display the tree data in a CIcerViewWindow.

4.2.3.4 Relationship Between OE-CSC Requirements and Classes

Previous sections have enumerated the ICERVS basic requirements for the OE-CSC, defined a set of derived requirements for each basic requirement and identified a set of software classes for the OE-CSC. This section will define the relationships between the requirements and the classes.

Table 4-6 identifies the classes that implement each ICERVS basic requirement. Requirements that apply to Phase I are in bold type. Requirements that not part of Phase I requirements but strongly influence the software design are italicized. Some requirements that span multiple phases have been reworded to clarify the Phase I requirement. These requirements are marked with an asterisk.

Table 4-7 identifies the ICERVS basic requirements related to each OE-CSC class.

Table 4-8 identifies the detailed relationships between the derived requirements and the OE-CSC classes. Most of the derived requirements relate to one data member or one function member of a single class. These relationships are denoted by the expression *class_name*::*data_member_name* or *class_name*::*function_name*().

Table 4-6 -- OE-CSC Classes For Each ICERVS Requirement

Requirement Number	Description
R1.01	**Octree representation, spatial data.** COctree COctScan COctNode COctDualScan CTreeScaling COctDisplay CNodeData COctStats CNodeNext COctPrint
R1.02	*Octree representation, property data.* COctree COctNode CNodeData
R1.03	*Octree representation, spatial interpolation.* COctree COctNode CNodeData
R1.04	**Octree representation, linear resolution of 1:512.** COctree COctNode
R3.06 *	**Update octree representation as input points received.** COctree COctNode CCube
R5.02	*Set region within octree to selected state.* COctree COctNode
R6.02	**Save/retrieve models to/from disk.** COctree COctNode

Table 4-7 -- ICERVS Requirements For Each OE-CSC Class

Class	Related Requirements
Tree Classes	
COctree	R1.01, R1.02, R1.03, R1.04, R3.06, R5.02,R6.02
COctNode	R1.01, R1.02, R1.03, R1.04, R3.06, R5.02,R6.02
CNodeData	R1.01, R1.02, R1.03
CNodeNext	R1.01
CCube	R3.06
Scaling Class	
CTreeScaling	R1.01
Tree Traversal Classes	
COctScan	R1.01
COctDualScan	R1.01
COctDisplay	R1.01
COctStats	R1.01
COctPrint	R1.01

Table 4-8 -- OE-CSC Detailed Required/Class Relationships

Requirement Number	Requirement Description and Detailed Class Relationships
R1.01	**Octree representation, spatial data.**
a.	COctree Constructor, Destructor
b.	COctree Constructor, COctree::SetSize(), GetSize()
c.	COctree::Add(), COctNode::GetNode()
d.	COctNode::GetNextCluster(), SetNextCluster, CNodeNext::GetNextCluster(), SetNextCluster()
e.	CTreeScaling::Scale(), Unscale(), COctree::ScalePoint(), UnScalePoint()
f.	COctDisplay::Display, COctPrint::Print(), COctStats::GetTheStats()
R1.02	*Octree representation, property data.*
a.	COctree::Add()
b.	COctNode::GetProp(),PutProp(), COctData::GetProp(), PutProp()
R1.03	*Octree representation, spatial interpolation.*
a.	COctTree::Interpolate()
b.	COctTree::Interpolate()
R1.04	**Octree representation, linear resolution of 1:512.**
a.	COctree::GetLevel(), SetLevel()
b.	COctree Constructor
R3.06 *	**Update octree representation as input points received.**
a.	COctree::Add(), COctNode::Add()
b.	COctree::Add(), COctNode::Add()
c.	COctree::AddAndSculpt(), COctNode::Add()
d.	COctree::AddAndSculpt(), COctNode::Add()
e.	COctree::Add(), COctNode::PutProp()
R5.02	*Set region within octree to selected state.*
a.	COctree::SetRegion()
b.	COctree::SetRegion()
R6.02	**Save/retrieve models to/from disk.**
a.	COctTree::Read(), Write()
b.	COctNode::ReadSubTree(), WriteSubTree()

4.2.4 Major Function Descriptions

The functions of the OE-CSC are used internally by the ICERVS software subsystem. No operator will directly interface with these functions. Instead, the operator will interact with the UI-CSC and the UI-CSC will activate functions from the OE-CSC on the operator's behalf. The OE-CSC software's major functions are described below:

4.2.4.1 Write Octree Data To Disk File

This function is initiated as a consequence of closing a view or opening a new tree for an existing view. In either case, if the tree data has changed, a file dialog box will prompt the user for the name of the tree file in which to save the tree data. A call is made to COctree::Write to transfer the data from the tree structure to the disk file. The octree data is written to disk in binary format matching the internal format of the COctNode class.

