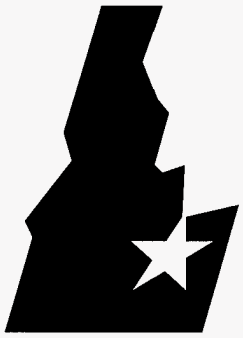
INEEL/EXT-98-00303

March 1998



Idaho National Engineering Laboratory

Corral Monitoring System Assessment Results

E. E. Filby and K.J. Haskell

LOCKHEED MARTIN

INEEL/EXT-98-00303

Corral Monitoring System Assessment Results

E. E. Filby and K.J. Haskell

Published March 1998

Idaho National Engineering and Environmental Laboratory
National Security Programs
Lockheed Martin Idaho Technologies Company
Idaho Falls, Idaho 83415

Prepared for the
Defense Special Weapons Agency
under IACRO HD1102-7-1490-15
Work For Others authorized by the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-94ID13223

DISCLAIMER

ABSTRACT

This report describes the results of a functional and operational assessment of the Corral Monitoring Systems (CMS). The assessment was performed at three levels: One level evaluated how well the planned approach addressed the target application, which involved tracking sensitive items moving into and around a site being monitored as part of an international treaty or other agreement. The second level examined the overall design and development approach, while the third focused on individual sub-systems within the total package. Unfortunately, the system was delivered as dis-assembled parts and pieces, with very poor documentation. Thus, the assessment was based on fragmentary operating data coupled with an analysis of what documents were provided with the system. The system design seemed to be a reasonable match to the requirements of the target application; however, important questions about site manning and top-level administrative control were left unanswered. Four weaknesses in the overall design and development approach were detected: (1) Poor configuration control and management, (2) inadequate adherence to a well-defined architectural standard, (3) no apparent provision for improving top-level error tolerance, and (4) weaknesses in the object oriented programming approach. The individual sub-systems were found to offer few features or capabilities that were new or unique, even at the conceptual level. The CMS might possibly have offered a unique *combination* of features, but this level of integration was never realized, and it had no unique capabilities that could be readily extracted for use in another system.

EXECUTIVE SUMMARY

This work was sponsored by the Defense Special Weapons Agency (DSWA) as the *CMS Assessment and Integration project*, under the programmatic designation IACRO HD1102-7-1490-15. The CMS was designed to detect and document accountable items entering or leaving a monitored site. Its development was motivated by the possibility that multiple sites in the nuclear weapons states of the former Soviet Union might be opened to such monitoring under the provisions of the Strategic Arms Reduction Treaty (START). The Local Operator Workstation (LOW) component of the CMS would collect sensor information with the intent of identifying unauthorized boundary crossings. Data collected by the LOW would be downloaded via telephone or satellite telemetry into an identically-structured database resident in the Remote Operator Workstation at the Remote Data Center.

The design strategy for the CMS included making the maximum possible use of commercial off-the-shelf (COTS) components. The Operator workstations would host a software suite centered around custom-designed System Integration Software (SIS) developed within the object-oriented programming (OOP) environment, VisualAge™ (an International Business Machines, IBM, product). Initial CMS development took place at Raytheon Service Company in 1995 and into January of 1996. When this assessment project was initiated in May 1997, a extensive set of CMS hardware and software components were gathered together and shipped to the INEEL. Along with two workstations, a total of 16 RAID (Redundant Array of Independent Disk) drives were included. One of the early problems was to identify which disks were usable boot disks, and which were only for data storage. This was further complicated by the fact that there were just under *150 thousand* separate files located in hundreds of directories. While many were duplicates (often in a different directory on the same disk or partition), many were not and there seemed to be nearly a thousand unique .EXE files.

The original assessment project had two basic goals. The first goal was to evaluate the operability and functionality of the System, while the second was to assess the feasibility of integrating its capabilities with a "distributed" monitoring and control network. However, as early work proceeded, it became clear that many parts of the prototype were incomplete, un-tested, or based on obsolete technology. The focus then returned to the first goal, with the proviso that the project would emphasize an in-depth analysis of the functionality provided by various parts of the CMS. Understanding how CMS provided these functions and whether they did, or did not, address critical nonproliferation surveillance needs would provide valuable input for future programs in this area. It should be emphasized that the documentation received with the CMS components was not very complete. Also, in their rush to finish the prototype, the developers seem to have lost all version control, and were never able to regain it.

As the functional assessment proceeded, it was concluded that a tremendous level of effort would be required to reverse-engineer enough system knowledge to decide which modules were operable, finish interim versions for software that was "almost there," and fill in the many gaps in the documentation. It did not help that some design features had crucial dependencies upon commercial components that had moved ahead significantly in the 14 months the CMS was standing still. Beyond this, the assessment highlighted four areas of concern in the overall approach: The first problem was the poor configuration control and management already highlighted by the loss of version control. The second problem was inadequate adherence to a clear architectural standard. While IBM OS/2 was the "standard" selected, some vendors of their selected COTS components had no plans to ever support that particular operating system. The third problem was the lack of any provision for improving top-level error tolerance. Error recovery routinely required a complete system re-boot, and there was almost no mention of exception handlers in

any of the design documents provided to us. Finally, although they did their development in an OOP environment, their overall approach was very weak. Many normal elements of proper OO analysis and design were missing, and the actual approach seemed to be an unplanned commingling of OO and procedure-oriented philosophies.

Because all the key CMS components were embedded in the workstation, everything had to work together properly for us to be able to get any kind of system up and running. This turned out to be very difficult. Some individual equipment items could be operated, but these were not of much use. Despite systematic disk swapping, at no time could a reasonably complete array of CMS components be made to work together. The final conclusion was that it was basically impossible to implement and evaluate many individual features planned for the CMS.

