

# **Enroute Monitor and Control Proof of Concept Initiative Report**

October 11, 2002

Status: Final

---

# Preface

## TEMPLATE CHANGE LOG

<b>Submission</b>	<b>Date</b>
Preliminary	July 22, 2002
Draft	September 6, 2002
Final	October 11, 2002

---

## Table of Contents

<b>1</b>	<b>ENGINEERING TEAM.....</b>	<b>1</b>
<b>2</b>	<b>PURPOSE .....</b>	<b>2</b>
<b>3</b>	<b>DEFINITION OF TERMS .....</b>	<b>3</b>
<b>4</b>	<b>BACKGROUND.....</b>	<b>4</b>
4.1	WHY AN EN ROUTE MONITOR AND CONTROL PROOF OF CONCEPT? .....	4
4.2	DISPLAY SYSTEM REPLACEMENT .....	4
4.3	DAS/RSD .....	5
4.3.1	<i>DAS/RSD Design.....</i>	<i>5</i>
4.3.1.1	DAS/RSD User Interface.....	5
4.3.2	<i>DAS/RSD Architectural Issues .....</i>	<i>6</i>
4.4	HOCSR PHASE 3 AND HID/NAS LAN M&C .....	6
4.4.1	<i>HOCSR Phase 3 M&amp;C.....</i>	<i>7</i>
4.4.2	<i>HID/NAS LAN NSM.....</i>	<i>7</i>
<b>5</b>	<b>APPROACH .....</b>	<b>8</b>
5.1	POC HIGH LEVEL DESIGN .....	8
5.1.1	<i>NV User Application - The Object Manager .....</i>	<i>10</i>
5.1.1.1	NV User Application Object Manager Adaptation.....	10
5.1.2	<i>Subsystem Agents.....</i>	<i>10</i>
5.1.3	<i>End User Clients.....</i>	<i>10</i>
5.1.3.1	Object Adaptation.....	11
5.2	EM&C POC TEST BED .....	11
5.2.1	<i>Test Bed Environment.....</i>	<i>11</i>
5.2.2	<i>NV User Application Object Manager Design Details.....</i>	<i>12</i>
5.2.2.1	Tivoli Netview Network Node Manager.....	13
5.2.2.1.1	Netview Application Programmers Interface.....	13
5.2.2.1.2	Netview Object Database.....	13
5.2.2.1.2.1	Use of Netview Fields within the NV User Application .....	14
5.2.2.1.2.2	Status Propagation.....	15
5.2.2.1.3	Netview SNMP Message Processing.....	15
5.2.2.2	NV User Application End User Client Service.....	15
5.2.3	<i>Sherrill/Lubinski Java End User Client.....</i>	<i>15</i>
5.2.3.1	Java.....	16
5.2.3.1.1	Java Background.....	16
5.2.3.1.2	Java Architecture Advantages.....	17
5.2.3.2	Sherril/Lubinski Graphical User Interface (GUI) Builder Software .....	17
5.2.3.2.1	GMS Draw Graphical Editor .....	17
5.2.3.2.2	Object Dynamic Properties and Object Name and Object Userdata .....	19
5.2.3.2.3	An Adaptation Approach Using SL Map Data .....	21
5.2.3.2.4	Process Oriented Vs. Network Oriented Approach for GUI.....	22
5.2.3.3	SL End User Client GUI.....	23
5.2.3.3.1	SL Client Synchronization.....	24
5.2.3.3.2	Map/Submap Navigation .....	24
5.2.3.3.3	Event List.....	27
5.2.3.3.4	Main Menu Bar Functionality .....	27
5.2.3.3.4.1	System Menu Functions .....	27
5.2.3.3.4.2	Control Menu Functions .....	28
5.2.3.3.4.3	Event List Menu Functions .....	29

5.2.3.3.4.4	View Menu Functions .....	30
5.2.4	<i>External MySQL Relational Database Updater Application</i> .....	31
5.2.4.1	Event History Viewer .....	32
5.2.5	<i>Subsystem Agents</i> .....	32
5.2.5.1	Agent Simulation Tool .....	33
5.2.5.2	DAS Replacement .....	35
<b>6</b>	<b>SUMMARY</b> .....	<b>36</b>
<b>7</b>	<b>PROPOSED REQUIREMENTS</b> .....	<b>36</b>
	<b>EXAMPLE FIELD AND NV USER APPLICATION FIELD DEFINITIONS</b> .....	<b>37</b>
	<b>EXAMPLE OF THE NV USER APPLICATION ADAPTATION DATA FILE</b> .....	<b>39</b>
	<b>AN EXAMPLE OF A DECODED SHERILL/LUBINSKI MAP FILE</b> .....	<b>41</b>

---

# 1 Engineering Team

Below is the list of engineering team members:

## POC Development Team

Stephen Souder	609-485-6170	ACB-230 / EIIF
Bill Pfeiffer	609-485-7906	Enroute Computer Solutions/ EIIF
Michael Perseo	609-485-7915	Joseph Sheairs Associates/ EIIF
Tom MacWright	609-485-6170	Joseph Sheairs Associates/ EIIF
Dave Ingegneri	609-485-6170	ACB-230 / EIIF
John Gauntt	609-485-8075	J&N/EIIF

---

## 2 Reference Documents

---

### 3 Purpose

The purpose of this report is to document the Integrated Enroute Monitor and Control (IEM&C) Proof of Concept (POC) effort conducted at the En Route Integration and Interoperability Facility (EIIIF). This work was performed for the FAA Enroute Program Office (AUA-200).

To initiate the POC, a viable development approach was selected based on standard industry Network Management concepts using open standards. Using a Commercial Off The Shelf (COTS) Graphical Development Tool, Java based technology, Simple Network Management Protocol (SNMP) communication and a COTS Application Programmer's Interface (API) to a COTS Network Node Manager, a core application was developed that can be used as a model for future development efforts and can help to define and refine FAA requirements for a modern Integrated En Route M&C capability.

This document discusses the technologies that were considered and selected for the POC including development platforms and COTS applications. It also discusses the POC design and how it could be extensible to a field-capable En Route M&C system.

---

## 4 Definition of Terms

The following list is a definition of terms provided for use in the context of this report

- **Graphical User Interface** – The common practice of providing a graphic (i.e. visual object) to a user for interaction with a software application.
- **Client / Server Model** – An industry standard system design that allows multiple user applications (clients) to connect to a common service application (a server) to acquire common data types.
- **Integrated Development Environment** – A software development tool that provides a source code editor, source compiler, and additional development tools within a common application environment.
- **Graphical Editor** – An editor application that allows the user to modify software objects graphically (a visual of the object).
- **Relational Database** – A table based data storage model that provides quick and efficient storage and access capability of various data types.
- **Structure Query Language** - An industry standard language for querying, storing, or modifying specific data within the context of a relational database.
- **Application Programmer Interface** – A well-defined method, typically provided with an application, for communicating or exercising functions within the application. The interface usually comes in to form of software routines that may be linked to by external applications so that the routines may be used by the application.
- **Berkeley UNIX Socket Based Communication** – A method of digital communication originally developed at the University of California Berkeley that utilizes Internet Protocol (IP) to establish a data socket between two software processes. This socket can then be used to share data between the processes.

---

## **5 Background**

### ***5.1 Why an En Route Monitor and Control Proof of Concept?***

The E&MC POC at the EIIF was initiated to address the many issues surrounding En Route System Monitor and Control operations. These issues pertain to the current Area Manager Command and Control (AMCC) systems as well as new systems being deployed to the field, specifically the Host and Oceanic Computer System Replacement Phase 3 (HOCSR P3) Monitor and Control System. These issues really boil down to two items: lack of M&C integration within each En Route Subsystem and inflexibility in the end-user presentation.

The current AMCC M&C model is one of disparity. The current operation contains up to 45 M&C workstations all displaying status and offering control features for the various subsystems located within the En Route facility. In many cases these systems have similar data points display. The primary reason for this situation is the lack of a common integrated platform for hosting an integrated En Route M&C. In addition, each subsystem within the En Route domain proposes its own complex architecture and method for M&C. Without a requirement for new Subsystems to interface to a common well-defined M&C platform, it is not surprising that M&C capabilities are disparate from subsystem to subsystem.

Complicating this situation is the fact that many of the systems being monitored and controlled are legacy systems which were designed and implemented before any common network/system management concepts had become accepted as standard.

Many of the current M&C platforms located at the AMCC are also highly customized regarding the User Interface. This precedence has made it difficult to implement more modern M&C applications that provide flexibility to customize the user interface. This problem has become evident in such programs as HOCSR Phase 3 where a modern management application was chosen for M&C, but the integrators struggled to provide a user interface that was acceptable to the En Route Airways Facilities personnel.

### ***5.2 Display System Replacement***

A very robust and somewhat modern Monitor and Control subsystem was fielded with the Display System Replacement (DSR) subsystem. This M&C was considered at the onset of this POC. The DSR subsystem, however, was developed using proprietary software without the use of accepted open standards and no common interface is provided for external applications (i.e. an API). It was decided that even though the DSR M&C is robust, it would not be acceptable for an integrated M&C.

## **5.3 DAS/RSD**

One of the currently fielded systems in the AMCC area that did try to address the issue of M&C system disparity was Data Acquisition System/Remote System Display (DAS/RSD) M&C subsystem.

The DAS/RSD system was an AOS initiative that brought together M&C for various subsystems within the En Route domain into one M&C environment.

### **5.3.1 DAS/RSD Design**

The current DAS/RSD is a standalone Monitor and Control system that consists of, a proprietary operating system, INTEL processor boards, custom C language software, optically isolated digital sensor connections and serial communications to the maintenance and status displays. An RS-422 serial connection is made to one of the devices sensors for status information regarding the back-up display channel (EDARC).

#### **5.3.1.1 DAS/RSD User Interface**

One interesting aspect of the DAS/RSD design is its highly customized User Interface. The RSD maps look more like functional system diagrams rather than the network oriented diagrams commonly found in COTS M&C products such as Tivoli Netview Node Manager. An example of an RSD screen is depicted in Figure 5.1. This approach seems to be more acceptable to airways personnel responsible for monitoring of the items depicted.

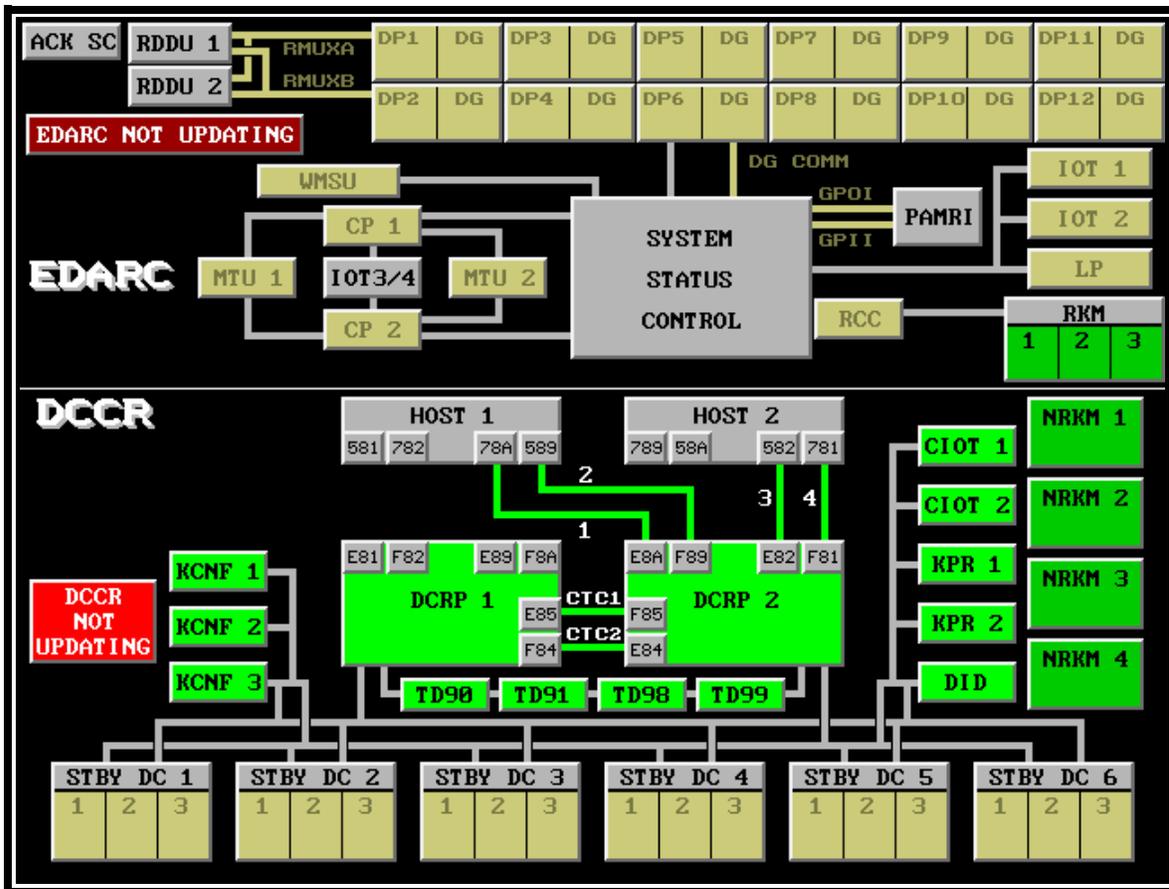


Figure 5.1 Example RSD Screen

### 5.3.2 DAS/RSD Architectural Issues

Although the DAS/RSD system design is interesting in that it integrates various En Route subsystems, the architecture, particularly the RSD, is archaic. This design of proprietary operating system and old INTEL processor boards would not be extensible to any modern M&C platform.

Although these functional maps used in the DAS/RSD User Interface are more in line with user needs, they have been rather difficult to maintain. The maps are maintained in C++ motif and each map change needed, no matter how trivial requires a code change. This makes DAS/RSD system maintenance very cumbersome.

### 5.4 HOCSR Phase 3 and HID/NAS LAN M&C

In recent years, open system network node managers have been fielded for use with different projects. Two such managers are the HOCSR Phase 3 Monitor and Control and the Host Interface Device (HID)/ National Airspace System (NAS) Local Area Network

(LAN) Network System Monitor (NSM) which are both essentially Tivoli Netview Network Node Managers.

#### **5.4.1 HOCSR Phase 3 M&C**

The HOCSR Phase III M&C subsystem could potentially represent the cornerstone of the future En Route M&C architecture. It utilizes Ethernet based architecture and the Internet Protocol, ICMP and Simple Network Management Protocol (SNMP). System components that are to be monitored are identified as objects in the Tivoli Netview Network Manager application of this subsystem. Status information for these objects is derived or communicated via these protocols.