4.2.4.2 Read Octree Data From Disk File

This function is initiated as a consequence of creating a new view or opening a new tree for an existing view. In either case, a file dialog box will prompt the user for the name of the tree file to load into the view. A call is made to COctree::Read to transfer the data from disk to the tree structure.

4.2.4.3 Add Single Point To Octree

This function is initiated in the UI-CSC from the VIEW-ADD menu. A dialog box will be displayed so that the user can define the coordinates, property value and level for the new point. A call to COctree::Add is made to insert the new point into the tree. The data for the point must be in ICERVS internal units.

4.2.4.4 Add List Of Points To Octree

This function is initiated in the UI-CSC from the VIEW-ADD menu. A dialog box will be displayed so that the user can specify the filename for the list of points. A call to COctree::Add is made to insert a file of points into the tree. Note that the data contained in the file must be in ICERVS internal units. The UI-CSC must translate the user's datafile into a temporary file using ICERVS internal units. Otherwise, the UI-CSC must use the Add Single Point function (section 4.3.4.3) and add the user's points one at a time.

4.2.4.5 Add With Sculpting Single Point To Octree

This function is similar to the Add function (4.2.4.3). A dialog box will be displayed so that the user can define the coordinates, property value and level for the new point. A call to COctree::AddAndSculpt is made to insert the new point into the tree. When added, all points above the new point will be cleared. This function is used specifically when adding points from sensor data. As surface points are added, it is assumed that the space between the sensor and the surface is empty. Therefore, the state of nodes in the octree which represent that space

are automatically set to empty as the new data points are added to the tree. The data for the point must be in ICERVS internal units.

4.2.4.6 Add With Sculpting List Of Points To Octree

This function is similar to the AddList function (4.2.4.4). A dialog box will be displayed so that the user can specify the filename for the list of points. A call to COctree::AddAndSculpt is made to insert a file of points into the tree. Note that the data contained in the file must be in ICERVS internal units. The UI-CSC must translate the user's datafile into a temporary file using ICERVS internal units. Otherwise, the UI-CSC must use the Add With Sculpting Single Point function (section 4.3.4.5) and add the user's points one at a time. When added, all points above the new point will be cleared.

4.2.4.7 Perform Tree Scan To Print Tree Data

This function is initiated in the UI-CSC from the VIEW-DEBUG menu. A call to COctPrint::Print is made to scan the tree and print the data to an intermediate disk file. Upon return from the print routine, the UI-CSC will pass the intermediate file to a CBrowseWindow object for viewing. Data will be printed in ICERVS internal units.

4.2.4.8 Perform Tree Scan To Print Tree Statics

This function is initiated in the UI-CSC from the VIEW-DEBUG menu. A call to COctStat::GetTheStats is made to scan the tree and print the statistics data to an intermediate disk file. Upon return from the print routine, the UI-CSC will pass the intermediate file to a CBrowseWindow object for viewing.

4.2.4.9 Perform Tree Scan To Display Tree In View Window

This function is initiated in the UI-CSC as a result of one of the following:

1. Translation of any view
2. Rescaling of any view
3. Movement of the cutplanes in any view
4. Redefinition of view parameters (color, level, etc.)
5. Addition of new data in any view

A call is made to CTreeInterface::Display by the UI-CSC. This routine will extract view-specific parameters (window size, translation offsets, scaling factors, display level, cutplane locations, etc.) and pass these parameters to the associated octree using a variety of COctDisplay::Set/Get functions. In addition, the CTreeInterface::DisplayRoutine will pass the view shared parameters (node colors, etc.) to COctDisplay. Finally, COctDisplay::Display is called to scan the tree and display each visible octree point. The view parameters passed by CTreeInterface are used to determine the visibility of each tree point. The data for a point (coordinates, property, state, etc.) are sent to a view-specific DrawPoint() routine in ICERVS internal units. However, the data will be displayed in the user's external units.

4.3 Object Modeling Computer Software Component (OM-CSC)

The OM-CSC encompasses all functions relating to the representation, storage, and management of 3D geometric objects. Geometric objects are an arbitrary collection of polyhedral objects and primitives with attached text and other attributes.

Two sets of geometric objects are maintained by the system:

1) The Geometric Objects List contains all the geometric objects for a particular world model (work space). A separate list exists for every world model.

2) The Geometric Objects Library contains templates for objects that can be recalled by the user and placed into the current world model. A single library exists in the ICERVS system.

4.3.1 Preliminary Design Requirements and Functions

The requirements for the OM-CSC are summarized in Table 4-9. For a more detailed description of each requirement, refer to the ICERVS System Design Report. Requirements that apply to Phase I are in bold type. Requirements that not part of Phase I requirements but strongly influence the software design are italicized. Some requirements that span multiple phases have been reworded to clarify the Phase I requirement. These requirements are marked with an asterisk. Requirements that span multiple CSCs are also identified.