In addition to the top-level functional assessment, eleven sub-systems identified as being the fundamental components of the CMS were evaluated on a case-by-case basis. The *User Interface* sub-system was meant to provide "navigational" branches to all the other CMS functions. This component was found to be "marginally adequate," but had no special or unique features. The *Security Log-On* sub-system was also only marginally adequate. One problem was that the hardware security check gave no visual indication of any kind to show that the check had occurred. Actually, we had a lot of trouble with this whole hardware-based security approach. Other than that, this sub-system performed like a standard database password protection system. The *Event Assessment* sub-system was considered potentially one of the best features of the CMS. Unfortunately, we experienced a great many debugger error messages as we worked with this sub-system, and encountered a number of unexplainable lockups and other problems. Basically, this sub-system was too far from being a finished product for us to assess how well it might have worked.

The *System Configuration* sub-system was meant to allow one to define and configure sensor suites, establish user profiles, and even configure individual sensors. It was supposed to do all this via an "intuitive" Graphical User Interface (GUI); however, we found that the only way to actually generate a new working site configuration was to modify the underlying SmallTalk code itself. A great deal of work remained to make this sub-system usable. Some fairly serious anomalies were observed in trying to run the *Maintenance Functions* sub-system, so it could only be assessed "on paper." We decided that the design did not offer much that was particularly innovative or unusual. Access to the *Data Transfer* sub-system was not really possible. The custom scripts apparently worked for the developer under carefully controlled conditions, but we had very little success. We finally decided that the obsolete OS/2-specific links created for this program probably had little to offer for new configurations that might be planned.

The *Data Management* sub-system was meant to allow the user to manage disk utilization and archived data. Unfortunately, the only portion of this sub-system that could be tested was the deletion of data using the "Assessment" window. It did not appear that the CMS design offered any ideas that might be transferable for use in some other application. We spent by far the most effort trying to make the *Data Acquisition* sub-system work. Unfortunately, these efforts were largely unsuccessful because many units would not run, or ran inconsistently or unreliably. Plans for this sub-system called for the use of the COTS Alarm Assessment Sensor Processor (AASP™) neural network algorithm, but it was never implemented. Basically, the device interface part of the CMS was not something we felt had any promise for future applications. The *Database* sub-system seemed to offer only standard functionality. Thus, we did not feel there was that much to learn from the work done on this part of the system. The total lack of documentation made it impossible to determine if the *Scheduler* sub-system actually worked, or was even a part of the CMS SIS. Similarly, we had no way to assess the CMS *Compression/Decompression* sub-system because there was no video input to compress or decompress.

FORWARD

This work was sponsored by the Defense Special Weapons Agency (DSWA) as the *CMS Assessment and Integration project*, under the programmatic designation IACRO HD1102-7-1490-15. It was approved by the U.S. DOE Idaho Operations Office as an authorized Work For Others effort in May 1997. The CMS prototype had been developed previously as a project sponsored by the Defense Nuclear Agency (DNA, predecessor to the DSWA). The original intent of the current project was to evaluate the maturity of the CMS package, to assess what portions of it might be integrated with elements of the DOE-sponsored Modular Integrated Monitoring System (MIMS), and (given a positive assessment) assist in integrating appropriate CMS and MIMS components and functions. Before the evaluation had proceeded very far, it was decided there would be no follow-on integration work, so the work scope was changed to focus on this thorough evaluation of all the features of the existing CMS. This report contains the results of that assessment effort.

ACKNOWLEDGMENTS

Many individuals besides the authors contributed to the work described in this report. Major Michael J. Keleher (DSWA, Field Command) provided not only programmatic guidance, but also valuable technical suggestions as we all explored the purpose and structure of the CMS. Robert L. DesJardin (SAIC) provided useful programmatic and technical guidance as the project proceeded, and invaluable input on the creation of this report. Thomas E. Smith and Brian C. Clark provided hands-on help and useful "second opinions" to further our efforts to understand various CMS hardware components and make them operable. Kurt W. Derr and Miles A. McQueen performed somewhat similar functions for the software side of the system. We are most grateful for all their contributions.

CONTENTS

ACRONYMS AND ABBREVIATIONS

AAD	*Application Architecture Document*
AASP	Automated Assessment Signal Processing™
AIMS	Authenticated Item Monitoring System
AIMS ASTX	AIMS Authenticated Sensor Transmitter
AIMS RPU	AIMS Receiver Processing Unit
CCM	Configuration Control and Management
CMS	Corral Monitoring System
COTS	Commercial-off-the-shelf
CVR	Center for Verification Research
DAVID	Digital Automated Video Intrusion Detection
DBMS	Database management system
DNA	Defense Nuclear Agency (predecessor to DSWA)
DOE	U. S. Department of Energy
DSWA	Defense Special Weapons Agency
F&OR	Functional and Operational Requirements
FIF	Fractal Image Format
GUI	Graphical user interface
INEEL	Idaho National Engineering and Environmental Laboratory
LDC	Local Data Center
LOW	Local Operator Workstation
MIMS	Modular Integrated Monitoring System
mMCA	mini-Multichannel Analyzer
OID	Object interaction diagram
OMT	Object Modeling Technique
OOA&D	Object oriented analysis and design
OODBMS	Object-oriented database management system
OOP	Object-oriented programming
RAID	Redundant Array of Independent Disk
RDBMS	Relational database management system
RDC	Remote Data Center
RDM	Relational Database Model
rf	radio-frequency
RMS	Remote Service Manager
ROW	Remote Operator Workstation
RSC	Raytheon Service Company
SAIC	Science Applications International Corporation
SIS	System Integration Software
SLTA	Serial-to-LonTalk Adapter
SNM	Special Nuclear Material
SSD	*Summary System Description* (document)
START	Strategic Arms Reduction Treaty
TACT	Testbed for Arms Control Technologies
VCL	Visual Component Library
VMD	Video motion detection, detector

Corral Monitoring System Assessment Results

I. System Background and Description

Functional Background for CMS Proposal

Target Application

According to its *Summary System Description*, the original Corral Monitoring System (CMS) development project was initiated in response to "opportunities" created by the signing of the Strategic Arms Reduction Treaty (START) and the subsequent breakup of the Soviet Union. In the proposers view, "a number of opportunities have emerged for the employment of various on-site monitoring regimes as adjuncts to the periodic or continuous presence of U.S. inspectors at sites within the nuclear weapons states of the former Soviet Union." A key goal would be to monitor "nuclear warhead inventories and by-products of the associated conversion and elimination" activities and processes. "To satisfy a potential need for verification at numerous sites simultaneously, DNA is investigating the possibility of [a] monitoring system with remote reporting and control capability that is supported by minimal periodic maintenance." These needs and desires drove the design for the CMS development effort.