Many aspects of the HOCSR Phase 3 design are used in the EM&C POC, particularly the use of the Netview application and the use of an underlying IP infrastructure. However, there is a particular item that was omitted in the POC design; the use of Netview's User Interface called OVW. OVW uses a method of symbols or icons that correspond to objects in the Netview database. These symbols reside on network maps that the Netview application creates. Symbol status is determined based on SNMP information that Netview obtains via agent polling or agent traps. OVW is customizable, but for the most part is rather constricting when it comes to symbol and map representation.

#### **5.4.2 HID/NAS LAN NSM**

The Host Interface Device (HID)/NAS LAN (HNL) Network System Monitor (NSM) is an International Business Machines (IBM) Reduced Instruction Set Computer (RISC) System/6000 processor running the IBM Advanced Interactive Executive (AIX) Version 4.1 operating system. The NSM contains the network management software used to monitor and manage the hardware and software applications resident on the HNL. The NSM computer system is also used to access other applications resident on the HNL in order to perform functions such as adaptation changes.

The NSM software is based on Netview for AIX. All hardware and software management executed by the NSM uses this software. Similar to the HOCSR Phase 3 M&C, the NSM also uses Netview's OVW as an end-user interface and thus the same GUI issues apply here as in the HOCSR Phase 3 system.

---

## 6 Approach

The primary goals of this Proof of Concept were to define an approach for, and to design and implement an extensible core application that could be used to demonstrate available newer GUI Builder technologies. Once in place the application could then be used to solicit feedback from Human Factors organizations within the FAA, and to help define a set of requirements for an Integrated Monitor and Control position. To ensure an accurate and comprehensive understanding of user interface requirements in this area, the engineering team at the I2F evaluated existing En Route M&C systems as described in previous sections, to determine if their design characteristics could be exploited using the new technology. For more detail on the on the constraints and assumptions used for this **Proof of Concept see XXXX.**

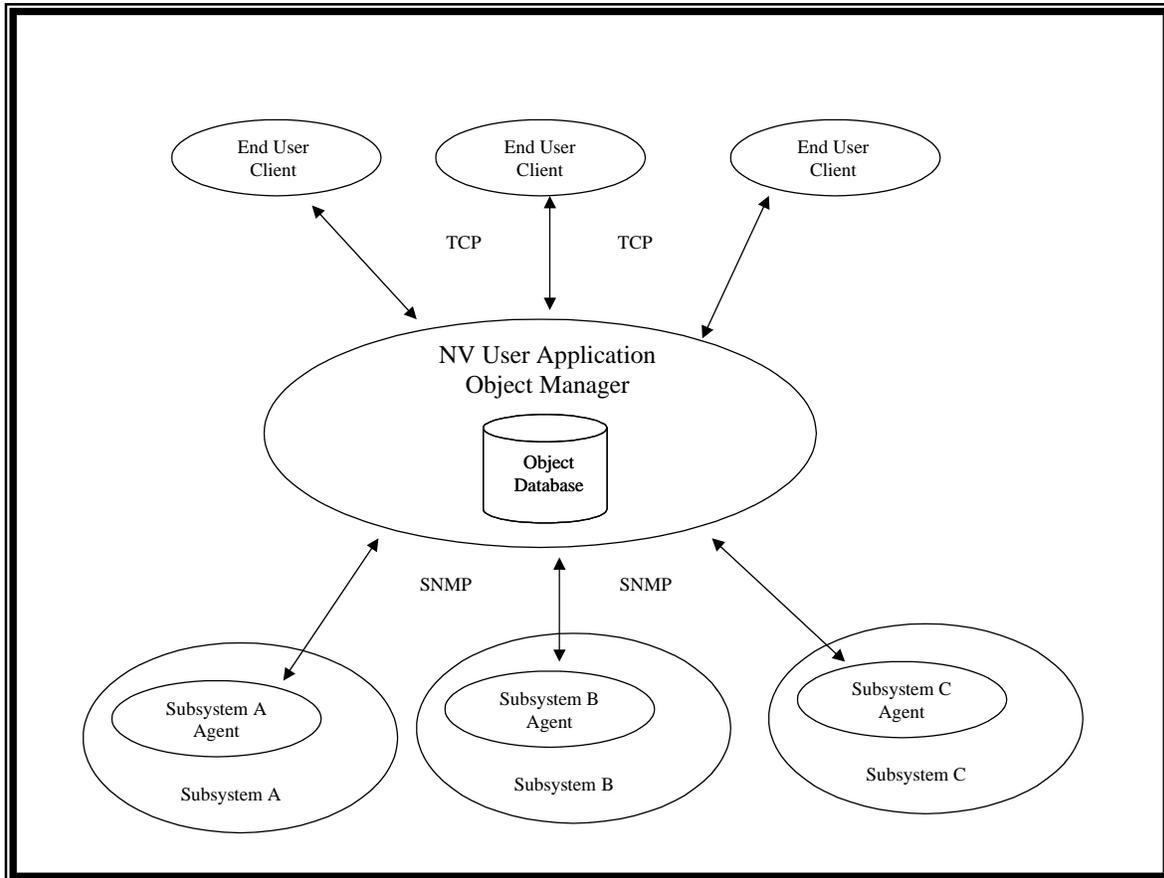
### 6.1 POC High Level Design

The EM&C POC design consists primarily of a client/server model that utilizes multiple open system standards for communication, namely Berkeley UNIX (BSD) Socket based TCP/IP communication and Universal Datagram Protocol (UDP) Simple Network Management Protocol (SNMP) messaging. Figure 5.1 depicts a high-level design diagram of the EM&C POC architecture. There are three basic elements within the design, an object manager, one or many subsystem agents providing status to the manager, and one or many end user clients that connect and receive object information from the manager.

The network environment for the EM&C POC, which is explained in more detail in subsequent sections, is borrowed directly from the HOCSR Phase 3 program (i.e. a switched IP network). Tivoli Netview Node Manager Application is also used in the EM&C. Netview was chosen as the Node Manager because of its use in the HOCSR Phase 3 and HID/NAS LAN programs.

During the user interface design selection, the advantage of having highly customizable user maps (as well as control features) like the DAS/RSD system was considered, however the complexity that this introduces in software maintenance was too great. An alternative method for creating and changing the user interface, specifically GUI Builder products, was evaluated and exploited.

The approach to designing the EM&C POC was to address apparent “trouble spots” within some fielded M&C products. An example of these problems is error propagation within the HOCSR Phase 3 (Netview) implementation. Low-level errors in the Netview product propagate up to higher-level map symbols in a fashion that is considered marginally acceptable to the end user. The POC design addresses this problem by implementing a customized propagation routine that could be designed to be adaptable (i.e. user changeable) in the future.



*Figure 5.2 EM&C POC High Level Design*

### **6.1.1 NV User Application - The Object Manager**

The manager portion of the software is a server application, which was given the name NV User Application. This software manages objects that get status from subsystem agent and provide that status to connected end user clients. The NV User Application uses the Tivoli Netview Node Manager API to take advantage of Netview Node Manager's object database as well as its SNMP data type processing capabilities. The server also implements a TCP/IP communication interface that will provide any connecting clients with object information.

#### ***6.1.1.1 NV User Application Object Manager Adaptation***

Because clients connect to the NV User Application to receive object data, an object synchronization method was developed so that a client can know what particular object information can be obtained from the server. This method came in the form of object adaptation (a text based file), which is read by the server software at start up. The adaptation provides the object name and information required to manage object status in the server.

### **6.1.2 Subsystem Agents**

Subsystem agents provide status related to a particular subsystem to the NV User Application object manager. All agents are designed to forward required status data via SNMP messages. Notably, the assumption that all Subsystem agents will have the capability to forward status data via SNMP to the object manager presents an interesting challenge for some of the legacy subsystems within the En Route infrastructure and Section 5.2.5.3 *DAS Replacement* discusses a potential approach for addressing this issue.

### **6.1.3 End User Clients**

End User Clients connect to the NV User Application object manager to receive object status data. The connection between the client and server is a TCP connection with a pre-defined messaging protocol. What the end user client application does with the received message is completely independent of the server. This design decouples the graphics generation software from the object status manager software, which permits the creation of alternative user interfaces that simply need to take advantage of the object status information, offered in a prescribed format. This notion of client-to-server functional independence has been explored in the EM&C POC using several client applications.

### **6.1.3.1 Object Adaptation**

As discussed in Section 5.1.1.1, the NV User Application uses adaptation data to create the object database. Object information obtained from an adaptation file is distributed to all end user clients that connect with the NV User Application in an initialization message (INIT). This method ensures that all end user clients receive the same initial information. Depending on the end user client's function it may or may not use all of the data provided by the server.

## **6.2 EM&C POC Test Bed**

Figure 5.2 depicts the EM&C Proof of Concept Test Bed. This test bed takes the POC design described in the previous section and implements it. Following the high level design, the test bed includes examples of three different types of elements; an Object manager, End user clients, and Subsystem agent.

### **6.2.1 Test Bed Environment**

The network environment for the EM&C POC is very similar to the HOCSR Phase 3 environment. Sun Blade platforms running Sun Solaris Version 8 are used for running the server and the clients. X Windows Desktop Manager (XDM) is used to display the end user client onto network thin clients. All network nodes are networked together using Cisco Layer 2 Network switches. Given the design, hardware platforms become less important provided that all elements are networked together using IP.

Tivoli Netview Node Manager Version 6.0 is installed on one of the Sun Solaris platforms. The Netview API is used to access the Netview object database. Additionally, Sun's Java Virtual Machine Version (JVM) 2.0 is installed on the Sun system to accommodate the Java end user client software.

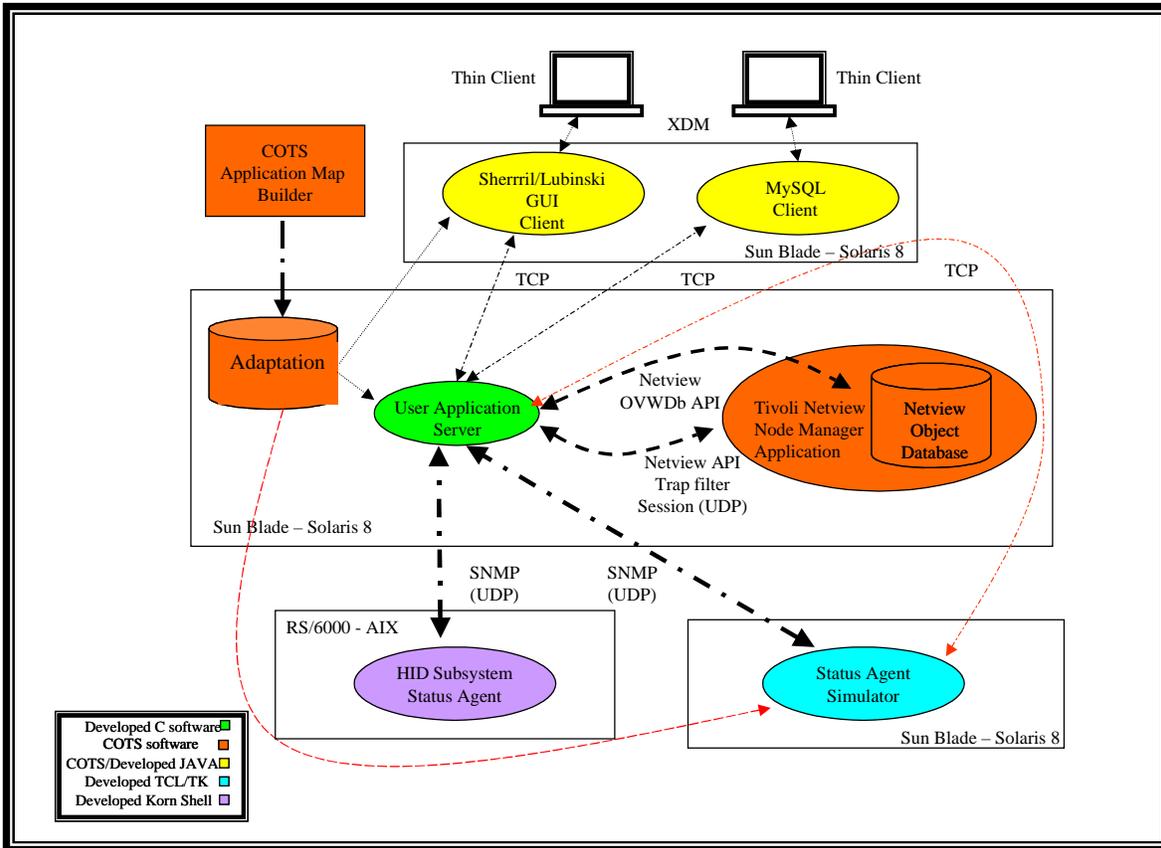


Figure 5.3 EM&C Proof of Concept Test Bed

## 6.2.2 NV User Application Object Manager Design Details

The developed NV User Application runs on a Sun Solaris Version 8 platform. It consists of three main functions:

- Creating and Maintaining Object Information – Objects are created and maintained in the Tivoli Netview Object Database.
- Listening for and processing SNMP status information pertaining to system objects – A UDP socket is established with the Netview Node Manager Trapd process using the Tivoli Netview API. This communication interface is polled for incoming SNMP data from subsystem agents.
- Listening for and accommodating End User Clients – A TCP server socket is established to listen for connecting clients and a separate process is started (i.e. forked) to communicate with clients once they are connected.

### **6.2.2.1 Tivoli Netview Network Node Manager**

As mentioned earlier, Tivoli's Netview Network Node Manager application is the centerpiece of the EM&C POC design. It provides two vital elements to the NV User Application software, The Object Database and SNMP message processing capabilities. Access to these elements is available via the Netview Application Programmer's Interface.

#### **6.2.2.1.1 Netview Application Programmers Interface**

Like most COTS Applications made for Enterprise environments, the Netview system includes an API as a part of the software package. This API exposes interfaces into the Netview application in many different areas. The EM&C POC design only exploits two of those areas.

- The SNMP API – This API allows applications to open “sessions” with the Netview SNMP trap process called Trapd.
- The End User Interface (EUI) API – This API provides visibility and control of objects within the Netview Object Database.

The Netview API comes in the form of pre-compiled C libraries that can be linked into application code. These C libraries allow visibility into the Netview Object Database and provide access to the Netview SNMP processing capabilities. Because the Netview API libraries are written in C, the NV User Application was written using the C programming language.