Table 4-9 -- System Requirements for the Object Modeling CSC

System Requirement Number	Description
R1.05 *	**Geometric models, polyhedral objects**
R1.06	*Geometric models, geometric primitives*
R1.07	**Geometric models, associated text each object**
R1.08	**Geometric models, support at least 100 objects, expansion capability**
R1.09	Geometric models, enter architectural plans and robot descriptions
R2.01 *	*Create, modify and store primitives / templates*
R2.02	Standard templates: station, registration targets, core sample

Table 4-3 -- System Requirements for the Object Modeling CSC (continued)

R2.03	*Operator can define templates and add to library (shared with UI-CSC)*
R2.04	Automatic waste surface modeling (shared with OE-CSC)
R2.06	**Create, modify, delete 3D polyhedral objects (swept volume)**
R2.08	**Attach text to objects** (duplicates R1.07)
R3.05	**Geometric Objects: wire frame polygons (shared with UI-CSC)**
R3.08	Property (object category) derived coloring (shared with UI-CSC)
R5.03	*Operator delete objects. (shared with UI-CSC)*
R5.04	Scan model for consistency - no suspended objects. (shared with OE-CSC)
R5.05	Compute volumetric difference between octree and object (shared with OE-CSC)
R6.02	**Save/retrieve models to/from disk (Distributed among CSCs)**
R6.06	Define disassembly data (Shared with OM-CSC)
R7.06 *	**Output: geometric models as text reports**

4.3.2 Derived Requirements and Functions

Derived requirements relate to the specialization of the basic requirements such that basic functions and primitives are readily identifiable. For the OM-CSC, the approach taken is to identify the primary requirement and then to list the individual derived requirements. Several basic requirements that are not part of Phase I have a strong impact on the design of the OM-CSC; these requirements were italicized in Table 4-9 and are also italicized below. For completeness and understandability, those requirements are treated as though they were part of the Phase I requirements.

4.3.2.1 R1.05 Geometric models, polyhedral objects

a. Prismoids stored in model list
b. Set and get prismoid's front and rear vertices

4.3.2.2 *R1.06 Geometric models, geometric primitives*

a. *Cylinders stored in model list*
b. *Set and get cylinder's diameter, height, and center*

c. *Cones stored in model list*
d. *Set and get cones's diameter, height, and center*

e. *Planes stored in model list*
f. *Set and get planes's normal and point*

g. *Spheres stored in model list*
h. *Set and get sphere's diameter and center*

i. *Groups of geometric primitives stored in model list*
j. *Add object to group*
k. *Delete object from group*
l. *Get object from group*

4.3.2.3 **R1.07 Geometric models, associate text with each object**

a. Store text in model list along with geometric objects
b. Set and get object name
c. Set and get object notes
d. Set and get object creation date and time
e. Set and get operator name
f. Set and get object category

4.3.2.4 **R1.08 Geometric models, support at least 100 objects, expansion capability**

a. Store geometric objects in disk file limited only by available disk space

4.3.2.5 *R2.01 Create, modify, and store primitives / templates*

a. *Store predefined set of geometric objects in the Geometric Object Library*
b. *Get a geometric object from the library*
c. *Add a new geometric object to the library*
d. *Replace an existing geometric object in the library*
e. *Delete an existing geometric object in the library*

4.3.2.6 *R2.03 Operator can define templates and add to library*

a. *Get predefined geometric object from Geometric Object Library*
b. *Interface with UI-CSC to create, modify, and display 2D convex polygons*
c. *Store user-defined geometric object to Geometric Object Library*

4.3.2.7 **R2.06 Create, modify, delete 3D polyhedral objects (swept volume)**

a. Store set of operator generated 3D polyhedral objects in the model list
b. Get a polyhedral object from the list
c. Add a new polyhedral object to the list
d. Replace an existing polyhedral object in the list
e. Delete an existing polyhedral object in the list

4.3.2.8 **R2.08 Attach text to objects (duplicates R1.07)**

a. Store text in geometric object list along with geometric objects
b. Set and get object name
c. Set and get object notes
d. Set and get object creation date and time
e. Set and get operator name
f. Set and get object category

4.3.2.9 **R3.05 Geometric Object - wire frame polygons**

a. Store geometric objects in model list as set of front and rear vertices
b. Interface with UI-CSC to display 2D representation of 3D geometric object

4.3.2.10 *R5.03 Operator delete objects*

a. *Interface with UI-CSC to select a 3D geometric object for deletion*
b. *Delete an existing polyhedral object in the Data List*

4.3.2.11 **R6.02 Save/retrieve models to/from disk**

a. Read/write the CModelObjectList from/to disk file (MODELOBJ.DIC)
b. Read/write the CModelObjectLibrary from/to disk file (MODELOBJ.LIB)
c. Read / write each geometric object entity type