System Overview

The CMS was designed to perform the generic mission of detecting and documenting accountable items entering or leaving a monitored site. It was to be a computer-driven system that would automatically collect, process, and analyze data received from sensors and video cameras positioned around the site. Figure 1, on the following page, illustrates the overall concept.

The outer-most monitoring layer would be an array of intrusion detectors covering the site *boundary*, a concept that is often defined quite explicitly in treaty-related situations. The boundary designation can include not only a description of the location and physical characteristics of the perimeter, but also a definition of the kind of monitoring units to be used. Note that the "monitored site" may or may not encompass an entire facility; it may, in fact, be only a part of a larger physical location. But the essential features of the definition are that boundary crossings are restricted, and activities inside are subject to surveillance. Field elements capable of detecting boundary crossings or intrusion would be placed at the designated boundary. These might include long-range break beam detectors, video motion sensors, passive infrared motions sensors, and others. The Local Operator Workstation (LOW) component of the CMS would collect sensor information with the intent of identifying unauthorized boundary crossings.

The next key part of the system would be monitoring devices at designated site *entry/exit points*; these too are generally clearly and explicitly called out in treaty-covered situations. Sensors and video equipment would collect data at these locations to document vehicle and personnel traffic, authenticate the identity of treaty-limited items, and provide a continuous record of all activities in the area. Like that from the boundary devices, these data would be transmitted to the LOW.

In principle at least, the CMS should also have been able to track the movement of treaty-limited items inside the area defined by the boundary. The *Summary System Description* concept diagram does not show this added monitoring layer, but does mention "*waypoint areas*."

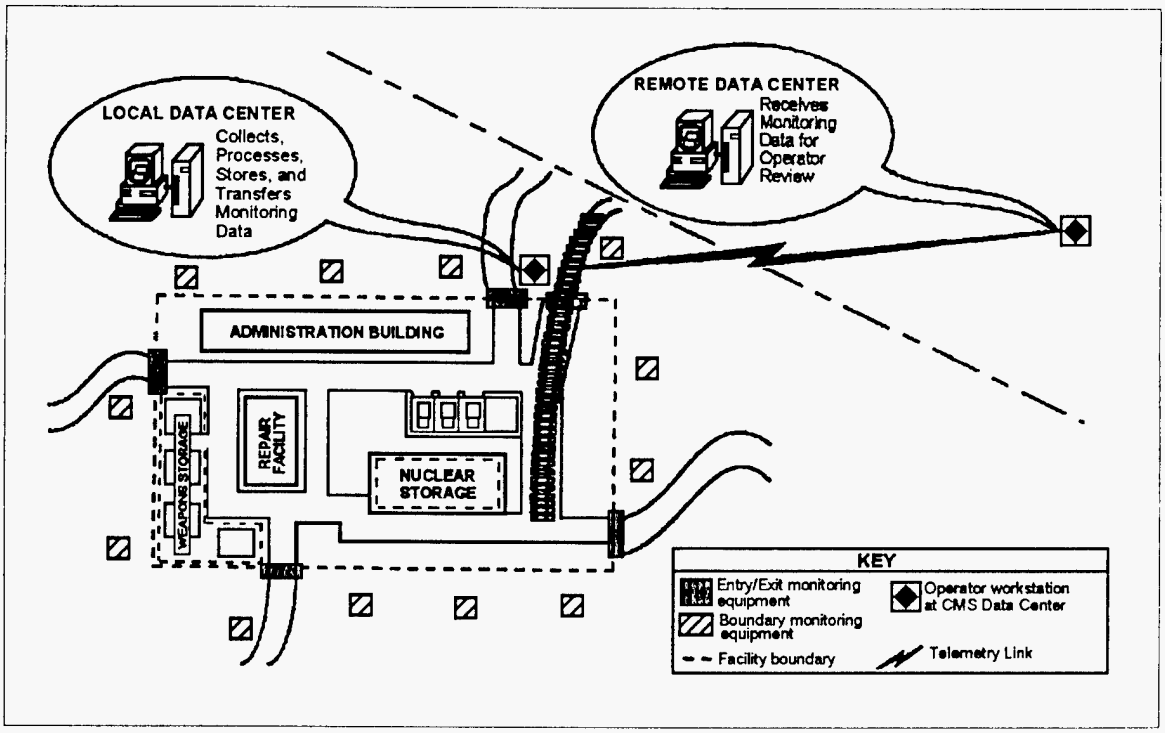


Figure 1. CMS Installation Concept

The LOW was to be housed at a Local Data Center (LDC), which would be located within, or near, the monitored site boundary. Software installed on the LOW would provide automated data collection (including storage of video surveillance images), continuously monitor equipment health, and guard against tampering. Collected data would be loaded into a Master Database. The software suite was meant to provide fully integrated functionality, including a user-friendly operator interface. The final version of the CMS was to have the ability to discriminate an accountable item from its background, and its Automated Assessment Signal Processing™ (AASP) neural network algorithm would train the system to discriminate between legitimate and false alarms for selected intrusion detection sensors.

Other features of the software were intended to allow the operator to review and analyze the collected information. The LOW Master Database would also be periodically synchronized via telephone or satellite telemetry with an identically-structured Slave Database resident in the Remote Operator Workstation (ROW) at the Remote Data Center (RDC).

The RDC was to be situated somewhere away from the monitored site, or sites. In principle, it could be located hundreds or thousands of miles away, and the intent was to be able to handle many monitored sites from one oversight facility. Clearly, ROW software functionality would be primarily for data review and analysis; components for "raw" data collection would not be necessary.