#### **6.2.2.1.2 Netview Object Database**

The Netview application uses an object data storage mechanism called the Netview Object Database. This database is a series of interlinked encoded files that contain specific data corresponding to each object that the Netview application has either detected on the network via it's auto discovery mechanism or has been directed to maintain via external sources (i.e. User Applications). Objects typically represent nodes (hardware) that reside on a network that the Netview application monitors, but they can also represent software applications, or other user defined elements.

Each object may be assigned characteristics called fields. These fields give the object form and meaning. Fields are typically assigned to objects based on responses to SNMP queries sent by Netview to nodes that have responded to the auto discovery mechanism. There is a base set of fields that comes with the Netview package. These fields are used by the Netview application to identify network nodes and represent them properly within the Netview infrastructure. The fields are defined using constructs similar to C data structures and reside in a pre-defined directory within the Netview application infrastructure. Netview fields can be one of four types.

- **Boolean** – A true/false value that indicate whether or not an object has a particular property. Examples of this type are:
  - IsNode – A field that indicates that an object is a network entity

- IsRouter - A field that indicates that an object is a network router
- **String** – A character string limited to 256 characters that indicates that objects value regarding that field. An examples of this type is:
  - SelectionName – A field that indicates a unique name for an object. There is a one to one mapping of SelectionName and ObjectId, which is an object’s unique number.
- **Enumeration** – An integer value that maps to predefined definitions. These definitions are listed in the field’s structure. An example of this type is:
  - IP Status- A field that enumerates the 9 different states that an object can be at any given time.
- **Integer32** - a 32-bit integer value. An example of integer is:
  - Tivoli Server Port – This field allows a programmer to set the Tivoli TCP/IP port.

Appendix A includes actual syntax for defining these examples.

#### **6.2.2.1.2.1 Use of Netview Fields within the NV User Application**

To effectively operate within the Netview object database, certain fields were added to Netview for use specifically with the NV User Application. These fields allow the server application to properly manage objects it is responsible for. The following fields are defined for the NV User Application:

- **IsNVUserApplication** – This Boolean field is set for objects that the NV User Application is responsible for.
- **IsParent** – This Boolean field is set for objects that are defined in map adaptation as a parent
- **LUID** – This String field gets set with a 4 byte DAS/RSD LUID value defined in map adaptation for each object.
- **Parent** – This String field gets set with a valid parent object name for each object that the NV User Application is responsible for.
- **IsAcknowledged** - This Boolean field is set for objects that do not have a state change that is unacknowledged at the user interface.

In addition to the above fields, the NV User Application uses the Selection Name field to identify the unique name for each object and the IP Status field to store the object current state.

Appendix A includes actual syntax for defining the fields defined used in the NV User Application.

#### **6.2.2.1.2.2 Status Propagation**

One of the main issues identified with the HOCSR Phase 3 M&C system is the status propagation conventions used. These are rules for propagating an object status change to a higher-level sub-map within the Netview User Interface. The rules used in the HOCSR Phase 3 implementation are defined within the Netview application (i.e. standard Netview propagation). These standard Netview propagation rules do not allow for any flexibility in terms of propagation from child object to parent object. For example, object A and object B on a submap propagate their node status upward to a compound propagated parent object on the same submap or parent map, considering the status value of each child object equally. If there is any object whose status should be propagated to the parent differently, there is no mechanism to do so within the Netview product. Additionally, if two or more objects on a submap have a relationship such that their collective status dictates that the parent object's status should reflect some type of "group" status, there is no method for asking Netview to force this group status to the parent.

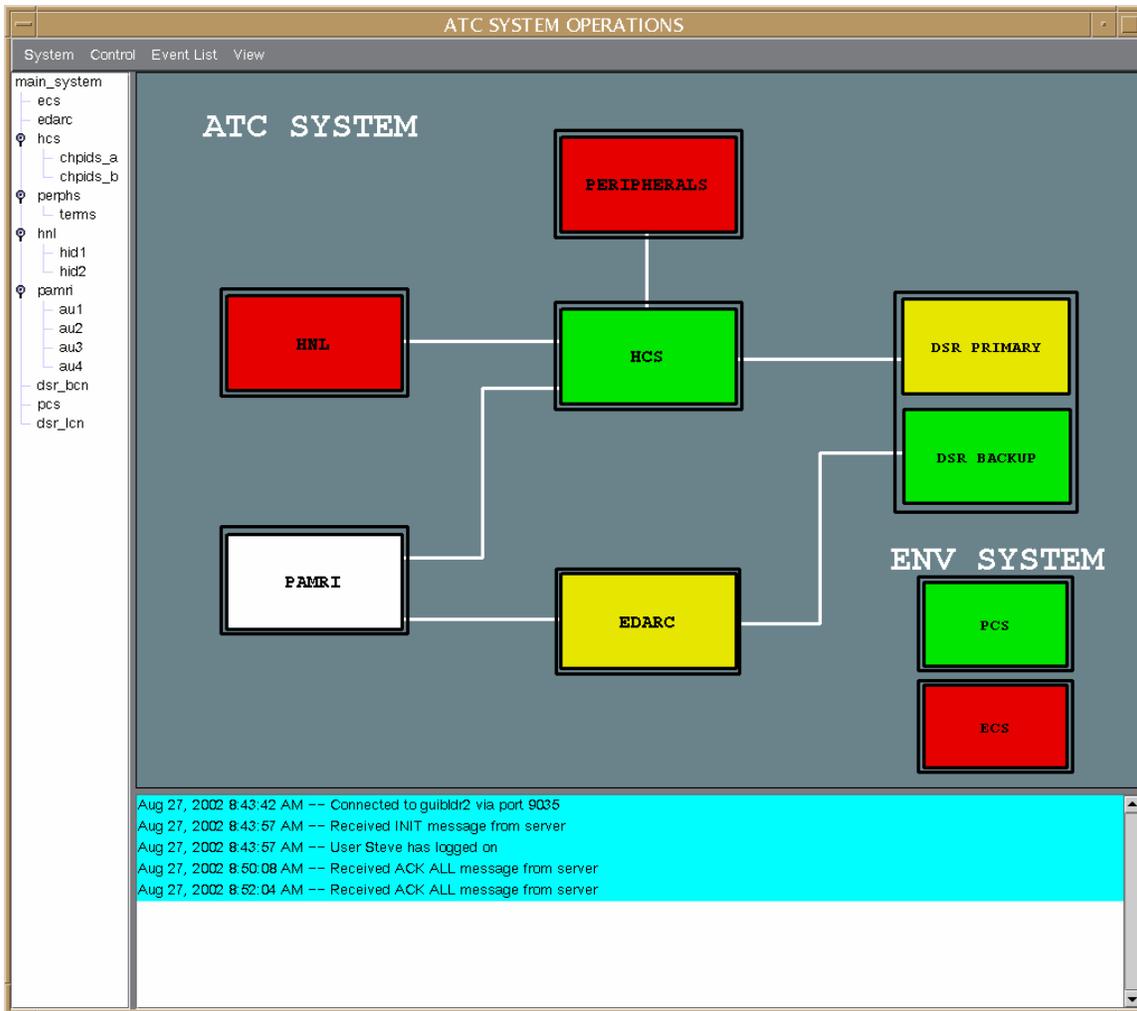
Status propagation issues and approaches are still being reviewed and developed. Technically, any status propagation methods can be implemented. The end user community needs to determine what the requirements are in this possibly complex area.

#### **6.2.2.1.3 Netview SNMP Message Processing**

#### **6.2.2.2 NV User Application End User Client Service**

#### **6.2.3 Sherrill/Lubinski Java End User Client**

Figure 5.3 depicts the main End User Interface for the EM&C POC. This interface was developed using a COTS Graphical User Interface (GUI) builder software package created by a Sherril/Lubinski, as well as Java technology. These development tools and the client application functionality are described in this section. . For more detail on the on how to add and modify objects used for this **Proof of Concept see YYYY.**



**Figure 5.4** The SL/Java End User Client

### 6.2.3.1 Java

The Java programming language has become one of the premier languages for modern networked system applications. There are many reasons for this. Some of these are pertinent to the FAA En Route environment and some are not. This section will discuss those that are.

#### 6.2.3.1.1 Java Background

The Java language started as an in house development tool at Sun Microsystems in the early 1990's. The company soon realized the tool's potential as a full-fledged programming language. Sun released Alpha version 1.0a2 in 1995 in a free distribution available over the Internet. Since that time Java has gone through 3 major revisions and has for the most part taken the network computing community by storm.

#### **6.2.3.1.2 Java Architecture Advantages**

Java technology is designed so that software developed in Java can be executed on many types of computing devices. This is accomplished using a software component call the Java Virtual Machine (JVM). When Java software source is compiled the resulting executable class is a basically a set of encode instructions called byte-code. This byte-code has nothing to do which any particular computer architecture. The JVM takes Java byte-code instructions and tailors them for the specific platform they are to be executed on. The JVM is what differs from platform to platform, not the application code. This greatly reduces issues with software portability between computer platforms, which has been a big issue in the FAA En Route domain in the past.

The java language is also completely object oriented using a model much like that of C++. Object Oriented Design (OOD) and Object Oriented Programming (OOP) techniques have proven their worth over the last 25 years and at this point it would be a mistake not use an object oriented approach to application development if one is available.

#### **6.2.3.2 Sherril/Lubinski Graphical User Interface (GUI) Builder Software**

Over the last ten years, software development vendors have begun to create Integrated Development Environments (IDE) for building application graphical interfaces. These IDEs come with packaged libraries that allow the developer to graphically create a user interface with minimal code. The code used to create the user interface objects is pulled from the packaged libraries. SL extends Basic Java Foundation (JFC) classes to create their own set of classes that may be used as is, or extended again to satisfy more specific needs. A combination of JFC, SL, and custom classes was used during the development of the EM&C POC. This approach facilitated a rapid prototype development methodology.

The following sections describe the features of various SL IDE components that were explored during the POC.

##### **6.2.3.2.1 GMS Draw Graphical Editor**

The SL-GMSDraw graphical editor is a graphical development tool available from the Sherill/Lubinski Corporation. The tool can be used to create interactive graphical screens with colorful dynamic and even animated components that represent “real” system objects. The tool allows the user to expedite the creation and editing of graphical models. When you create a new Model, you can draw any screen object, attach dynamic behavior to the object, and preview the behavior without leaving the graphical editor and without writing a single line of code. After you have verified and tested the Model within SL-GMSDraw, you can then easily integrate it into a user-developed application, either as the primary interface screen or as a Sub-Model. Models get saved to disk as encoded files with a file name with an “. m1” suffix.

There are two different environments for which SL-GMSDraw is available; Unix/Motif and 32-bit Microsoft Windows. The vendor indicates that the product is functionally identically in both environments.

SL-GMSDraw provides a comprehensive and powerful set of tools that allow you to control the real-time updating of screen objects in response to “real” system object status changes. Dynamics specification is accomplished entirely within the SL-GMSDraw editor through the SL-GMS dynamics scripting language. You may either edit the dynamics scripting language directly or use a GUI-driven interface. As a developer, you may be required to write minimal code in order to translate the user application variables into a form recognized by the dynamic translator engine, and thereby allow the mapping between the variables defined in the Models and the application variables. Fundamental dynamic behaviors, such as changes to object fill-color, object edge-color or style, and the text contained within an object, are only a subset of the complete offerings of the SL GMSDraw toolset and are accessible directly through the editor. You may need to write code when developing Models using more sophisticated dynamic behavior.

The SL-GMSDraw editor provides:

- Tool panels to create and change objects
- Menus organized according to functionality
- Easy-to-use dialog windows
- Printing capabilities
- Import capabilities for foreign file formats, including ".xwd" (Motif) and ".bmp" (32-bit Windows) bitmaps
- Zoom and pan capabilities
- "Drag-and-drop" drawing operations for moving, scaling, rotating, and duplicating objects, and point handling
- Capability to select a list of objects and modify a graphical property on the entire list with a single key-click
- Capability to select a list of objects and move forward and backward through the list during editing operations
- Multiple "undo" and "redo" operations with the type of each operation identified
- Line-by-line error checking of dynamics scripts
- 32-bit Windows- and Motif-compliant copy, cut, and paste operations via a clipboard

Figure 5.5 is a depiction of the GMS Draw Graphical Editor.

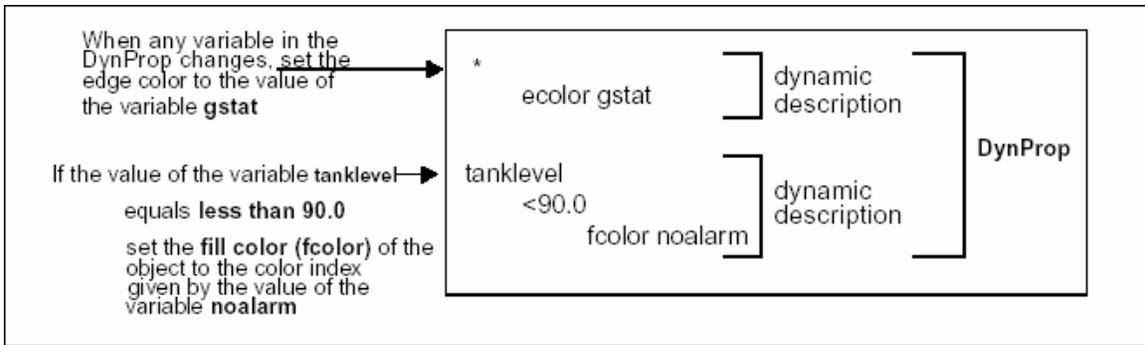


Figure 5.5 The SL GMS-Draw Model Editor

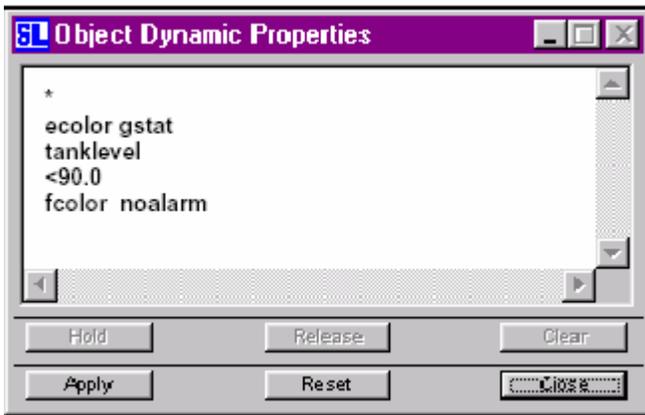
### 6.2.3.2.2 Object Dynamic Properties and Object Name and Object Userdata

In SL-GMSDraw, dynamic behavior is added to an object by attaching a Dyna-Prop (Dynamic Property). A DynaProp consists of one or more dynamic descriptions. The dynamic descriptions are written in a language recognized by SL-GMS that specifies a change in the appearance of an object in response to a change in an application variable, or an action to be taken in response to input events for the object. Figure 5.6 depicts the dynamic property syntax and Figure 5.7 illustrates the Object Dynamic Properties view.