4.3.2.12 *R7.06 Output - geometric models, text report*

a. *Print each geometric object entity type to file in human readable form*
b. *Write Geometric Object List contents to temporary disk file*
c. *Write Geometric Object Library contents to temporary disk file*
d. *Interface with UI-CSC to use general purpose text browsing window*

4.3.3 Class Descriptions

The OM-CSC is implemented in C++ and consists of approximately nine (9) classes. The C++ class provides a mechanism for combining the data and the manipulation procedures related to a high-level entity into a single construct. Classes facilitate abstraction (ignoring details of processes and how data is represented), promote encapsulation (hiding of the internal workings of entities) and support inheritance (defining new entities as specializations of other entities). The first step in object-oriented design is to identify the classes. Later steps involve assignment of attributes and behavior, identification of relationships between classes and arrangement of the classes into hierarchies. This section identifies the OM-CSC classes and discusses their general characteristics (attributes, behavior and relationships). The last subsection (4.3.3.3) will describe the relationship of the classes to the basic and derived requirements defined in previous sections (4.3.1 and 4.3.2). Section 4.3.4 will discuss the major functions assigned to the OM-CSC and describe how the software classes implement the functions.

To simplify the discussion of the OM-CSC software, these classes have been organized into two groups: 1) Collection classes and 2) Object classes. In addition, several of the classes from the Utility group of classes (see UI-CSC above) are used by the OM-CSC.

The sections that follow describe the purpose and function of each group and each class with the group. Figure 4-8 provides an illustration of the interactions among the OM-CSC classes.

4.3.3.1 Collection Classes

The OM-CSC maintains two collections of geometric objects: the objects in the work volume and a library of object templates. These two collections are similar, but are encapsulated in two different classes.

CModelObjectList: A class that contains all the geometric objects for the current word model and encapsulates all accesses to the set of geometric objects. An instance of CModelObjectList is created and owned by a CModelObjectInterface instance.

CModelObjectLibrary: A class that contains templates for geometric objects and encapsulates all accesses to the set of templates. An instance of CModelObjectLibrary is created and owned by a CModelObjectInterface instance.

4.3.3.2 Object Classes

Many different types of object classes are required to represent the various types of geometric objects.