Operational Activities

The CMS was meant to be totally "transparent" to the operators of a monitored site. It would be set up as a completely independent, fully automated, system that would never interfere with normal site activities. In particular, it would not prevent or even hinder any entry or exit. Despite the high level of planned automation, the *Summary System Description* also refers to a *CMS operator*, who would apparently be on-

site during normal working hours. The CMS operator would perform some fairly high level data review functions, but would supposedly not need to know a lot about the technical workings of the system.

Typically, an operator coming on duty would first confirm the CMS state-of-health. The system was to be designed so this information could be acquired automatically, or at the operator's demand. Any system degradation noted would be recorded in a maintenance log database for later reference during periodic maintenance activity. The operator would then manually enter any declaration information for impending transits of treaty items, to be later correlated with the actual transit events.

The operator would be notified regarding any monitoring data that might have been received and stored in a queue in anticipation of operator review. The operator would then examine the monitoring data, typically on a first-in first-out basis. This review process would involve reading textual data and viewing visual data associated with an accountable event. During this process, the operator might need to log assessment responses into the system. The review of routine events was expected to mainly involve confirmation of declaration matches.

Monitoring data associated with events requiring immediate attention, such as undeclared entry/exit transits or boundary breaches, would be subjected to close scrutiny. These events would automatically be queued ahead of groups of routine events awaiting review. Each event data set that had been completely reviewed would then be removed from the operator review queue and archived for later adjudication, if necessary. It would have been entirely possible for a number of events to accumulate before an operator had a chance to review them. This latency would have had no adverse impact, because an event-driven response was not considered necessary in the baseline CMS design. The ultimate goal of the CMS was merely to conclusively document accountable monitored site activity.

The "representative scenario" described in the *Summary System Description* focuses on the arrival of a truck carrying two treaty-covered items at an entrance to the monitored facility. The arrival would be detected by a video motion detector, which would transmit a message to the LOW and trigger the capture of a video frame of the arriving vehicle. The video frame would be displayed on the LDC monitor, while a radio-linked tag partially would confirm the identities of the two items, and radiation data would be collected to verify their isotopic signatures. Further video frames would also be captured and displayed. All these data and video were to be stored at the LDC and transmitted via satellite to the RDC in near real time. In this scenario, the *Remote* operator would analyze this information, confirm that there has been a previous declaration, and verify that the identity of the arriving items matched that provided in the declaration. No mention is made of the Local operator, but presumably he or she would have had to be on site to enter the declaration into the system.

Support Activities

The CMS maintenance concept embodied what the designers called "a worst case scenario" for maintaining the system, which corresponded to the remote operating mode. Routine maintenance visits by circuit-rider teams were expected to be scheduled at 90-day intervals. The term circuit-rider refers to personnel on temporary duty, who would circulate between CMS monitored sites according to a pre-determined schedule. Circuit-riders would typically provide maintenance at the Remote Data Center between site visits, and might perform CMS operator functions there as well.

Activities during a site visit would encompass both corrective and preventative maintenance work. When all corrective and preventive maintenance activity had been completed, the circuit-riders would document their actions and the resulting system status in a maintenance log database for future reference. Changes to the system configuration baseline would also be noted.

Proposed Structure

The design strategy for the CMS included making the maximum possible use of commercial off-the-shelf (COTS) components. These were to be integrated together to create the final system. Figure 2 shows a block diagram of the intended configuration.

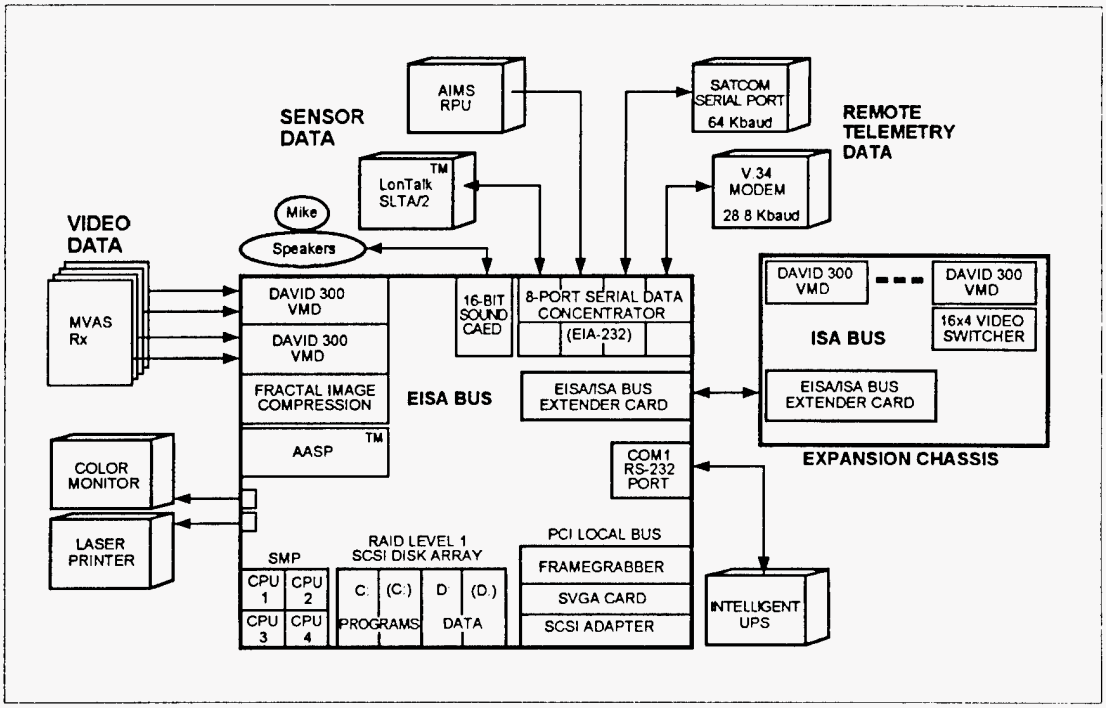


Figure 2. Workstation Block diagram

Basic Workstation Equipment

The CMS hardware design architecture was centered around EISA/PCI "superserver" workstations configured with two 90-MHz Intel® Pentium processors; these were to be used for the Local operator Workstation (LOW) and the Remote Operator Workstation (ROW). These workstations were equipped with 64-megabytes of error checking and correcting random-access memory.