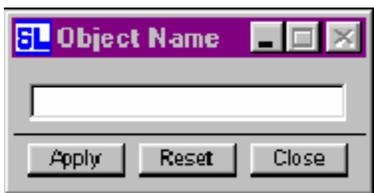
SL-GMSDraw allows a user to attach an Object Name to objects created in the editor. Attaching a name to an object allows objects to be selected by name (e.g., for selection of invisible or undetectable objects) and more importantly allows objects to be found by an application program. Names may be any collection of alphanumeric characters and must begin with a letter. There are SL-GMS reserved words that cannot be used as names. Figure 5.8 depicts the Object Name view where an Object Name can be entered.



*Figure 5.6 Dynamic Properties Example*

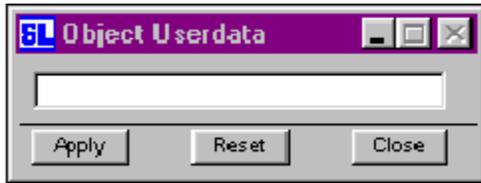


*Figure 5.7 The SL GMSDraw Object Dynamic Properties View*



*Figure 5.8 The SL GMSDraw Object Name View*

SL-GMSDraw also allows a user to attach Userdata to objects created in the editor. UserData is an arbitrary string used by an application program in any way desired. UserData must be fewer than 256 characters in length. Figure 5.9 depicts the Object Userdata view where the user data that is presented to the application can be entered.



---

*Figure 5.9 The SL GMSDraw Object Userdata View*

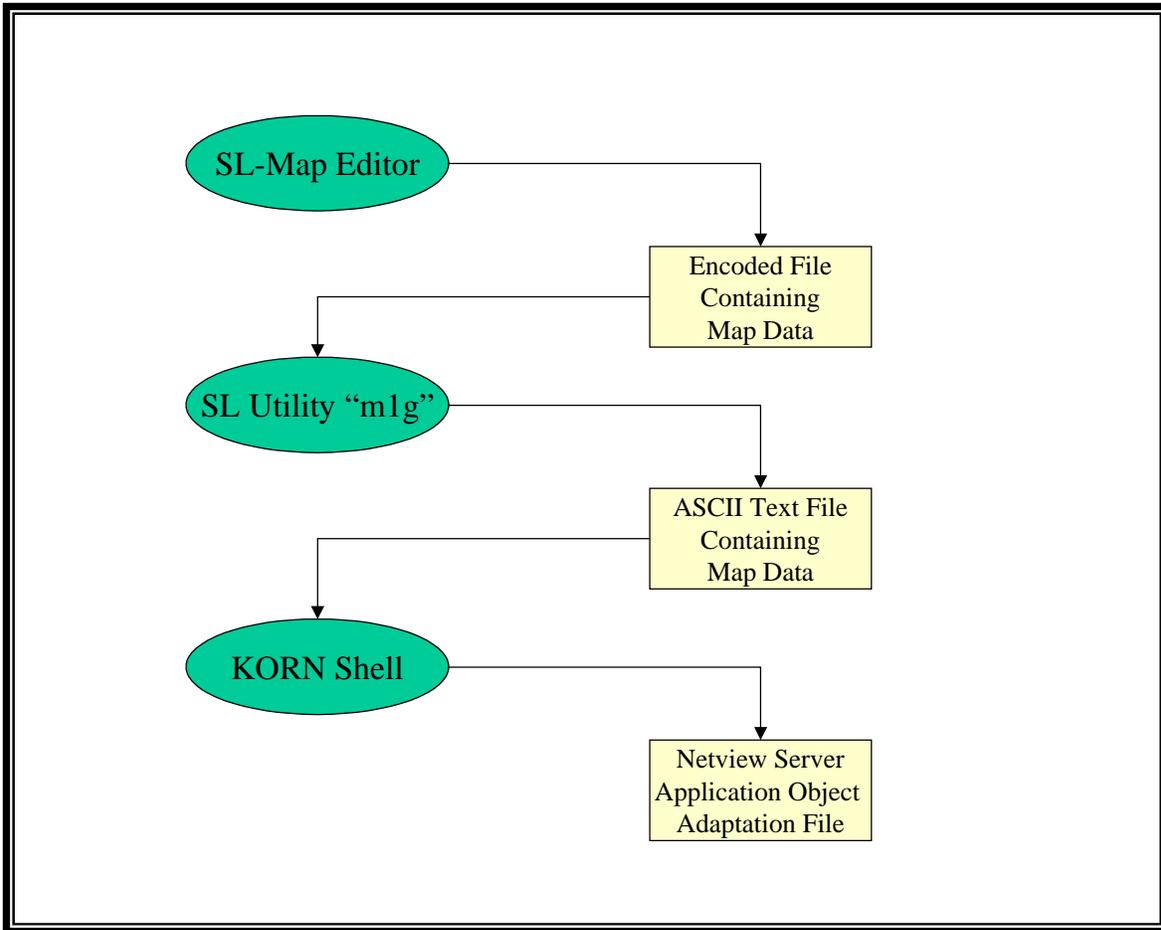
---

### **6.2.3.2.3 An Adaptation Approach Using SL Map Data**

SL-GMSDraw output data is stored in encoded binary format. SL provides utilities to convert this encoded binary format into readable ASCII text data. This text data can then be parsed using automation to create adaptation products specific to map objects.

This approach was used to create the adaptation used by the NV User Application Server. The SL utility is used to un-encode the “.m1” map files into text-based data files and UNIX shell-scripts are used to extract map object information from the text files. See the process diagram in Figure 5.10 below for clarification. Appendix C also includes an example of a decoded SL GMS Draw Map file.

The NV User Application server uses this adaptation to create Netview Object Database entries. It also uses the adaptation to define object characteristics such as parent/child object relationships, to specify identification data related to DAS objects, and to create status propagation characteristics for each object. The Agent Simulation Tool also uses the same adaptation to build its GUI dynamically. The benefit of designing the system with object adaptation that is built using the GMS Graphical Editor is that changes to the adaptation can also be maintained in the editor. No new code is required. Appendix B provides an example of the NV User Application adaptation data file.



*Figure 5.10 Netview Server Adaptation Generator Process Diagram*

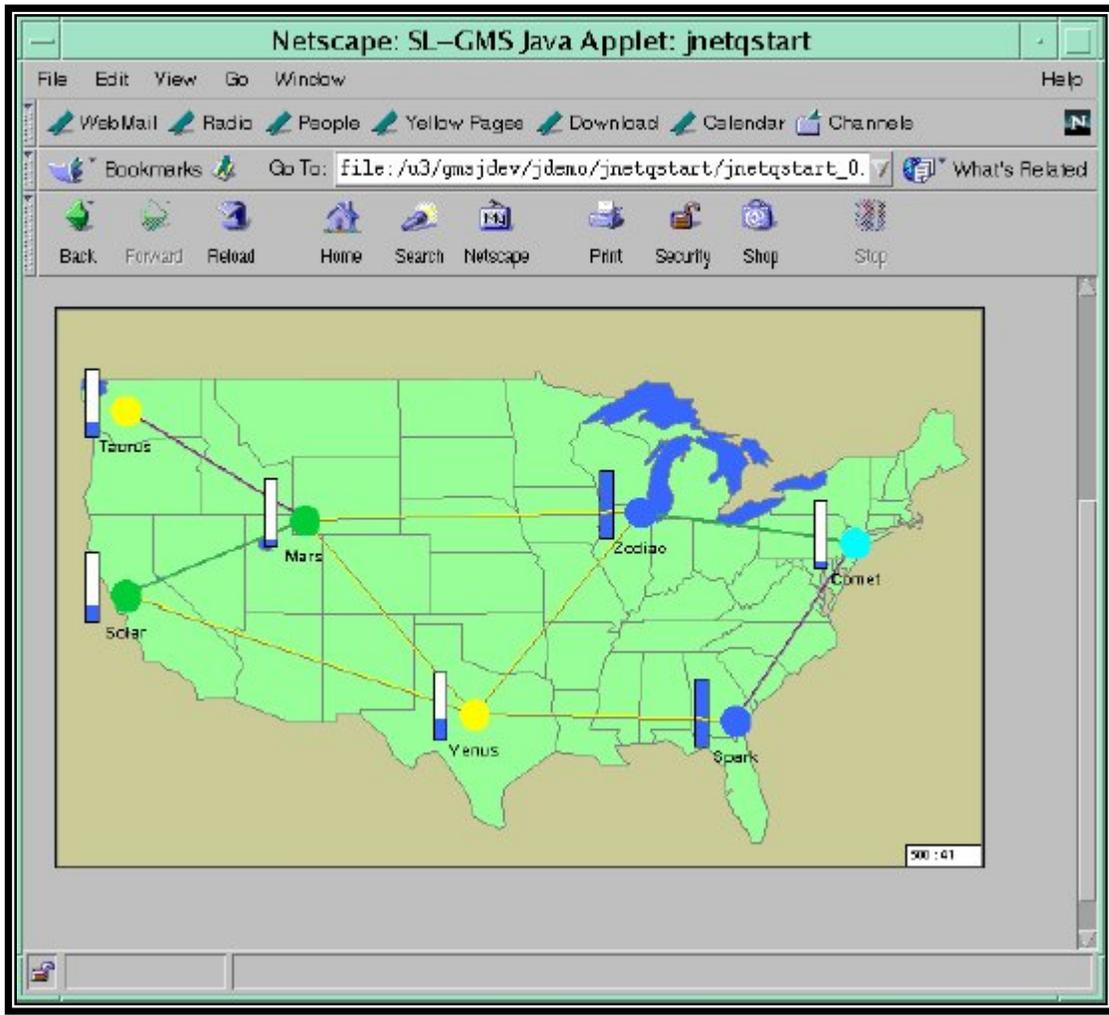
#### 6.2.3.2.4 Process Oriented Vs. Network Oriented Approach for GUI

SL provides two different mechanisms to build a GUI. The Process-Oriented scheme utilizes the GMS editor to draw static maps as mentioned in previous sections. The other scheme uses different java classes to draw maps dynamically that can then be saved.

The typical approach for this Dynamic or Network-Oriented scheme creates a set of objects with the editor (as opposed to creating actual process or schematic maps). These objects are then placed dynamically across a static backdrop display. A common static backdrop in the SL examples is a map of the United States. Various nodes can then be placed on the map dynamically using application software. The advantage to the Network-Oriented scheme is that large map changes can be affected quickly without recompiling software. This is advantageous in a continually evolving network. The drawback is that more application software is needed to wrap the GUI front-end into an end-state product.

The EI2F POC team decided to use the Network-Oriented scheme early in the development process to get the basic map layouts. We later decided to forego this method

since our map layouts would not significantly change and more code would need to be written to support the many different types of objects. However, the Network-Oriented scheme may be a good map tool for rapid prototyping of new en-route systems early in their design phase. Figure 5.11 is an SL example of a Network-Oriented Scheme application.



*Figure 5.11 SL example of a Network-Oriented scheme application*

### **6.2.3.3 SL End User Client GUI**

The SL End User Client GUI currently consists of four main components. They are the Main Menu Bar, the Navigation Panel, the Graphical Display Panel, and the Event List Display Panel. With the exception of the Graphical Display Panel, all components use only the base Java Foundation Class (JFC) classes in their implementation. The

Graphical Display Panel uses extensions of SL-provided classes and their methods to expedite the creation and modification of the schematic-like presentation of system objects. A Java wrapper using JFC and SL extensions is used to manipulate the presentation of the Graphical Display Panel information in a way that, after many internal I2F technical meetings, has been decided to be the most appropriate method for conveying object changes.

Deciding upon the most appropriate method for conveying object status change information is not a simple or straightforward process. There are many subtleties and nuances to consider as in any GUI design effort. The goal of this effort was to include conveyance methods that are used successfully in current FAA systems and non-FAA systems, to include those that most PC users are now comfortable with, and discard those that are overly complex such that they would be cumbersome to modify and maintain. Leading edge graphical technologies were also considered in the decision process, which with the proper training would be simple for the end-user to learn.

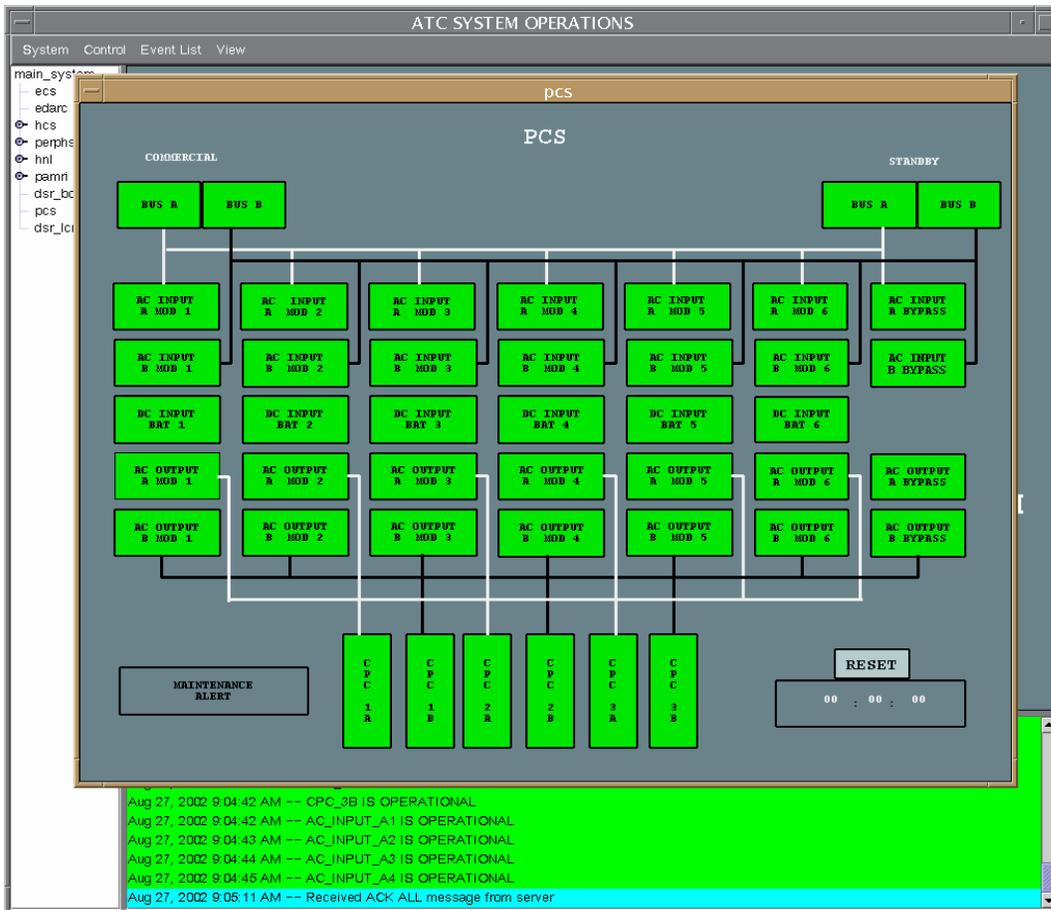
We believe that the current product of this POC, while still in its infancy in some areas, is a solid starting point for consideration in the development of an Integrated En Route Monitor and Control System. The following sub-sections provide more detailed information describing the functionality included in the four main components of the SL End User Client.