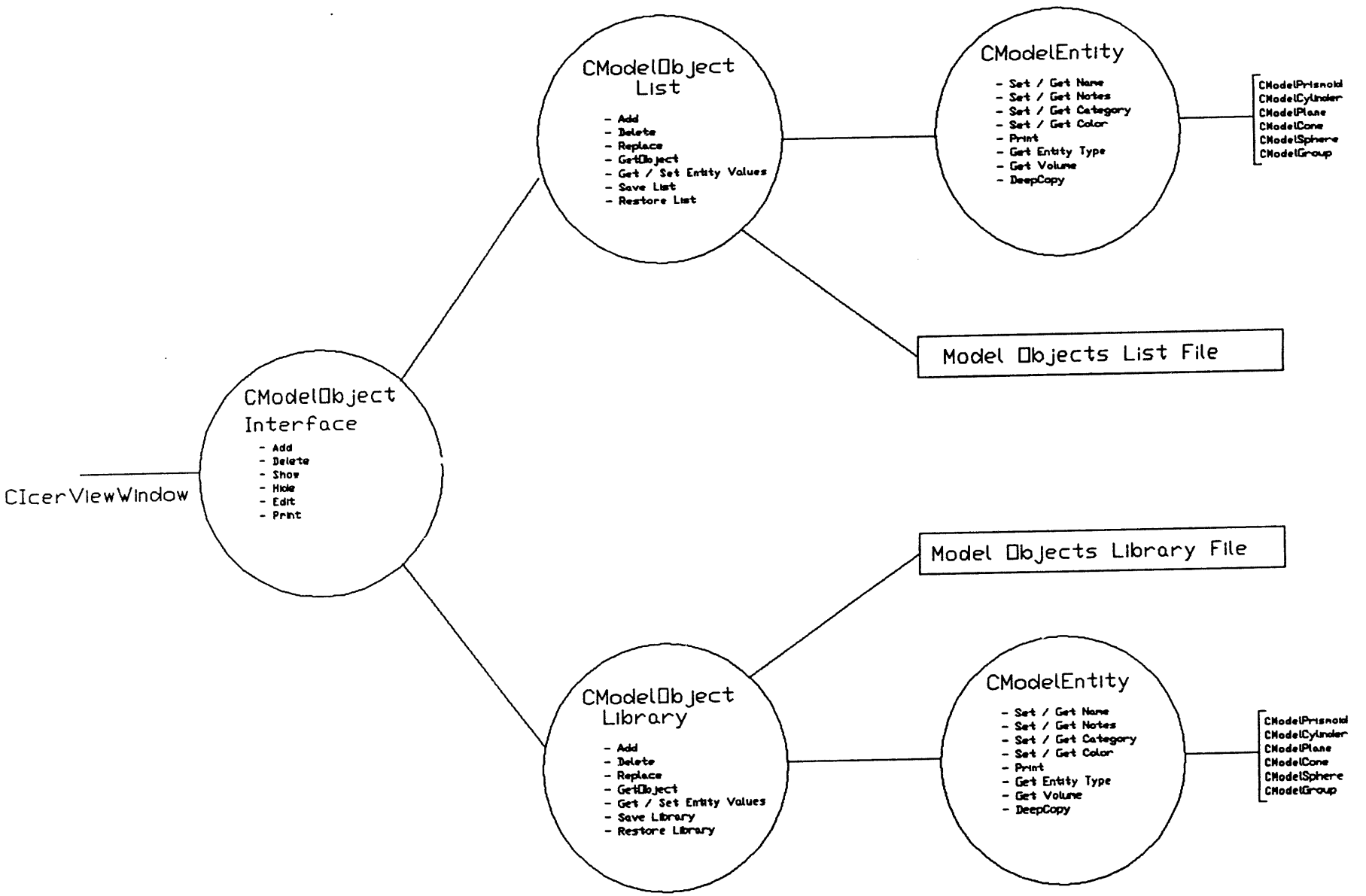


Figure 4-8 -- Object Modeling CSC Class Interactions

CModelEntity: The abstract base class for all geometric object classes. This class provides all the common behavior for geometric objects, including a common type that can be handled by the CModelObjectList and CModelObjectLibrary classes. The class data members include object name, type, notes, creation date/time, operator identification, object category name, and color of the object. All other geometric object classes must be derived from CModelEntity.

CModelPrismoid: A class that implements a type of geometric object characterized by two parallel polygonal faces and orthogonal facets. The two faces have the same number of vertices. The extra data members of this class include the list of coordinates for the front and rear face vertices.

CModelCylinder: A class that implements a geometric cylinder type of geometric object. The extra data members of this class include the center of the base, the diameter of the base and the height of the cylinder.

CModelCone: A class that implements a geometric cone type of geometric object. The extra data members of this class include the center of the base, the diameter of the base and the height of the cone.

CModelSphere: A class that implements a geometric sphere type of geometric object. The extra data members of this class include the center of the sphere and its diameter.

CModelPlane: A class that implements a planar geometric object. The extra data members of this class include a point on the plane and the normal vector to the plane.

CModelGroup: A class that implements a composite geometric object that is composed of several other geometric objects. The extra data members of this class include a list of other objects that comprise the group.

4.3.3.3 Relationship Between OM-CSC Requirements and Classes

Previous sections have enumerated the ICERVS basic requirements for the OM-CSC, defined a set of derived requirements for each basic requirement and identified a set of software classes for the OM-CSC. This section will define the relationships between the requirements and the classes.

Table 4-10 identifies the classes that implement each ICERVS basic requirement. Requirements that apply to Phase I are in bold type. Requirements that are not part of Phase I requirements but strongly influence the software design are italicized. Some requirements that span multiple phases have been reworded to clarify the Phase I requirement. These requirements are marked with an asterisk.

Table 4-11 identifies the ICERVS basic requirements related to each OM-CSC class.

Table 4-12 identifies the detailed relationships between the derived requirements and the OE-CSC classes. Most of the derived requirements relate to one data member or one function member of a single class. These relationships are denoted by the expression *class_name*::*data_member_name* or *class_name*::*function_name*().

Table 4-10 -- OM-CSC Classes For Each ICERVS Requirement

Requirement Number	Description
R1.05	**Geometric models, polyhedral objects** CModelObjectList CModelEntity CModelPrismoid
R1.06	*Geometric models, geometric primitives* CModelObjectList CModelEntity CModelCylinder CModelCone CModelSphere CModelPlane CModelGroup
R1.07	**Geometric models, associated text each object** CModelObjectList CModelEntity
R1.08	**Geometric models, support at least 100 objects, expansion capability** CModelObjectList
R2.01	*Create, modify and store primitives / templates* CModelObjectsLibrary CModelEntity
R2.03	*Operator can define templates and add to library* CModelObjectLibrary CModelGroup

Table 4-10 -- OM-CSC Classes For Each ICERVS Requirement (continued)

R2.06	**Create, modify, delete 3D polyhedral objects (swept volume)** CModelObjectsList
R2.08	**Attach text to objects** CModelEntity
R3.05	**Geometric Objects: wire frame polygons** CModelObjectList CModelEntity CModelPrismoid
R5.03	*Operator delete objects* CModelObjectList
R6.02	**Save/retrieve models to/from disk** CModelObjectList CModelObjectLibrary CModelEntity
R7.06 *	**Output - geometric models as text reports** CModelObjectList CModelObjectLibrary CModelObjectEntity