Video Motion Detection The Local Operator Workstation would contain Senstar DAVID[1] 300 video motion detection (VMD) cards to provide vehicle and personnel detection capabilities using cameras located at the entry/exit and boundary areas. Each DAVID 300 ISA-bus card can accommodate two monochrome or composite color video camera inputs. The DAVID 300 digitizes each video frame and contains embedded processing hardware that executes an algorithm that detects motion subject to pre-set parameters and otherwise filters out nuisance alarm sources such as cloud shadows and camera shake. Alarm information is annotated on the video output signal while alarms would be documented in alarm log files on the host hard drive.

[1] Digital Automated Video Intrusion Detection

Video Switching Coreco OCULUS-CS 16 X 4 video multiplexor ISA-bus cards were proposed to minimize the need for many "framegrabber" cards in CMS configurations with large numbers of cameras. Only three 4-input framegrabbers would be needed to handle up to 48 camera channels. Switching camera channels would be done under software control, to capture sequential frames from successive cameras.

Video Framegrabbing The Imaging Technologies PCI-bus video frame-grabber card was proposed to capture video images associated with entry/exit and boundary events; such event-triggered images would be embedded in the Master Database. Each card has four monochrome or composite color inputs. These inputs are compatible with the Photonics Darkstar low-light-level cameras at the CMS Boundary, as well as the Cohu color video camera at the entry/exit area. This framegrabber takes advantage of the 133 MB/sec PCI-bus transfer speed to allow the host workstation CPU(s) to perform the digitizing process.

Fractal Image Compression Fractal Image Format (FIF) compression makes it possible to zoom into an image without the blockiness (pixellation) typical of other approaches. The proposed Fractal Image Compression Accelerator unit was an ISA-bus card that provides fractal transform co-processing to quickly convert framegrabbed camera image bitmaps to .FIF files. This format provides 100:1 or better lossless compression of 8-bit monochrome or 24-bit color images to greatly enhance data transfer and data storage efficiency. Decompression was to be accomplished in software only.

Communications Modules All the local telemetry, including that from the video cameras, was to involve radio-frequency (rf) data links, with extensive use of Authenticated Item Monitoring System (AIMS) components. An rf receiver near the LOW would collect the sensor signals, and forward the data via a serial interface link. The system was designed to use relatively inexpensive portable satellite terminals providing 64-Kbps dial-up access. An external V.34 modem was to be included in the equipment suite. In most cases, serial data interchange with the LOW would take place via an "intelligent" multi-port I/O-mapped serial card (Control RocketPort™), using OS/2® drivers supplied by the vendor. They intended to make use of two other data authentication measures: The Modular Video Authentication System (MVAS) for video links, and the COTS public key encryption program ViaCrypt PGP™ for the satellite data link.

As the project evolved, a data-collection approach based on the use of the LonTalk™ protocol was also included in the design. LonTalk is the specialized operating system at the core of the COTS LonWorks® technology licensed to many vendors by Echelon Corporation (Palo Alto, California). LonWorks provides programmable device controllers that can be linked to communicate among themselves in a classic peer-to-peer network. During the time period when CMS was being developed, rf-linked LonWorks nodes were not available, so a twisted-pair wire link was required to collect the sensor data. The interface between that network and the LOW was to be via a standard Serial-to-LonTalk Adapter (SLTA/2).

Operator Workstation Data processing involved a number of multi-tasked application programs that would run in the Operator Workstations under a version of the IBM OS/2® operating system. This system was proposed because it was said to allow scaleability for up to 16 CPUs along with "inherently seamless" memory management and ability to multi-task DOS and MS-Windows™ programs. All workstation resident application software would be linked into a graphical user interface (GUI) for use by operators and maintenance personnel.

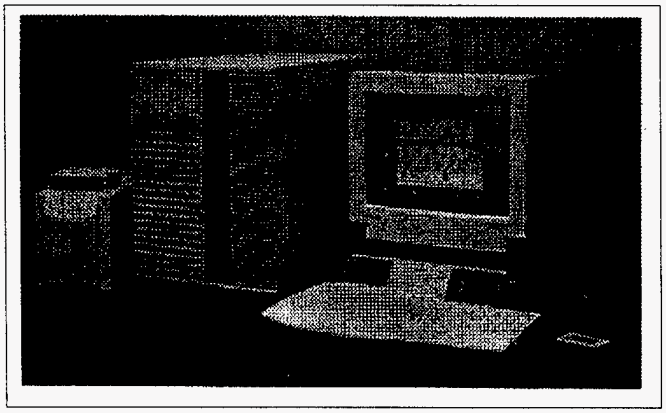


Figure 3. CMS Workstation

Field Sensor Equipment

Entry/exit field sensors would detect the approach of both vehicles and people, screen both for the presence of isotopic material, characterize any material present, read the unique identifier code from tagged items, and collect video snapshots of activity at the entry/exit area.

Radiation Detector The CMS design called for the use of a TSA Systems, Ltd. Model VM-250 drive--through vehicle portal monitor. This detector auto-adjusts to the background radiation level and can provide a contact-closure alarm when it detects very small amounts of radiation, such as that which might leak through item containers. The unit is battery-powered, and can be trickle-charged from AC prime power or from a suitable solar power source. It can also be used to screen personnel passing between the pillars.

Multichannel Analyzer Another sensor included in the CMS plan was the TSA Systems, Ltd. Model mMCA-430 multi-channel analyzer. This unit is a 256-channel analyzer that characterizes isotopic material by scanning to determine the gamma radiation energy spectrum and counting incident neutrons. The unit is battery powered and can store numerous scans in non-volatile memory, as well as transferring scan data via an integral EIA-232 serial port.