#### **6.2.3.3.1 SL Client Synchronization**

Since every SL-client connects to one instance of the NV User Application Server, object status change indications, object status change acknowledgements, and event list updates are synchronized across all connected clients. When Graphical Display Panel (map) objects change status, the fill color of the object changes to reflect the current state of the object. The SL-client application also triggers a color blink function upon a state change. The object status change acknowledgement end user action at one SL-client causes the object to stop blinking and forwards notification to all connected SL-clients causing all SL-client Graphical Display Panels to halt blinking for the specific object. Additionally, Event List Display Panel contents are synchronized across all active clients. Events displayed on one client are displayed on all clients. Client synchronization of object status and notification messages is accomplished using pre-defined, straightforward, and simple SL-client/NV User Application server interface messages over a TCP socket interface.

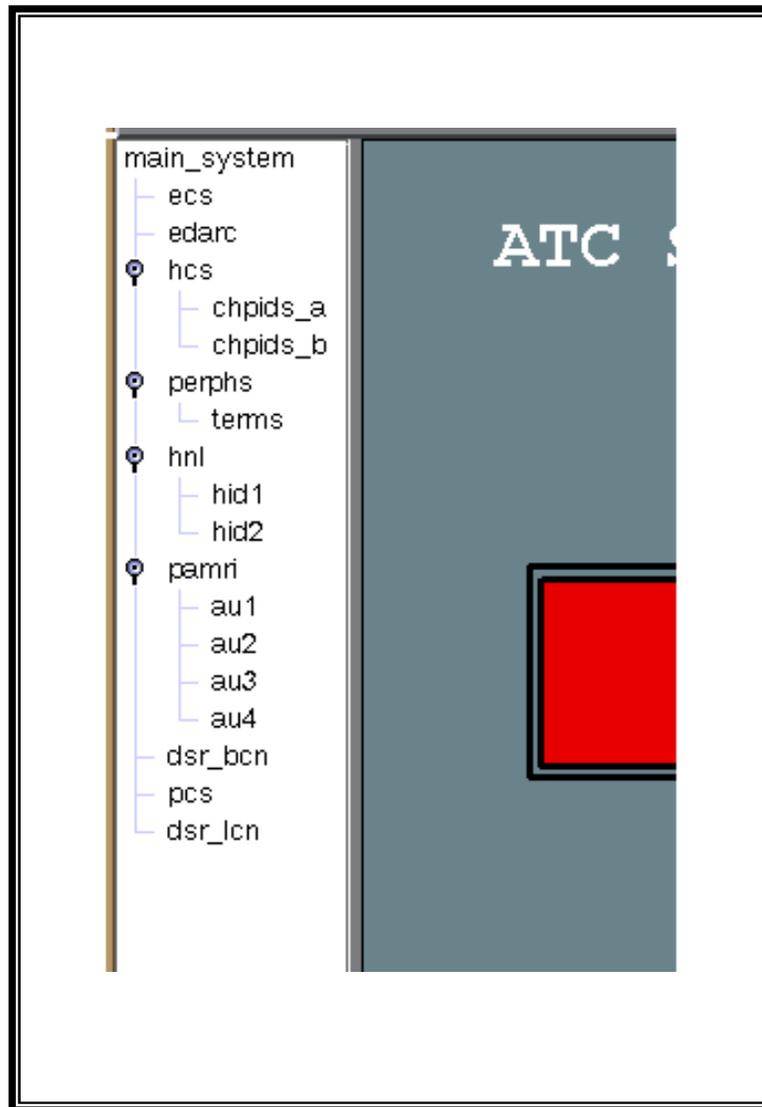
#### **6.2.3.3.2 Map/Submap Navigation**

The I2F EM&C product provides two methods for end-user submap access. One method is to double-click (drill-down on) a map object that has an extra-thick edge-style (indicating that there is a child map corresponding to the object). When this occurs, the child map is presented in a separate window that overlays the parent map window. This new window can be moved independently from the parent map window. This method allows the end user to monitor multiple levels of system functionality (i.e. HCS and EDARC) simultaneously. See Figure 5.12, EM&C Drill Down View Example.



**Figure 5.12 SL End User Client Drill Down Capability**

Additionally, the Navigation Panel provides direct access to maps within the application using a Java class called “MutableTreeNode”. This class is very similar to the PC-based windows explorer display method that most PC-users are familiar with. Using the Navigation Panel causes the Graphical Display Panel to replace its current contents with the selected map’s contents. This method minimizes the appearance of multiple windows that may complicate the overall conveyance of important status information. The SL End User Client Navigation Tree is depicted in Figure 5.13.



---

*Figure 5.13 SL End User ClientNavigation Tree*

---

### 6.2.3.3.3 Event List

The Event List Display Panel appears only at the foot of the top-level Graphical Display Panel and is used to convey real-time color-coded text-based notifications to the end user. The Event List is built with two base Java Swing classes called JList and JScrollPane. Figure 5.14 depicts the SL End User Client Event List.



*Figure 5.14 SL End User Client Event List*

These notifications consist of object status change notifications and user-directed notifications. User-directed notifications can be a specific user logon to the workstation or an indication that a user has acknowledged an object status change event.

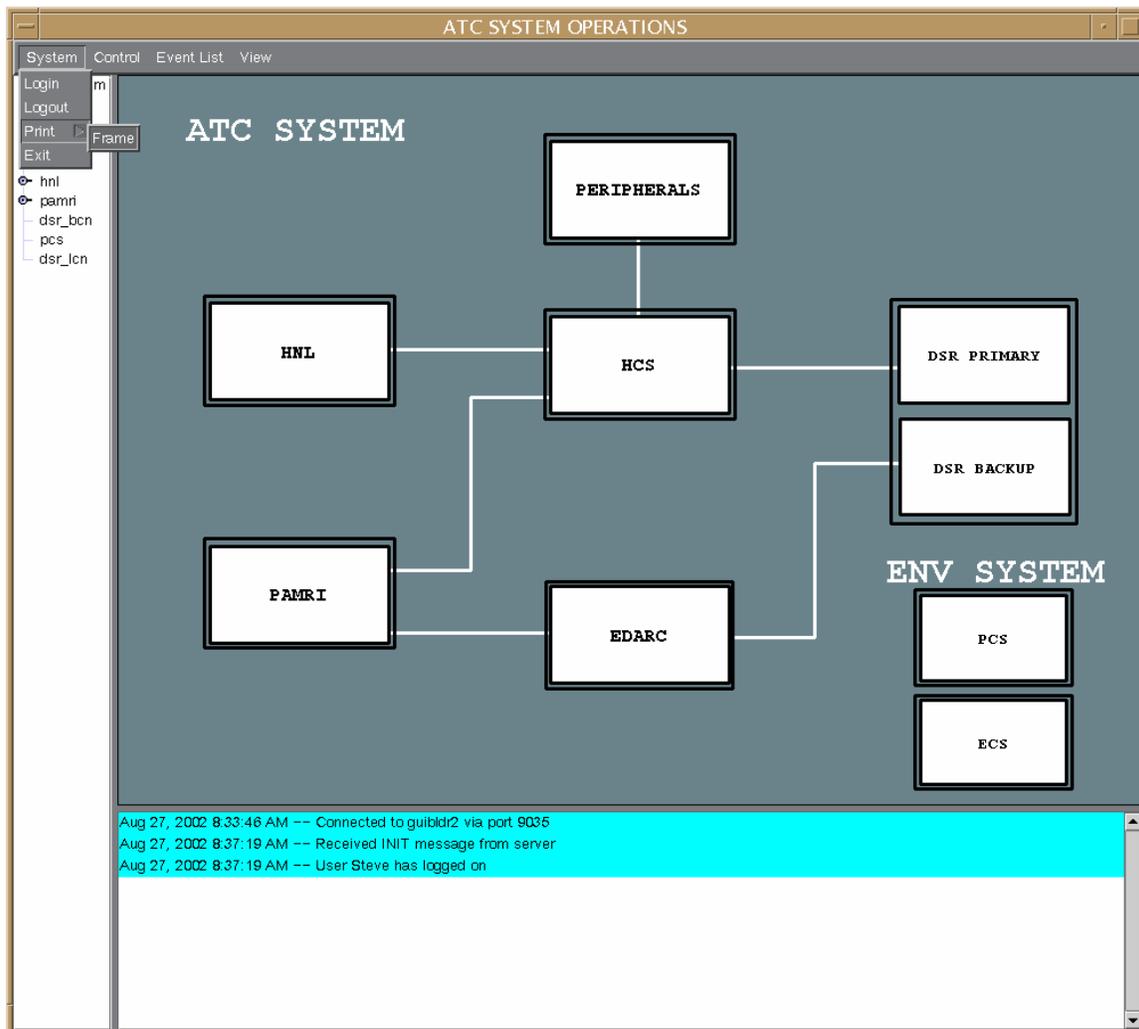
### 6.2.3.3.4 Main Menu Bar Functionality

The SL-client's Main Menu Bar provides a rich set of end user functionality that can be easily expanded and modified by a system developer. Since the initial functionality was developed using object oriented programming methods, extending or modifying current product functionality is fairly simple. The following sections describe the SL-client end user functions. The sections are divided by pull down menu as they appear in the application.

#### 6.2.3.3.4.1 System Menu Functions

- **Login** - Prompts the User for a Login name and Password. If there is a User currently logged on, it will log that user out and bring up a new Login Prompt.
- **Logout** - allows the current user to log out of system.
- **Security** - Enable or Disable password validation on acknowledgement of status change
- **Print** - Prints the Screen in its current state (WYSIWYG).

- **Exit** - closes main window and exits program.

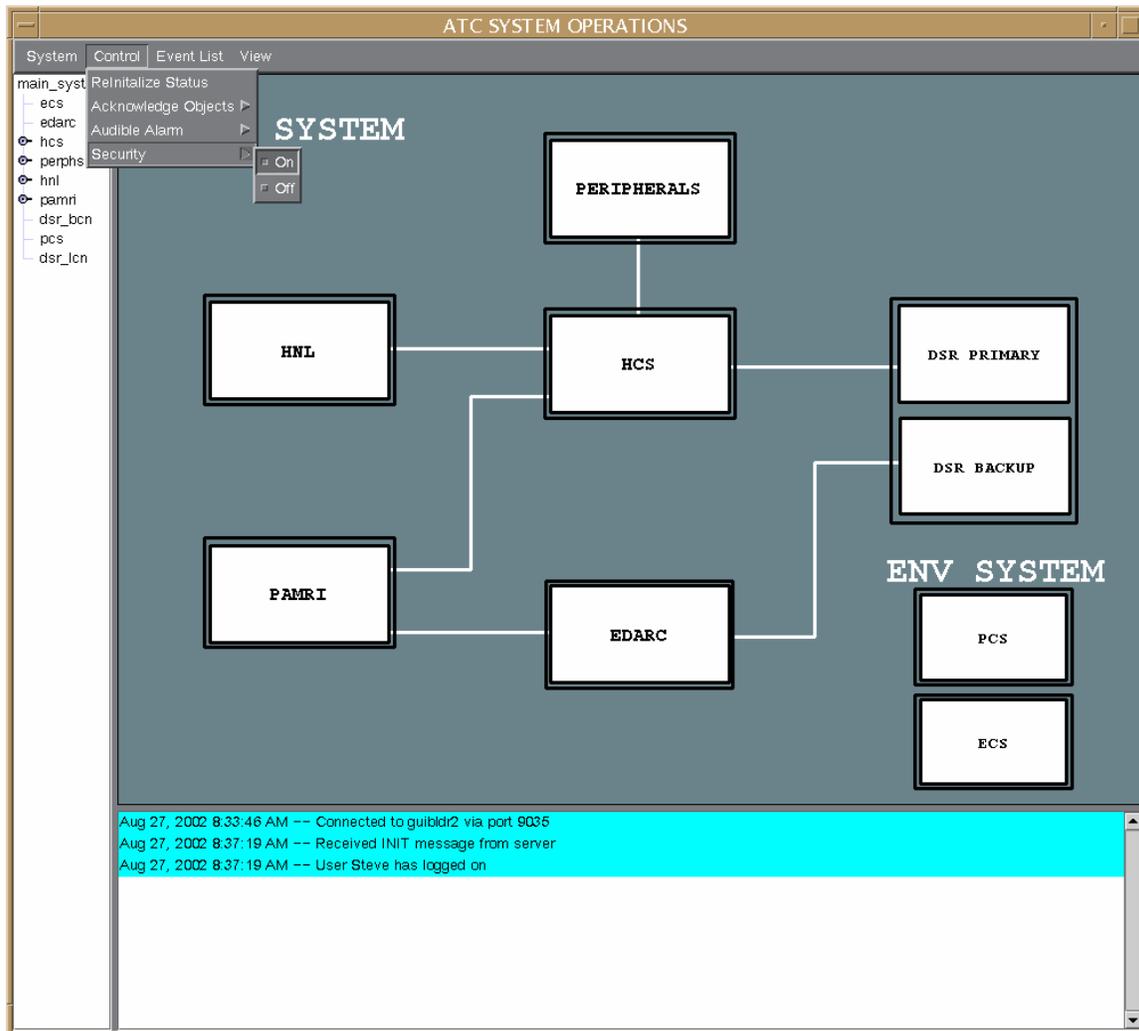


*Figure 5.15 The SL/Java User Interface System Menu*

#### 6.2.3.3.4.2 Control Menu Functions

- **Reinitialize From Server** - Disconnects from the server, and establishes a new connection.
- **Acknowledge All** - All objects that are blinking due to a status change will cease blinking.
- **Acknowledge Selected** - All selected objects that are blinking due to a status change will cease blinking.
- **Alarm** - Enable or Disable Audio Alert on Status Change