Table 4-11 -- ICERVS Requirements For Each OM-CSC Class

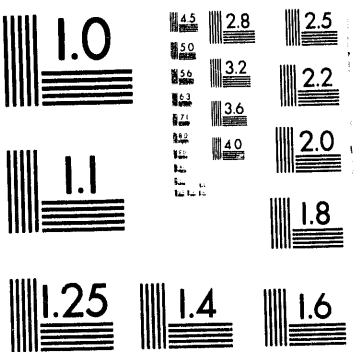
Class	Related Requirements
Collection Classes	
CModelObjectList	R1.05, R1.06, R1.07, R1.08, R2.06, R3.05, R5.03, R6.02, R7.06
CModelObjectLibrary	R2.03, R6.02, R7.06
Object Classes	
CModelEntity	R1.05, R1.06, R1.07, R2.01, R2.08, R3.05, R6.02, R7.06
CModelPrismoid	R1.05, R3.05
CModelCylinder	R1.06
CModelCone	R1.06
CModelSphere	R1.06
CModelPlane	R1.06
CModelGroup	R1.06

Table 4-12 -- OM-CSC Detailed Required/Class Relationships

Requirement Number	Requirement Description and Detailed Class Relationships
R1.05	**Geometric models, polyhedral objects.**
a.	CModelObjectList::Add(), Replace() CModelEntity::DeepCopy(), CModelPrismoid::DeepCopy()
b.	CmodelPrismoid::SetVertices(),SetFrontVertex, SetRearVertices(), GetFrontVertices(), GetRearVertices()
R1.06	*Geometric models, geometric primitives.*
a.	CModelModelObjectList::Add(), Replace(), CModelCylinder::DeepCopy()
b.	CmodelCylinder::GetDiameter(), GetHeight(), GetCenter(), SetDiameter(), SetHeight(), SetCenter()
c.	CModelModelObjectList::Add(),Replace(), CModelCone::DeepCopy()
d.	CModelCone::GetDiameter(),GetHeight(),GetCenter(), SetDiameter(), SetHeight(), SetCenter()
e.	CModelModelObjectList::Add(), Replace(), CModelPlane::DeepCopy()
f.	CModelPlane::GetNormal(), GetPoint(), SetNormal(), SetPoint()
g.	CModelModelObjectList::Add(), Replace(), CModelSphere::DeepCopy()
h.	CModelSphere::GetDiameter(), GetCenter(), SetDiameter(), SetCenter()
i.	CModelObjectList::Add(), Replace(), CModelGroup::DeepCopy()
j.	CModelGroup::AddObjectToGroup()
k.	CModelGroup::DeleteObjectFromGroup()
l.	CModelGroup::GetGroupObject()
R1.07	**Geometric models, associated text each object.**
a.	CModelObjectList::Add(),Replace()
b.	CModelObjectList::Set/GetName(), CModelEntity::Set/GetName()
c.	CModelObjectList::Set/GetNotes(), CModelEntity::Set/GetNotes()
d.	CModelObjectList::Set/GetCreateDate(),Set/GetCreateTime(), CModelEntity::Set/GetCreateDate(),Set/GetCreateTime()
e.	CModelObjectList::Set/GetOperatorID(), CModelEntity::Set/GetOperatorID()
f.	CModelObjectList::Set/GetCategory(),CModelEntity::Set/GetCategory()
R1.08	**Geometric models, support at least 100 objects, expansion capability.**
a.	CModelObjectList Constructor and Destructor

Table 4-12 -- OM-CSC Detailed Required/Class Relationships (continued)

R2.01	*Create, modify and store primitives / templates*
a.	CModelObjectLibrary::Add(), Replace()
b.	CModelObjectLibrary::GetObject()
c.	CModelObjectLibrary::Add(), CModelEntity::DeepCopy()
d.	CModelObjectLibrary::Replace(), CModelEntity::DeepCopy()
e.	CModelObjectLibrary::Delete(), DeleteAll()
R2.03	*Operator can define templates and add to library.*
a.	CModelObjectLibrary::GetObject()
b.	CModelObjectInterface::Add(), Display()
c.	CModelObjectLibrary::Add(), Replace(), CModelGroup::AddObjectToGroup()
R2.06	**Create, modify, delete 3D polyhedral objects (swept volume).**
a.	CModelObjectList::Add(), Replace()
b.	CModelObjectList::GetObject()
c.	CModelObjectList::Add(), CModelEntity::DeepCopy()
d.	CModelObjectList::Replace(), CModelEntity::DeepCopy()
e.	CModelObjectList::Delete(), DeleteAll()
R2.08	**Attach text to objects**
a.	CModelObjectList::Add(),Replace()
b.	CModelObjectList::Set/GetName(), CModelEntity::Set/GetName()
c.	CModelObjectList::Set/GetNotes(), CModelEntity::Set/GetNotes()
d.	CModelObjectList::Set/GetCreateDate(),Set/GetCreateTime(), CModelEntity::Set/GetCreateDate(),Set/GetCreateTime()
e.	CModelObjectList::Set/GetOperatorID(), CModelEntity::Set/GetOperatorID()
f.	CModelObjectList::Set/GetCategory(),CModelEntity::Set/GetCategory()
R3.05	**Geometric Objects: wire frame polygons.**
a.	CModelObjectList::Add(), Replace(), CModelPrismoid::DeepCopy(), SetVertices(), SetFrontVertex(), SetRearVertex(),
b.	CModelInterface::Display()
R5.03	*Operator delete objects.*
a.	CModelObjectInterface::Delete()
b.	CModelObjectList::Delete()