Interrogatable Tagging System Items to be tracked by the CMS were to be tagged with the Amtech Corporation Smartpass™. This system utilizes a Model Al1611 integrated rf tag reader operating in the 2.4 GHz band, along with Model AT5510 battery-powered passive rf tags, and the AP4110 PC-based tag programmer. Vendor specifications claim that this system can identify a tagged item at a distance of up to 50 meters within a vehicle moving at up to 10 mph.

Microwave Sensor One unit suggested for use along the boundary of the surveillance area was the M.I.L. PAC version of the Southwest Microwave Model 310B bi-static microwave intrusion detection sensor. This unit is ruggedized and battery-operated, making it suitable for rapid deployment. Its excellent detection performance and low invalid alarm rate were said to make it ideal for this application.

Color Video Camera Video surveillance was to be provided using a Cohu, Inc., Model 8242-1-000/EH06 color CCD camera. This unit is battery powered, using solar cells for recharging. Aside from providing surveillance *per se*, this camera can act as an intrusion detector in conjunction with VMD hardware and software elements located in the LDC. These cameras supplement the low-light level cameras in cases where suitable host-provided area lighting is available for nighttime operation, as would often be the case in an entry/exit area.

Low-Light-Level Camera To provide surveillance coverage under low-light conditions, the Photonic Systems, Ltd. "Darkstar" unit was specified. This unit is battery powered, using solar cells for recharging. No area lighting is required. Like the CCD color camera, this camera would act as a boundary intrusion detector in conjunction with VMD hardware and software elements located in the LDC.

Software Approach

The Operator Workstations would host a software suite centered around custom-designed System Integration Software (SIS) developed with IBM's object-oriented programming (OOP) environment, VisualAge™. This software development tool enables the system integrator to rapidly develop advanced OS/2 client-server applications using a library of Smalltalk objects that are linked using a "graphical" programming paradigm. VisualAge was used to add functionality to a suite of legacy applications that would provide video framegrabbing, image compression, data filtering and analysis functions, and interfaces to diverse SIS data acquisition elements. The SIS application was designed to have a modular structure, to create a multimedia-enabled database server with operator interface and data acquisition client processes.

COTS software components that were to be resident in the CMS workstations included: the OS/2 operating system, the database management system (DBMS) associated with the integration application client-server database; utilities for file and disk management; system tuning and performance monitoring applications; and remote control, task scheduling, and communications privacy software.

Project Background

The CMS units evaluated here at the INEEL were actually a "mixed bag" of hardware and software components going back to a Proof-of-Concept CMS project under the Portal/Perimeter Monitoring System program, contract DNA-001-90C0018. Initial CMS development took place at Raytheon Service Company (RSC) in Burlington, MA, between March 1, 1995 (when software development commenced) and January 31, 1996 (the contract end date).

During the course of the project, interim products were twice demonstrated: One at the 1995 DNA Arms Control Conference in Philadelphia, Pennsylvania in June 1995; the second at the Testbed for Arms Control Technologies (TACT) site in Albuquerque, New Mexico in November 1995. According to the development team leader, the CMS LOW at the TACT site did operate in an integrated manner - capturing DAVID, AIMS and LonWorks sensor data and radiation spectra, and establishing telemetry sessions with the ROW at the Raytheon development lab in Burlington, Massachusetts. The only major caveat was the fact that automatic event-triggered framegrabbing was not possible at that time. In fact, the necessary OS/2 framegrabber driver and libraries did not become available until the very end of the project.

Apparently, CMS development continued until the very end of the contract, allowing little time to carefully document the end status of the project, especially as regards the OS/2-based SIS. This also meant that they had moved beyond the working system demonstrated in Albuquerque, with different members of the team at widely varied stages of their assigned sub-system development tasks. When the projected was terminated, these various pieces were gathered together and set aside in storage.

When this assessment project was initiated in May 1997, a disparate conglomeration of CMS hardware and software components were gathered together and shipped to the INEEL. Much effort was initially expended in trying to define combinations that would actually work together to demonstrate at least some sub-parts of the overall CMS functionality.

II. Assessment Goals and Approach

Assessment Goals

The original *CMS Assessment and Integration* project had two basic goals. The first was to evaluate the operability and functionality of the System, while the second was to assess the feasibility of integrating its capabilities with a "distributed" monitoring and control network sponsored by the U.S. Department of Energy. That distributed network approach is based totally on the LonWorks® technology previously noted in this report. The longer term goal was to combine some or all of the CMS features with those of the DOE approach to obtain a more powerful and versatile integrated system.

However, as early work proceeded, it became clear that many parts of the prototype were incomplete, un-tested, or based on obsolete technology. For example, much of the CMS system-integration computer code existed only in a "rapid prototype" form, and was very poorly documented. Moreover, it was determined early-on that the system should evolve toward a platform that has a greater market presence than OS/2; i.e., Microsoft (MS) Windows NT or Windows 95. The level of effort required to re-assemble,

re-do, revise, upgrade, etc. the existing prototype was judged to be simply too great; and therefore not possible under current fiscal constraints (and perhaps not technically worthwhile anyway).

The focus then returned to the first goal, with the proviso that the project should emphasize an in-depth analysis of the *functionality* provided by various parts of the CMS. This still included making every effort to actually run as much of the system as possible ... to actively demonstrate system, or subsystem, capabilities so the usability and maturity of the pertinent functions could be thoroughly assessed. Understanding how CMS provided these functions and whether they did, or did not, address critical nonproliferation surveillance needs would provide valuable input for future programs in this area. A sub-goal was to try to identify actual components, hardware and program code, that could be salvaged for eventual inclusion in a future monitoring system. Some of the specific topics to be covered by the assessment included:

- The strengths and weaknesses of the System Integration Software, and the Automated Alarm Assessment Process;
- The strength and adaptability of various video-handling features; and
- The maturity and compatibility of its LonWorks functionality and interface.