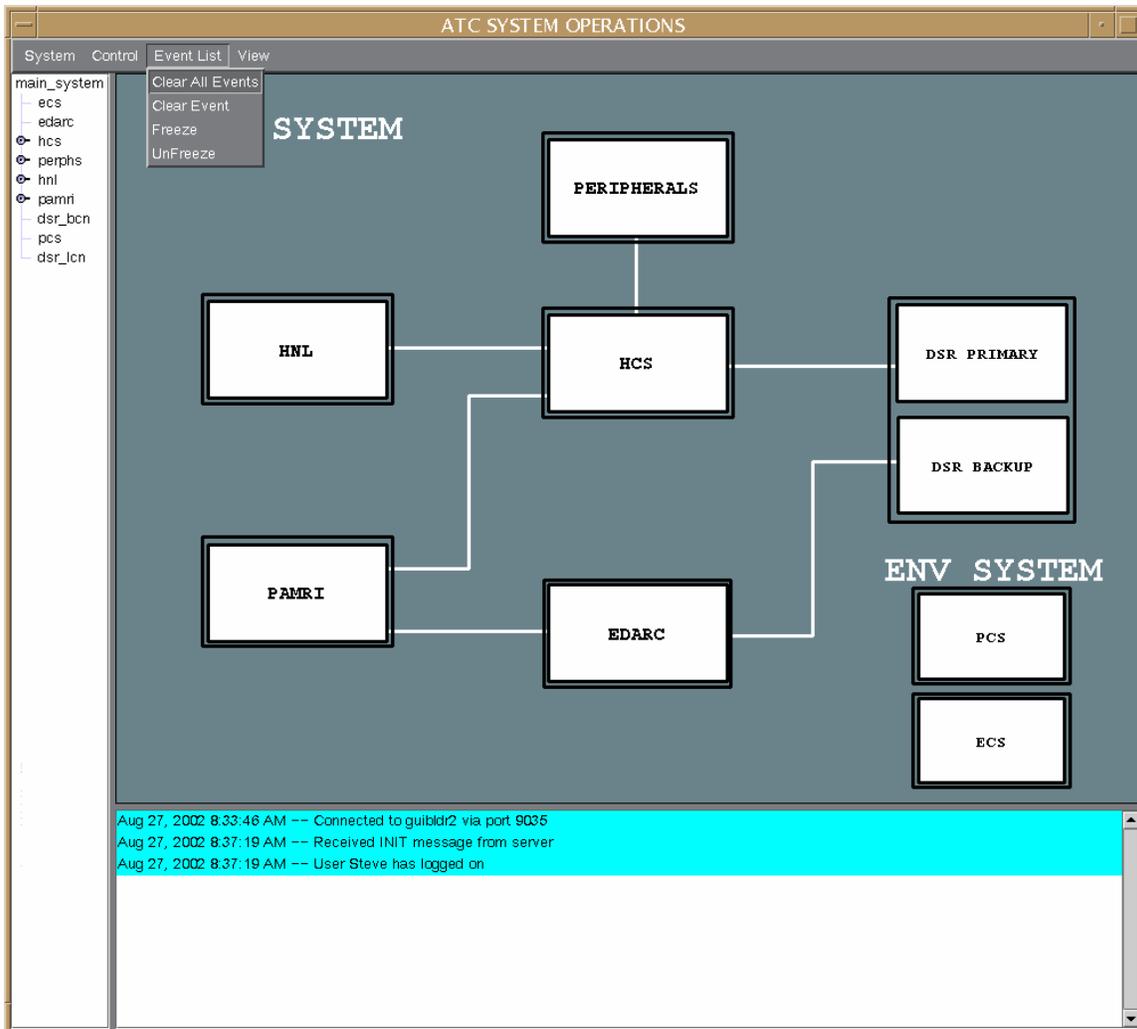
- **Security** - Enable or Disable Security Feature requiring password validation for certain end user commands. Turning this off requires a valid password.



*Figure 5.16 The SL/Java User Interface Control Menu*

#### 6.2.3.3.4.3 Event List Menu Functions

- **Clear All Events** - Removes all of the entries in the event list.
- **Clear Event** - Removes only selected entries from the event list. To select more than one event, hold down the Ctrl or Shift Key.
- **Freeze** - Holds the scrollbar in the event list at its current position, and remains when new entries are entered.
- **UnFreeze** - releases the freeze on the event list and scrolls to the newest entry.



*Figure 5.17 The SL/Java User Interface Event List Menu*

#### 6.2.3.3.4.4 View Menu Functions

- **Java Documentation** - Opens a web browser containing documentation on the Java classes and APIs used in this program.
- **Phone List** - Displays a web browser with a list of Phone Numbers to call for support.
- **Standard Operating Procedures** - Displays a list of the Standard Operating Procedures for all Devices.
- **View Logged Events** - Executes a separate program that provides a front-end GUI that will display archived event entries from an SQL Database. This application allows the user to specify search criterion to return the specific event

information desired. Filters include specific dates, user data (login/logoff), device identifiers, and status changes to these device identifiers. (See Section 5.2.4).

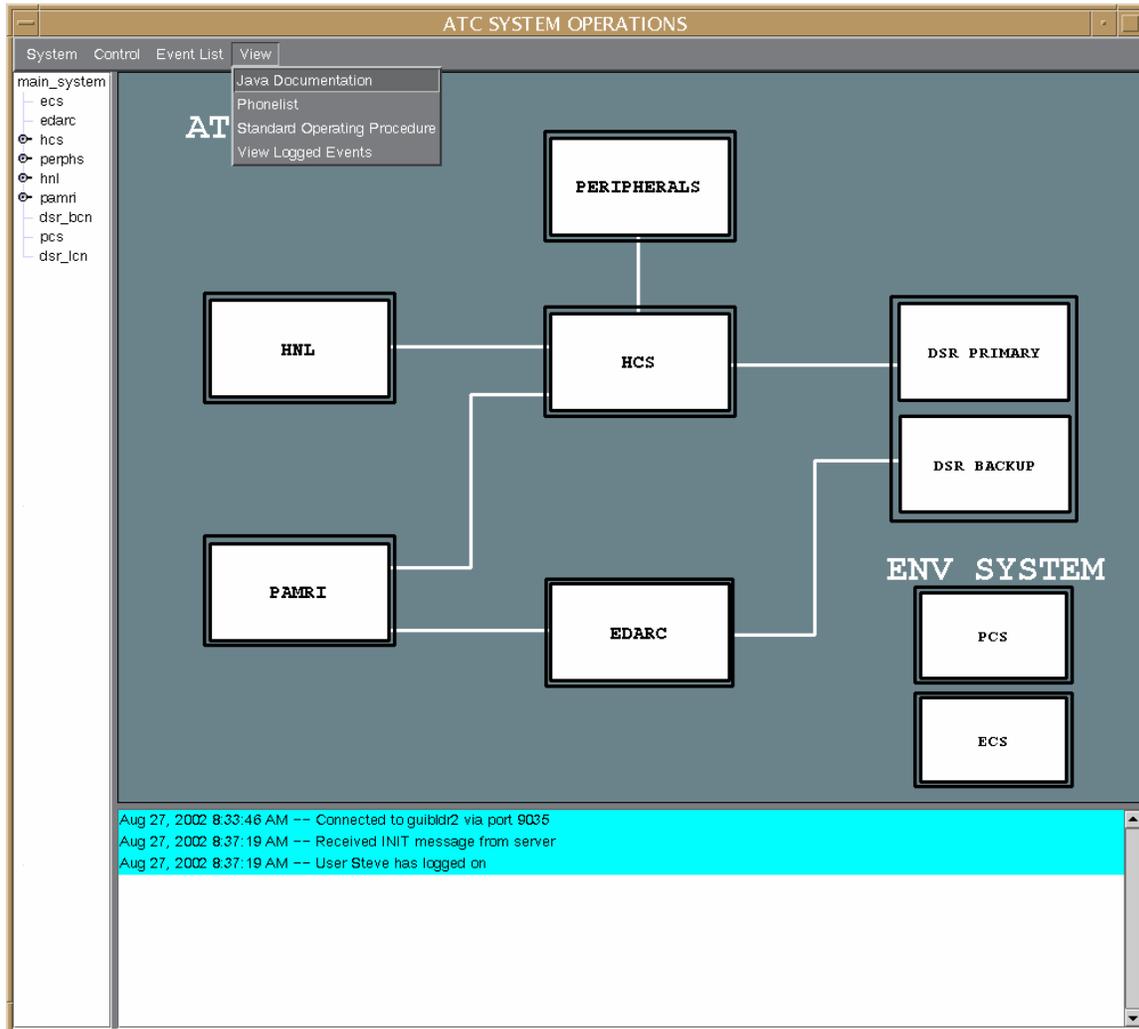


Figure 5.18 The SL/Java User Interface View Menu

### 6.2.4 External MySQL Relational Database Updater Application

This application is similar to the SL end user client only in the way it establishes a connection to the Netview Server Application. End user client Java code was re-used to rapidly develop this background application. The reason it is considered a background application is because it does not appear to the user as a visible GUI. It simply connects to the Netview Server Application at server startup and receives the same information that all other connected clients receive. Once it receives this information, it does not update a visible GUI application like the SL end user client application. It collects the information and writes text-based messages via another connection to an external Relational DataBase (RDB) product known as MySQL. MySQL is a freeware package

that is similar to the more powerful RDB products available from companies such as Oracle, Informix, and Sybase. The database connection is established using standard Java Database Connectivity (JDBC) Java classes and database inquiries/entries are made using standard Structured Query Language (SQL) code.

#### 6.2.4.1 Event History Viewer

The information stored in the Event database is made available to the end user via the View Logged Events menu choice presented in the previous section. Figure 5.14 illustrates the Event History GUI developed when the View Logged Events menu choice is selected. The Event History Viewer allows the user to view events by date, object status and using “keyword” search mechanism.

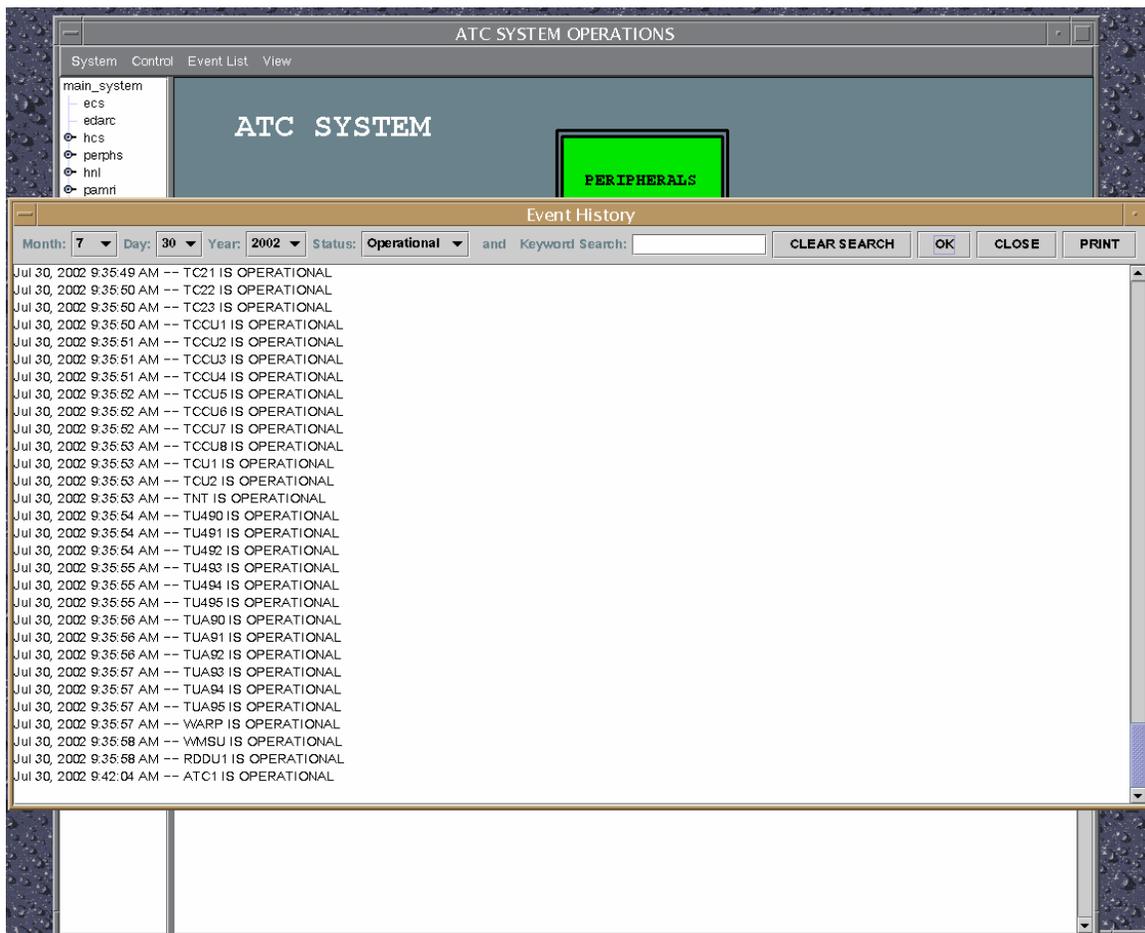


Figure 5.19 The MySQL Relational Database Event History GUI

#### 6.2.5 Subsystem Agents

Initial SNMP Subsystem agents were quickly developed using Korn shell and C programs to provide real-time status information of actual and simulated hardware and software

components and processes available at the I2F. The NAS subsystems that SNMP agents have recently been developed for include PAMRI, HID, KCONF, EDARC, and DAS devices. Currently, SNMP traps are the method employed for conveying status information on objects that the end user application is interested in monitoring. In the near future, SNMP MIBs will be developed for each subsystem that will take advantage of additional SNMP features allowing I2F engineers to integrate more sophisticated monitoring and control capabilities into the end user application.

#### ***6.2.5.1 Agent Simulation Tool***

In order to test or demonstrate the EM&C POC product, status agents needed to be developed. Prior to developing the current set of SNMP agents currently available, a tool was developed that simulates all the subsystem agents that would be represented in a full system as presented by the SL end-user client. This Status Simulator reads the adaptation file generated by the SL map build process and provides a mechanism to change the status of any object contained in the adaptation file via SNMP trap messages.