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1

2 of 2

Table 4-12 -- OM-CSC Detailed Required/Class Relationships (continued)

R6.02 a. b. c.	**Save/retrieve models to/from disk.** CModelObjectList::ReadtDictionaryFile(), WriteDictionaryFile() CModelObjectLibrary::ReadLibraryFile(), WriteLibraryFile() CModelEntity::SaveGuts(),RestoreGuts()
R7.06 * a. b. c. d.	**Output - geometric models as text reports** CModelEntity::Print() CModelObjectList::PrintTextReport() CModelObjectLibrary::PrintTextReport() CModelObjectInterface::Print()

4.3.4 Major Function Descriptions

The functions of the OM-CSC are used internally by the ICERVS software subsystem. No operator will directly interface with these functions. Instead, the operator will interact with the UI-CSC and the UI-CSC will activate functions from the OM-CSC on the operator's behalf. The OM-CSC software's major functions are described below:

4.3.4.1 Read Geometric Objects List/Library Files

This function is internal to the CModelObjectList and CModelObjectLibrary classes. When an instance of either class is created, the constructor for the object will read the appropriate disk file and load the CModelObjectList or CModelObjectLibrary with the object data. Since both the CModelObjectList and CModelObjectLibrary are both based upon Rogue Wave collection classes (RWOrdered), the Rogue Wave virtual stream I/O capability built into RWCollection-type classes will be used to write/read these collections. This Rogue Wave supplied facility is simple to use and ideal for storing the ICERVS geometric objects on disk.

4.3.4.2 Write Geometric Objects List/Library Files

This function is internal to the CModelObjectList and CModelObjectLibrary classes. When an instance of either class is destroyed, the destructor for the object will write the appropriate disk file and store the CModelObjectList or CModelObjectLibrary object data to disk. If no changes have been made to the object data, the file write process will be bypassed. Since both the CModelObjectList and CModelObjectLibrary are both based upon Rogue Wave collection classes (RWOrdered), the Rogue Wave virtual stream I/O capability built into RWCollection-type classes will be used to write/read these collections. This Rogue Wave supplied facility is simple to use and ideal for stored the ICERVS geometric objects on disk.

4.3.4.3 Conversion Of A 3D Geometric Object Into A Set Of 2D Displayable Objects

A 3D geometric object will be displayed as a set of 2D polygons in one or more view windows. Before display, it will be necessary to convert the 3D geometric object into the set of 2D displayable polygons. Each geometric object class will contain a GetGraphic() method to accomplish this conversion.

4.3.4.4 Conversion Of A Set Of 2D Displayable Objects Into A 3D Geometric object

A 3D geometric object will be displayed as a set of 2D polygons in one or more view windows. Before creating or updating the 3D geometric object, it will be necessary to convert the 2D polygon data. Each geometric object class will contain a SetGraphic() method to accomplish this conversion.

4.3.4.5 Add New 3D Object To List or Library

Addition of new geometric objects to the list or library involves only the creation of a new instance of CModelEntity (actually one of its derived classes) and the addition of the new instance to the list or library. The data from which to create the new object is supplied by the UI-CSC. The UI-CSC is also responsible for any interaction with the operator to define the required data.

4.3.4.6 Delete 3D Object From List or Library

Deletion of a geometric object from the list or library involves only the removal of the object from the list or library. The UI-CSC is responsible for interfacing with the operator to select the desired object. The association of the selected 2D objects with the appropriate 3D object is performed before the delete operation.

4.3.4.7 Modify 3D Object In List or Library

Modification of a geometric object in the list or library involves the extraction of the objects data, modification of that data, and re-insertion of the object into the list or library. Since the 2D display of an object requires the extraction of the objects data, it is not necessary to extract the data again. The UI-CSC will interface with the operator to select and modify a 2D object. After that modification is completed, the associated 3D object data will be modified to be consistent with the new 2D data. Then the new data for the object can be sent to the OM-CSC for insertion into the 3D object.

4.3.4.8 Associate (add or modify) Text With Geometric Object

This function allows for the association of text with the 3D polyhedral objects that currently exist in the geometric object list. This text consists of an object name and description, the operator name who created the object, the date and time the object was first created, the category of the object, the color of the object, and a list of vertices that describe the object. The OM-CSC is responsible for inserting and extracting the text data into/from the geometric object. The UI-CSC will perform all editing and display of the text data.