To make the most of the lessons learned in this CMS functionality assessment, the second (integration) goal was re-cast as an evaluation of COTS "toolkits" that might allow implementation of the same, or similar, features on a workstation attached to a LonWorks network. This comparative feature analysis will be reported later in a separate document: *Corral Monitoring System Feature Comparison.* We expect to be able to recommend a preferred COTS approach when that document is prepared.

Assessment Approach

The assessment proceeded on three general levels: At the highest level, the match between the top-level system approach and the requirements of the target application were reviewed. At the next level, the overall system functionality and development process was examined. The third level examined the individual CMS sub-systems. For some topics, additional information has been included in the report to help put the assessment results in perspective. Thus, a brief discussion of OOP is included in the assessment section for the overall process because the RSC development approach was meant to be object oriented.

At the "macro" level, the assessment covered how well the CMS addressed the target application, and might address similar nonproliferation applications. It also tried to determine what lessons could be learned from how they actually approached their development tasks. And, as mentioned above, the top-level assessment dealt with the broader features of the System, such as the operating system used. Because we were never able to get a complete system up and running, this part of the assessment was based on an analysis of the documentation provided, supplemented by fragmentary data collected for those components that *could be* made operable.

It should be emphasized that the documentation received with the CMS components was not very complete; in particular, no basic Functional and Operational Requirements (F&OR) document was made available to us. At one point during the project, Mr. David Levy (the former project leader for RSC) visited the INEEL to help in understanding the system; he also provided some additional documentation. Yet it soon became clear that some of the documents we *did* receive no longer matched the actual state of the system. This situation created two problems: One, our assessment might very well have covered features or approaches of the project that were, in fact, no longer relevant. They could have already been changed or eliminated by the developers. And, two, there are likely to be items we considered questionable simply because the documentation provided an inadequate rationale for that particular feature or approach (or

provides none at all). As just one example: The "DB2" system was selected as the database development and management approach, but no where does the documentation explain what features made this system a suitable choice for the application. (As we shall see shortly, this selection does have some project-level disadvantages.) This poor documentation meant that some of the system-level judgements could not be very detailed and specific.

On the other hand, at least some of the sub-system assessments could be much more specific. In these cases, we asked: What was the sub-system *supposed* to do? Did it actually work? If we could get that part to run then ... What did the sub-system *actually* do? What problems were encountered? If we could not fully access and exercise a sub-system, we asked: How close were we to getting it to run? Did we know why we had problems?

Whether or not we could get a sub-system to run, we also asked: Did a feature provide a unique or unusual capability? If it did, was the feature really useful? Of course, this meant taking into account how much various steadily-progressing COTS products had changed while the CMS was standing still. Some of the other criteria included: What resources did this sub-system require? Were those resources available in the package received at the INEEL? Were they still available commercially?

III. Assessment Activities and Results

General Features

As noted above, a large and varied array of CMS hardware and software was shipped to the INEEL for this assessment project. Most was transported in the large "work station cases" procured for the original project; Figure 4 shows one of the nine containers sent here. A large box of software was also sent.

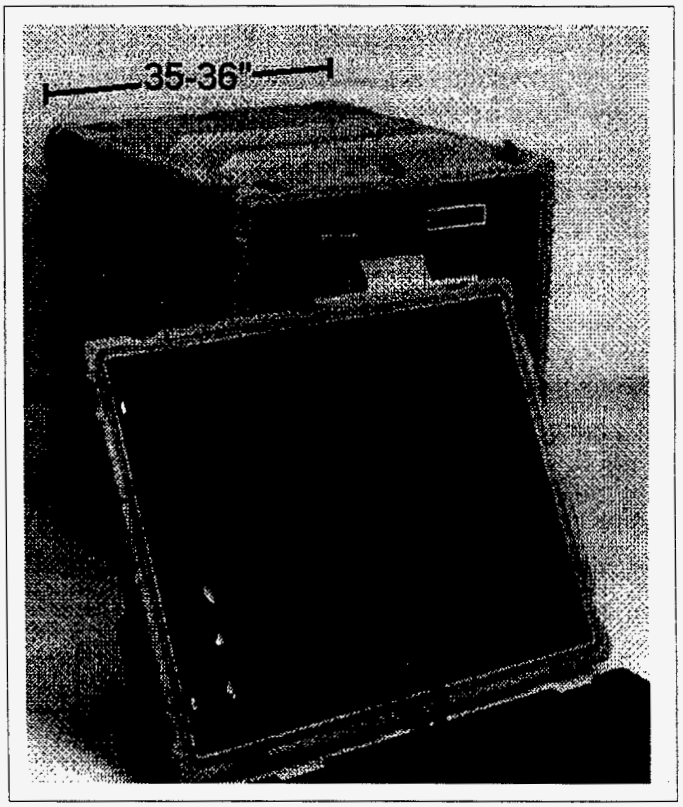


Figure 4. Workstation Shipping Case

Along with two workstations, a total of 16 RAID (Redundant Array of Independent Disk) drives were included; these, in turn, were divided into 19 different mass storage partitions. One of the early problems was to identify which disks were usable boot disks, and which were only for data storage. This was further complicated by the fact that there were just under *150 thousand* separate files located in hundreds of directories dispersed across the 19 partitions. There were over 11 thousand DLLs, nearly 10 thousand .EXE files, and nearly 22 thousand ".FIF" files. While many were duplicates (often in a different directory on the same disk or partition), many were not and there seemed to be nearly a thousand unique .EXE files.

Virtually no documentation was available to identify what these various executables did, or which versions were "current." Of course, the supporting DLLs, bitmap files, and so on have similar problems. By a process of directed trial and error, we were able eventually to find combinations that worked together. Unfortunately, in their rush to complete the prototype, the developers seem to have lost all version control, and were never able to regain it. Worst of all, it appeared that sometimes two or more versions of the same file sets are (in a sense) "right" ... but not right for the same reasons. One set might be the latest and best combination for accepting alarms from the video motion detector (VMD) but have an older set (or even non-functioning "ghost" versions) for getting data from the AIMS interface unit. Conversely, another set might be perfect for the AIMS interface, but be non-functional with the VMD. Thus, combinations such as these worked - after a fashion - in isolation, but could not be combined onto one common disk set.