Additionally, the Status Simulator tool connects to the server as if it were an actual end-user client. It receives initialization and update information regarding the status of all NV User Application objects contained in the Netview Object Database similar to the SL end-user client. It provides this information to the user in its own GUI. Figure 5.20 and 5.21 depict the Status Simulator GUI. The Agent Simulator Tool allows the user to build a scripted set of status changes that can be sent in batch format to the SL end-user client application. This is useful for demonstrating particular failure mode situations repeatedly.

This GUI was developed using a scripting language call Tool Command Language/ Tool Kit (TCL/TK). This scripting language provide pre-built C++ graphical “widgets”, such as shapes, menus, and control buttons that can be displayed and manipulated by using the scripting language constructs.

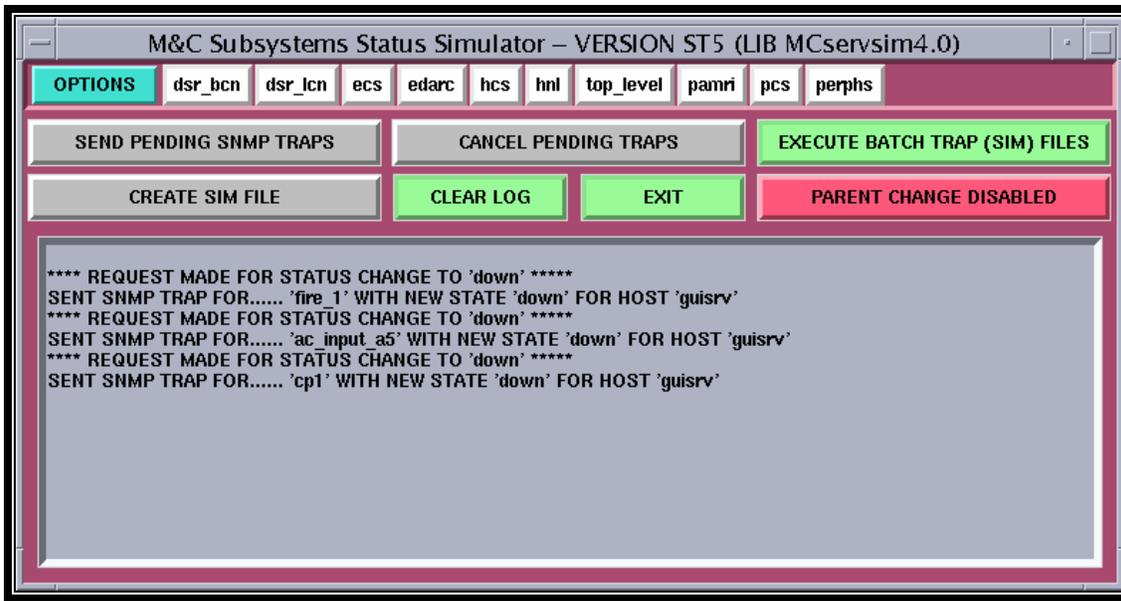


Figure 5.20 Agent Simulation Tool

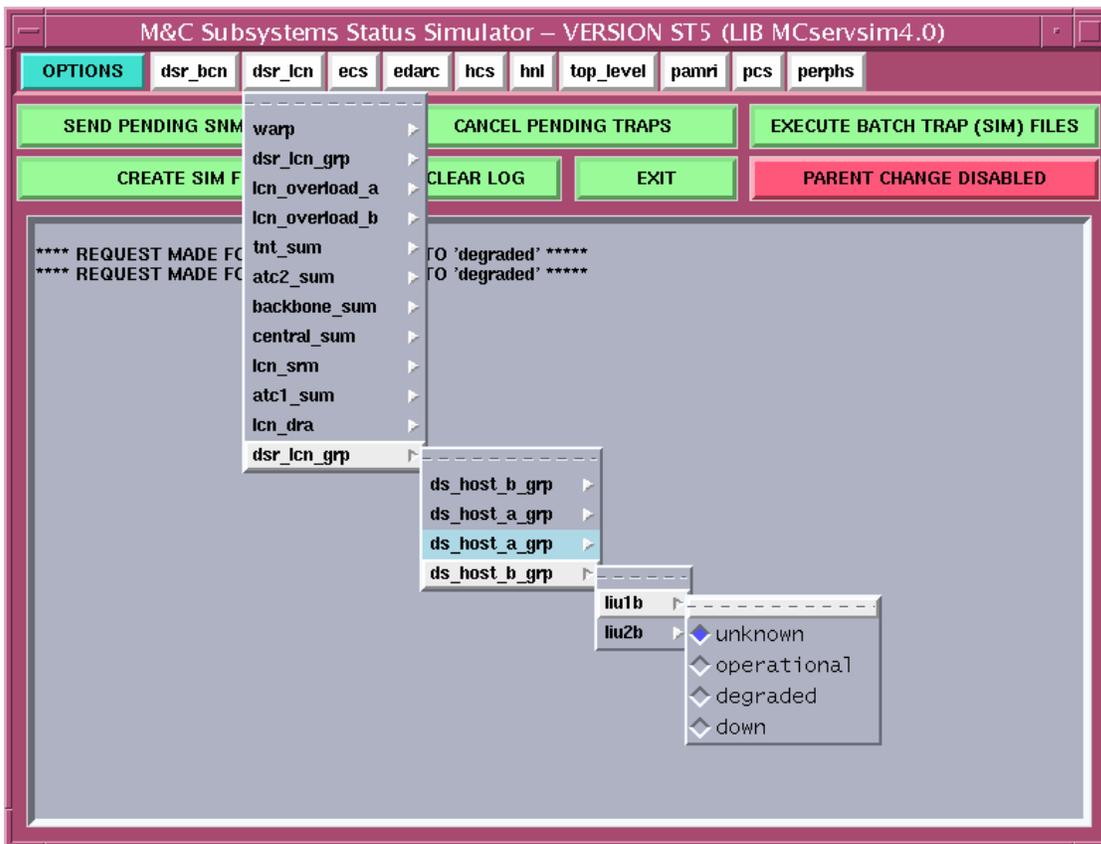


Figure 5.21 Agent Simulation Tool Change/Display Status Pull Down Menu

### ***6.2.5.2 DAS Replacement***

A follow on activity to the EM&C POC would be to consider a DAS replacement. An approach for this would have DAS connected devices convey their status directly to the Netview application. This means having an Ethernet connection and the network protocols operating as close to the devices as possible.

The company that developed the optically isolated connectors for the DAS (Opto22) has recently produced an Ethernet capable replacement for these connectors. This product is being evaluated as an integration component that would allow the devices connected to the DAS/RSD to seamlessly integrate with an SNMP based M&C subsystem. The Opto22 product can monitor various digital, analog and serial connections, and convey the status information over Ethernet via SNMP to a network manager application.

Opto22 has its own MIB and all of the information about the connector data points can be communicated through this data structure. The monitoring activity of the DAS, which is performed through custom C-based software, can be replaced with simple configuration items and some basic industrial control flowchart syntax. An optional syntax looks just like C or Pascal. Most of the low level system and I/O processing is performed by embedded code. Only the high-level control logic is required to be expressed by the system maintainer. The footprint of the replacement product is about 1/20th of the DAS. It can be deployed as several sub components as wall mountable units or possibly mounted in a cabinet of an existing device.

---

## **7 Summary**

This En Route Monitor and Control Proof of Concept conducted at the EIIF has shown that an Integrated Monitor and Control concept using new technology, Graphical User Interface Builders and open standard communications protocols could be implemented into the current or future En Route infrastructure in a fashion that would be acceptable to the user community. In addition, the product of the Proof of Concept provides a core application and design for the development of an integrated En Route M&C.

---

## **8 Proposed Requirements**

TBD

## Appendix A

---

### Example Field and NV User Application Field Definitions

The following are the definitions of the fields defined in Section 5.2.2.1.2. These definitions are used by Netview to give Netview objects particular characteristics. The definitions are in the required format for the Netview application.

```
Field "isNode" {
    Type Boolean;
    Flags capability;
}
```

```
Field "isRouter" {
    Type Boolean;
    Flags capability;
}
```

```
Field "Selection Name" {
    Type String;
    Flags name, locate;
}
```

```
Field "Tivoli Server Port"{
    Type Integer32;
    Flags locate;
}
```

```
Field "IP Status" {
    Type Enumeration;
    Flags locate;
    Enumeration "Unset",
                "Unknown",
                "Normal",
                "Marginal",
                "Critical",
                "Unmanaged",
                "Acknowledged",
                "User1",
                "User2",
                "Unreachable";
}
```

## NV User Application Fields

```
Field "Obj_Status" {  
    Type Enumeration;  
    Flags locate;  
    Enumeration "Operational",  
                "Degraded",  
                "Down",  
                "Unmanaged",  
                "Passive";  
}
```

```
Field "isNV_User_App_Obj" {  
    Type Boolean;  
    Flags capability;  
}
```

```
Field "Parent" {  
    Type String;  
    Flags locate, general;  
}
```

```
Field "isParent" {  
    Type Boolean;  
    Flags capability;  
}
```

```
Field "isAcknowledged" {  
    Type Boolean;  
    Flags capability;  
}
```

```
Field "LUID" {  
    Type String;  
    Flags locate, general;  
}
```

```
Field "dynaText" {  
    Type String;  
    Flags locate, general;  
}
```

## Appendix B

---

### Example of the NV User Application Adaptation Data File

The following adaptation data is part of a data file that is discussed in Section 5.2.3.2.3:

**Object\_name Parent\_name LUID Propagation\_Weight**

```
ac_input_a1 ac_input_a_grp 2224 0
ac_input_a2 ac_input_a_grp 2229 0
ac_input_a3 ac_input_a_grp 222E 0
ac_input_a4 ac_input_a_grp 2233 0
ac_input_a5 ac_input_a_grp 2238 0
ac_input_a6 ac_input_a_grp 223D 0
ac_input_a_by ac_input_a_grp 2242 0
ac_input_a_grp pcs NO_LUID 100
ac_input_b1 ac_input_b_grp 2225 0
ac_input_b2 ac_input_b_grp 222A 0
ac_input_b3 ac_input_b_grp 222F 0
ac_input_b4 ac_input_b_grp 2234 0
ac_input_b5 ac_input_b_grp 2239 0
ac_input_b6 ac_input_b_grp 223E 0
ac_input_b_by ac_input_b_grp 2243 0
ac_input_b_grp pcs NO_LUID 100
ac_output_a1 ac_output_a_grp 2227 0
ac_output_a2 ac_output_a_grp 222C 0
ac_output_a3 ac_output_a_grp 2231 0
ac_output_a4 ac_output_a_grp 2236 0
```

ac\_output\_a5 ac\_output\_a\_grp 223B 0  
ac\_output\_a6 ac\_output\_a\_grp 2240 0  
ac\_output\_a\_by ac\_output\_a\_grp 2244 0  
ac\_output\_a\_grp pcs NO\_LUID 100  
ac\_output\_b1 ac\_output\_b\_grp 2228 0  
ac\_output\_b2 ac\_output\_b\_grp 222D 0  
ac\_output\_b3 ac\_output\_b\_grp 2232 0  
ac\_output\_b4 ac\_output\_b\_grp 2237 0  
ac\_output\_b5 ac\_output\_b\_grp 223C 0  
ac\_output\_b6 ac\_output\_b\_grp 2241 0  
ac\_output\_b\_by ac\_output\_b\_grp 2245 0  
ac\_output\_b\_grp pcs NO\_LUID 100  
atc1\_sum dsr\_lcn NO\_LUID 0  
atc2\_sum dsr\_lcn NO\_LUID 0  
au1 au13\_grp NO\_LUID 0  
au13\_grp pamri NO\_LUID 100  
au1\_370a au1 NO\_LUID 0  
au1\_370b au1 NO\_LUID 0  
au1\_chnmain au1 NO\_LUID 100  
au1\_ddiapp au1 NO\_LUID 0  
au1\_mntagent au1 NO\_LUID 0  
au1\_mntcts au1 NO\_LUID 0  
au1\_mnterror au1 NO\_LUID 0  
au1\_mnthealt au1 NO\_LUID 0  
au1\_mntmain au1 NO\_LUID 100  
au1\_mntresol au1 NO\_LUID 0  
au1\_mntstats au1 NO\_LUID 0  
au1\_psar au1 NO\_LUID 0  
au1\_trnm2buf au1 NO\_LUID 0

## Appendix C

---

# An Example of a Decoded Sherill/Lubinski Map File

### Edarc.g –

```
mtran0
vis 1
detect 1
edarc: model
  . backgrflag 1
  detect 0
  fcolor 14
  fstyle 1
  finter 1
  fdir 0
  fpercent 100
  ecolor 7
  estyle 1
  ewidth 1
  background: frect -0.3799 0.6339 99.6201 75.6339
  . move 0.379913 -0.633865
  detect 1
  tcolor 0
  bcolor 0
  height 1.05901
  path 1
  font 2
  prec 0
  align 1 3
  size 0 0
  text "EDARC" 35.8638 68.4527
  . move -14516.1 -143.775
  . scale -14516.1 -143.775 404.839 3.13758
  fcolor 22
  ewidth 2
  tcolor 7
  height 1.5
  align 2 3
  tconstraint 0
  wmsu: ftrect 19 52 30 47 "WMSU"
  . dynprop \
    (* \
      (fcolor fc_wmsu) \
      (estyle es_wmsu))
  . userdata "edarc_grp:2B25:"
  . move -2.27274 -5
  . scale -2.27274 -5 0.909091 1
  fcolor 30
  height 1.04001
  NO_OBJECT_rkm: ftrect 84 25 92 21 "RKM"
```

```

. move 0 -5.25
. scale 0 -5.25 1 1.25
fcolor 22
height 1.3
rcc: ftrect 67 26 75 21 "RCC"
. dynprop \
  (* \
    (fcolor fc_rcc) \
    (estyle es_rcc))
. userdata "eif_grp:2B08:"
. move -1 1
. scale -1 1 1 1
fcolor 30
height 1.21875
NO_OBJECT_ssc: ftrect 45 38 66 23 "SYSTEM STATUS\nCONTROL"
. move -3.14285 -2.53333
. scale -3.14285 -2.53333 1.04762 1.06667
height 1.3
NO_OBJECT_iot3_4: ftrect 15 34 25 29 "IOT 3/4"
. move 0 -2
. scale 0 -2 1 1
fcolor 22
cp2: ftrect 15 27 25 22 "CP 2"
. dynprop \
  (* \
    (fcolor fc_cp2) \
    (estyle es_cp2))
. userdata "ecp_grp:2B04:"
. move 0 -1
. scale 0 -1 1 1
fcolor 1
ewidth 3
tcolor 0
edarc_not_updating: ftrect 12 17 31 11 "EDARC NOT\nUPDATING"
. dynprop \
  (* \
    (fcolor fc_edarc_not_updating) \
    (vis vi_edarc_not_updating) \
    (estyle es_edarc_not_updating))
. userdata "edarc:2B31:100"
. move 11.875 -3.375
fcolor 22
ewidth 2
tcolor 7
height 1.56
iot1: ftrect 44 15 53 9 "IOT 1"
. dynprop \
  (* \
    (fcolor fc_iot1) \
    (estyle es_iot1))
. userdata "eio_grp:2B05:"
. move 0 2.5
. scale 0 2.5 1 0.833333
height 1.3
iot2: ftrect 56 15 64 10 "IOT 2"
. dynprop \
  (* \
    (fcolor fc_iot2) \
    (estyle es_iot2))
. userdata "eio_grp:2B26:"
. move -7 0