4.3.4.9 Output Text Report Of Geometric Objects

This function allows for the browsing and optional printing of any or all 3D polyhedral objects that currently exist in the geometric object list/library. The output consists of an object name and description, the operator name who created the object, the date and time the object was first created, the category of the object, the color of the object, and a list of vertices that describe the object. The output is formatted and written to an ASCII disk file. The OM-CSC simply creates the output file. The UI-CSC will handle the display and printing of the file.

Appendix A: Revised Requirements

The requirements for the ICERVS were developed from the set of mission profiles set forth in the System Design Report, issued in Dec. 21, 1992. In determining the subsystem design, these requirements were studied and a set of derived requirements was determined. This study led to greater insights into the requirements and identified instances where

- the wording needed to be clarified,
- the allocation of requirements to CSCs required adjustment, and
- the set of requirements having design influence in Phase 1 needed to be altered.

Each of these revisions is described below.

1. Clarification of Wording

The following requirements have been revised as follows. The changes are indicated in italics.

Delete

R1.01 **octree representation, spatial data**--Spatial data will be represented in octree format. Each volume element will represent a state of empty, partial, full, or unknown.

Insert

R1.01 **octree representation, spatial data**--Spatial data will be represented in octree format *and will be provided in rectangular coordinates*. Each volume element will represent a state of empty, partial, full, or unknown.

Delete

R1.02 **octree representation, property data**--Each volume element within the octree representation will optionally include at least 16 bits of property information.

Insert

R1.02 **octree representation, property data**--Each volume element within the octree representation will optionally include at least 16 bits *to represent* property information.

Delete

R1.05 **geometric models, polyhedral objects**--The system will build polyhedral objects to model real objects in the work volume.

Insert

R1.05 **geometric models, polyhedral objects**--The system will *provide a representation of* polyhedral objects to model real objects in the work volume.

Delete

R2.01 **create/modify/*display* primitives/templates**--The system will support primitives and templates for object modeling. The operator will be able to create, modify and *display* geometric templates.

Insert

R2.01 **create/modify primitives/templates**--The system will support primitives and templates for object modeling. The operator will be able to create *and modify* geometric templates.

Delete

R2.05 **create/modify/display 2D polygons (sequence of vertices)**--The system will allow the operator to create, modify, delete, and display 2D polygons. It is assumed that convex polygons will be sufficient to support all mission profiles.

Insert

R2.05 ***define*/modify/display 2D polygons (sequence of vertices)**--The system will allow the operator to *define*, modify, *erase*, and display 2D polygons. It is assumed that convex polygons will be sufficient to support all mission profiles.

Delete

R3.01 **translate and scale**--The graphical display will be able to scale the display of the volume in the window, from the whole volume to a single volume element. The graphical display will be able to display any portion of the volume in the window.

Insert

R3.01 **translate and scale**--The *ICERVS* will *allow the user* to scale the display of the volume in the window, from the whole volume to a single volume element. The *ICERVS* will *allow the user* to display any portion of the volume in the window.

Delete

R3.03 **pair of parallel cut planes**--The graphical display will support a single pair of parallel cut planes to be used in defining the region of the volume to be viewed.

Insert

R3.03 **pair of parallel cut planes**--The graphical display will support *at least a single pair* of parallel cut planes to be used in defining the region of the volume to be viewed.

Delete

R7.01 **input: x,y,z position**--The input data to the octree will contain full 3D coordinates (X, Y, Z).

Insert

R7.01 **input: x,y,z position**--The input data *provided by the mapping system or its equivalent* will contain full 3D coordinates such as rectangular coordinates (X, Y, Z) or polar coordinates (R, j, q).

For requirements R7.02 to R7.05, the phrase

"input data to the octree"

should be replaced with

" "input data provided by the mapping system or its equivalent".

2. Allocation to CSCs

The more detailed understanding of the User Interface CSC obtained during the study indicated the need for the following re-allocation of requirements.

User Interface CSC

Add: R2.05, R3.05, R3.06, R6.02, R7.01

Octree Engine CSC

Add: R6.02, R7.01

Delete: R3.01, R3.03, R3.11

It should be noted (Table 3-2) that the three deleted requirements had already been allocated to the User Interface CSC, so that none of the Phase 1 requirements have been omitted.

3. Design Influence

Greater insight into the nature of the CSCs prompted the following changes in the set of requirements having design influence during Phase 1.

Requirements added to the set include:
R2.01, R2.03, R3.10, R5.03.

Requirements deleted from the set include:
R7.02,R7.03, R7.04, R7.05,
R9.01, R9.11

All the deleted requirement affect CSCs to be developed in later Phases of the project.

DATE
FILMED
5/2/94
END