```

```

. scale -7 0 1.125 1
lp: ftrect 67 15 76 10 "LP"
. dynprop \
  (* \
    (fcolor fc_lp) \
    (estyle es_lp))
. userdata "eio_grp:2B01:"
cr: ftrect 78 15 86 10 "CR"
. dynprop \
  (* \
    (fcolor fc_cr) \
    (estyle es_cr))
. userdata "eio_grp:2B02:"
ecolor 0
ewidth 4
NO_OBJECT_CP1_IOT3_4_IF: line 20 34 20 32
. move 0 16
. scale 0 16 1 0.5
NO_OBJECT_CP2_IOT34_IF: line 20 27 20 25
. move 0 13.5
. scale 0 13.5 1 0.5
NO_OBJECT_MTU1_CP1_IF: line 15 36 8 36 8 32
NO_OBJECT_MTU1_CP2_IF: line 8 27 8 23 15 23
NO_OBJECT_MTU2_CP1_IF: line 25 36 32 36 32 32
. move 0 8
. scale 0 8 1 0.75
NO_OBJECT_MTU2_CP2_IF: line 25 23 32 23 32 27
. move 0 13.5
. scale 0 13.5 1 0.5
NO_OBJECT_DG_COMM_IF: line 66 36 84 36
. move 8.70833 0
. scale 8.70833 0 0.868056 1
NO_OBJECT_GPO_IF: line 84 31 66 31
. move 9.95833 0.75
. scale 9.95833 0.75 0.847222 1
NO_OBJECT_GPI_IF: line 66 30 84 30
. move 10.0833 -1
. scale 10.0833 -1 0.847222 1
NO_OBJECT_RKM_IF: line 74 25 84 25
. move 0 -1
NO_OBJECT_DP_SSC_IF: line 55 38 55 50
NO_OBJECT_WMSU_SSC_IF: line 25 46 50 46 50 38
. move 0 9.5
. scale 0 9.5 1 0.75
NO_OBJECT_CP1_SSC_IF: line 25 36 44 36
NO_OBJECT_CP2_SSC_IF: line 25 24 44 24
ecolor 7
ewidth 2
cpl: ftrect 15 38 25 33 "CP 1"
. dynprop \
  (* \
    (fcolor fc_cpl) \
    (estyle es_cpl))
. userdata "ecp_grp:2B03:"
fcolor 3
height 1.66667
gpoi: ftrect 17 66 27 61 "GPO"
. dynprop \
  (* \
    (fcolor fc_gpoi) \
    (estyle es_gpoi))

```

```

. userdata "eif_grp:2B06"
. move 70.075 -6.47501
. scale 70.075 -6.47501 0.65 0.6
gpII: ftrect 17 66 27 61 "GPI"
. dynprop \
  (* \
    (fcolor fc_gpII) \
    (estyle es_gpII))
. userdata "eif_grp:2B07:"
. move 70.075 -9.47501
. scale 70.075 -9.47501 0.65 0.6
fcolor 30
height 1.34738
NO_OBJECT_host_darc: ftrect 84 40 92 35 "HOST\n/DARC"
. move 3.625 -14.4062
. scale 3.625 -14.4062 1 1.1875
fcolor 1
ewidth 3
tcolor 0
height 2
over_temperature: ftrect 12 17 31 11 "OVER TEMP"
. dynprop \
  (* \
    (fcolor fc_over_temperature) \
    (vis vi_over_temperature) \
    (estyle es_over_temperature))
. userdata "edarc:2B30:100"
. move -8.25 3.35417
. scale -8.25 3.35417 1 0.729167
rm_threshold: ftrect 12 17 31 11 "REFRESH MEM\nTHRESHOLD"
. dynprop \
  (* \
    (fcolor fc_rm_threshold) \
    (vis vi_rm_threshold) \
    (estyle es_rm_threshold))
. userdata "edarc:2B29:100"
. move -8.25 -2.45833
. scale -8.25 -2.45833 1 0.729167
NO_OBJECT_SSC_DEVS_IF: group
  fcolor 22
  ecolor 0
  ewidth 4
  line 48 23 48 15
  . move 0 1.875
  . scale 0 1.875 1 0.875
  line 72 19 72 15
  line 61 19 61 15
  line 48 19 82 19 82 15
endg
ecolor 7
ewidth 2
tcolor 7
height 1.3
dg_comm: ftrect 84 40 92 35 "DG \nCOMM"
. dynprop \
  (* \
    (fcolor fc_dg_comm) \
    (estyle es_dg_comm))
. userdata "eif_grp:2B27:"
. move 13.25 -1
. scale 13.25 -1 0.8125 1

```

```

fcolor 30
height 1.34738
NO_OBJECT_dg: ftrect 84 40 92 35 "DG"
. move 3.52576 -7.5
. scale 3.52576 -7.5 1 1.1875
fcolor 22
ecolor 0
ewidth 4
NO_OBJECT_RDDU_RMUXB_IF: line 25 61 39 61
. move 1.99782 0.25
. scale 1.99782 0.25 0.464912 1
NO_OBJECT_RDDU_RMUXA_IF: line 25 61 39 61
. move 3.73213 -9.75
. scale 3.73213 -9.75 0.410715 1
fcolor 30
ecolor 7
ewidth 2
height 1.25
NO_OBJECT_rddu2: ftrect 17 60 27 55 "RDDU2"
. move -13 -17.25
. scale -13 -17.25 1 1.2
NO_OBJECT_rddu1: ftrect 17 66 27 61 "RDDU1"
. move -13 -16.7
. scale -13 -16.7 1 1.2
NO_OBJECT_RDDU2_RMUXB_IF: group
  fcolor 22
  ecolor 0
  ewidth 4
  NO_OBJECT_RDDU_RMUXB_IF: line 25 61 39 61
  . tran 0.142857 0 0 0 1 0
  . move 14.1786 -0.875
  _NO_OBJECTRDDU_RMUXB_IF: line 25 61 39 61
  . tran 2.17555e-08 -1 0 0.518171 5.21556e-08 0
  . move 78.5927 40.0413
  NO_OBJECT_RDDU_RMUXB_IF: line 25 61 39 61
  . move 7.30356 -8.125
  . scale 7.30356 -8.125 0.267858 1
endg
NO_OBJECT_RDDU1_RMUXA_IF: group
  NO_OBJECT_RDDU_RMUXB_IF: line 25 61 39 61
  . tran 0.162179 0 0 0 1 0
  . move 9.925 -2.75
  NO_OBJECT_RDDU_RMUXB_IF: line 25 61 39 61
  . tran 2.17555e-08 -1 0 0.339286 3.41503e-08 0
  . move 77.2339 45.0178
  RDDU_RMUXB_IF: line 25 61 39 61
  . move 10.2232 -7.375
  . scale 10.2232 -7.375 0.241072 1
endg
NO_OBJECT_RMUXB_DP_IF: line 25 61 39 61
. move 23.0716 -0.375
. scale 23.0716 -0.375 0.232139 1
NO_OBJECT_RMUXA_DP_IF: line 25 61 39 61
. move 23.4465 -8.5
. scale 23.4465 -8.5 0.232139 1
ecolor 7
ewidth 2
height 2
ecp_grp: ftrect 3 34 13 29 "C"
. filled 0
. dynprop \

```

```

        (* \
          (vis vi_ecp_grp))
      . userdata "edarc::100"
      . userword 636630527
      . move 10.8375 -85.475
      . scale 10.8375 -85.475 1.1375 3.65
height 1
mtu1: ftrect 3 34 13 29 "MTU 1"
. dynprop \
  (* \
    (fcolor fc_mtu1) \
    (estyle es_mtu1))
. userdata "mtu_grp:2B23:"
. move 0 -2
mtu2: ftrect 27 34 37 29 "MTU 2"
. dynprop \
  (* \
    (fcolor fc_mtu2) \
    (estyle es_mtu2))
. userdata "mtu_grp:2B24:"
. move 0 -2
height 2
mtu_grp: ftrect 3 34 13 29 ""
. filled 0
. dynprop \
  (* \
    (vis vi_mtu_grp))
. userdata "edarc_grp::"
. move -8.3625 -19.325
. scale -8.3625 -19.325 3.5375 1.55
rmux_grp: ftrect 3 34 13 29 "R"
. filled 0
. dynprop \
  (* \
    (vis vi_rmux_grp))
. userdata "edarc::100"
. move 15.8909 -34.125
. scale 15.8909 -34.125 1.08531 2.875
fcolor 3
height 1.66667
rmuxb: ftrect 17 66 27 61 "RMUX B"
. dynprop \
  (* \
    (fcolor fc_rmuxb) \
    (estyle es_rmuxb))
. userdata "rmux_grp:2B33:"
. move 2.625 5.25029
. scale 2.625 5.25029 1 0.875
fcolor 22
rmuxa: ftrect 17 66 27 61 "RMUX A"
. dynprop \
  (* \
    (fcolor fc_rmuxa) \
    (estyle es_rmuxa))
. userdata "rmux_grp:2B32:"
. move 2.625 -6.42818
. scale 2.625 -6.42818 1 0.925052
height 2
edarc_grp: ftrect 3 34 13 29 "S"
. filled 0
. dynprop \

```

```

        (* \
          (vis vi_edarc_grp))
. userdata "edarc::100"
. move 36.6638 -77.1
. scale 36.6638 -77.1 2.27872 3.4
eio_grp: ftrect 3 34 13 29 "I"
. filled 0
. dynprop \
  (* \
    (vis vi_eio_grp))
. userdata "edarc_grp::"
. move 30.5359 -36.2
. scale 30.5359 -36.2 4.33378 1.55
eif_grp: ftrect 3 34 13 29 "O"
. filled 0
. dynprop \
  (* \
    (vis vi_eif_grp))
. userdata "edarc_grp:NO_LUID:"
. move 58.5888 -96.7
. scale 58.5888 -96.7 2.30372 4.05
fcolor 4
ewidth 1
edp_grp: ftrect 31.125 63.875 97.375 49.25 "D"
. filled 0
. dynprop \
  (* \
    (vis vi_edp_grp))
. userdata "edarc:NO_LUID:100"
. move 0 0
fcolor 13
ewidth 2
frect 42.0268 67.6484 99.1135 55.0847
. move -14.9913 -6.99754
. scale -14.9913 -6.99754 1.1211 1.03473
fcolor 22
height 1.3
edp1: ftrect 39 67 47 62 "DP1"
. dynprop \
  (* \
    (fcolor fc_edp1) \
    (estyle es_edp1))
. userdata "edp_grp:2B09:"
. move -6 -5
edp2: ftrect 39 61 47 56 "DP2"
. dynprop \
  (* \
    (fcolor fc_edp2) \
    (estyle es_edp2))
. userdata "edp_grp:2B10:"
. move -6 -5
edp3: ftrect 48 67 56 62 "DP3"
. dynprop \
  (* \
    (fcolor fc_edp3) \
    (estyle es_edp3))
. userdata "edp_grp:2B11:"
. move -6.12497 -5
. scale -6.12497 -5 1.0026 1
edp4: ftrect 48 61 56 56 "DP4"
. dynprop \

```

```

      (* \
        (fcolor fc_edp4) \
        (estyle es_edp4))
. userdata "edp_grp:2B12:"
. move -5.97917 -5
edp5: ftrect 57 67 65 62 "DP5"
. dynprop \
  (* \
    (fcolor fc_edp5) \
    (estyle es_edp5))
. userdata "edp_grp:2B13:"
. move -5.95834 -5
edp6: ftrect 57 61 65 56 "DP6"
. dynprop \
  (* \
    (fcolor fc_edp6) \
    (estyle es_edp6))
. userdata "edp_grp:2B14:"
. move -5.95834 -5
edp7: ftrect 66 67 74 62 "DP7"
. dynprop \
  (* \
    (fcolor fc_edp7) \
    (estyle es_edp7))
. userdata "edp_grp:2B15:"
. move -5.93752 -5
edp8: ftrect 66 61 74 56 "DP8"
. dynprop \
  (* \
    (fcolor fc_edp8) \
    (estyle es_edp8))
. userdata "edp_grp:2B16:"
. move -5.93752 -5
edp9: ftrect 75 67 83 62 "DP9"
. dynprop \
  (* \
    (fcolor fc_edp9) \
    (estyle es_edp9))
. userdata "edp_grp:2B17:"
. move -5.91669 -5
edp10: ftrect 75 61 83 56 "DP10"
. dynprop \
  (* \
    (fcolor fc_edp10) \
    (estyle es_edp10))
. userdata "edp_grp:2B18:"
. move -5.91669 -5
edp11: ftrect 84 67 92 62 "DP11"
. dynprop \
  (* \
    (fcolor fc_edp11) \
    (estyle es_edp11))
. userdata "edp_grp:2B19:"
. move -5.89586 -5
edp12: ftrect 84 61 92 56 "DP12"
. dynprop \
  (* \
    (fcolor fc_edp12) \
    (estyle es_edp12))
. userdata "edp_grp:2B20:"
. move -5.89586 -5

```

```
edp13: ftrect 84 67 92 62 "DP13"  
. dynprop \  
    (* \  
        (fcolor fc_edp13) \  
        (estyle es_edp13))  
. userdata "edp_grp:2B21:"  
. move 3.125 -5  
edp14: ftrect 84 61 92 56 "DP14"  
. dynprop \  
    (* \  
        (fcolor fc_edp14) \  
        (estyle es_edp14))  
. userdata "edp_grp:2B22:"  
. move 3.125 -5  
endm
```