A supporting goal of our original work package was to create an "after-the-fact" Configuration Control and Management (CCM) infrastructure for the CMS. The intent was to establish proper version control, supported by a standard array of baseline documents for application requirements, system design basis, current configuration, and so on. Unfortunately, it was found that very little of the information needed for these documents was available, and even then it was generally in a non-standard format. It was also clear that some modules were in a state of software "limbo" - we could not find, or no longer had, a prior version that actually worked; but development of a new working version was not yet complete. We concluded that a tremendous level of effort would be required to reverse-engineer enough system knowledge to decide which modules were operable, finish interim versions for software that was "almost there," and fill in the many gaps in the documentation. This expected high level of effort was a major reason the follow-on integration task was eliminated from this project.

Target Application Assessment

As noted earlier, the CMS was intended to detect, identify, and document the movement of accountable items entering or leaving a number of sites monitored as some kind of follow-up to START. Locally collected data would be transmitted from all these sites to the RDC. However, the *Summary System Description (SSD)* actually presented two different views of how the system was to be implemented.

The Operational and Maintenance Concept descriptions listed many activities for an *on-site* CMS operator at the LDC: Checking the state-of-health, local data review, logging data for the maintenance circuit rider, and manual entry of declaration information. The concept description did make it clear that operator coverage would not be continuous. They mentioned "an operator coming on duty" after "a number of events" had accumulated. In contrast, these descriptions made no mention of an operator at the RDC, other than a brief suggestion that the circuit rider might "perform CMS operator functions there" [the RDC]. According to this view, the main focus of the implementation would be the monitored site and the LDC.

The description of the "Representative Scenario" in the back of the *SSD* completely reversed this emphasis. That scenario clearly expected the RDC to be continuously manned, asserting that monitoring data transmitted to the RDC would be "assessed in near real-time." After the assessment, the *remote operator* would prepare a report to document the event data. Although the scenario description said monitoring data would be displayed at the LDC, it made no mention whatsoever of a local CMS operator. It also said the remote operator would compare "the collected data with associated declaration," but never said how the declaration information got into the system in the first place.

Elsewhere, the *SSD* asserted that the monitoring system would be basically invisible to site personnel, and would never interfere with normal site activities. This seemed to support the view presented in the Representative Scenario, but begs the question of how declaration data would be entered into the system.

We were left not really knowing what the developers had in mind. The distinction is important because the system design could have been significantly altered depending upon the actual approach taken.

This point will need to be clarified before requirements are prepared for any possible future system development. At that time, it would be very important to involve "policy" people in the discussion. Ultimately, the approach used at a monitored foreign site is going to depend upon what higher-level negotiators can agree upon. It seems very likely that hard-and-fast requirements cannot be derived in advance of such negotiations. That in itself defines a development requirement: Whatever work is done must preserve enough flexibility so the results are applicable to a variety of situations.

Overall Development Approach

The only reasonably detailed description of the CMS software specifications we received were the *Application Architecture Document* and the *User Interface Design Document* for the proof-of-concept SIS prototype. We also received a *CMS SIS User Guide*, but of course this gave no specific information about the underlying design approach. The *Application Architecture Document* mentioned a set of "CMS Specifications" contained in a *System Description for the Proof of Concept CMS*; however, we did not receive a copy of this document. Our assessment of the software approach was based on the descriptions contained in these documents, a relatively cursory examination of the SmallTalk source code embedded in the VisualAge environment, and our review of the actual implementation.

The assessment highlighted four areas of concern in the overall approach: (1) Poor configuration control and management, (2) inadequate adherence to a well-defined architectural standard, (3) no apparent provision for improving top-level error tolerance, and (4) weaknesses in the OOP approach. Each of these concerns is discussed below.

Configuration Control and Management (CCM) The problem of poor CCM was highlighted above by the discovery of multiple software versions, at different stages of development, on the system disks. The purpose of a CCM system is (obviously) control; but control in the sense of keeping track of what's going on, not in the sense of unduly restricting development. It is especially important for larger and more complex projects, like the CMS, where the development work was separated into sub-tasks that were assigned to different workers.

Effective CCM has three primary benefits for the product being developed. First, it insures that all the pieces of a project will work together. This can be extraordinarily difficult because there are so many ways for a complex system to fail. Programs to operate external devices, drivers for on-board functions, software "hooks" into the user interface, data streams that map onto complex database structures ... all these have potential for incompatible behavior, resource conflicts, and other unpredictable problems.

Second, effective CCM provides the basis for maintaining the working product. That is, as bugs are identified (and there are *always* bugs in complex software), the framework established by the CCM should help diagnose their nature and source. Often, the developmental history captured by effective CCM provides clues as to how the bugs could have "crept in" and suggests preferred approaches for fixing the problems.

Finally, effective CCM is important when changes must be made in the product. Inevitably, complex hardware-software systems *must* change to meet new conditions: Application requirements change, users want new features, hardware becomes obsolete, and on and on. As with changes to fix bugs, changes to meet new needs will benefit greatly from the CCM framework.

Of course, CCM also provides project management and oversight benefits. Clearly, a CCM system should provide information needed to assess progress on the project. But also, as development proceeds, it may become clear that some project goals are unrealizable or need to be modified. Proper CCM provides

the information to make that judgement, and can help guide necessary changes in the project plan itself. Standard CCM methods are well established, so we need not go into great detail here. The core "tools" of CCM are easily listed: Timely review, clear and prompt reporting, and an appropriate mix of basic documentation. "Appropriateness" is important because CCM costs money and time; one must walk a fine line between providing enough control and documentation without over-loading the project budget.

None of the documentation we would normally expect from a formal CCM system was provided to us for the CMS project. This does not necessarily mean the original development project had no formal CCM, but whatever controls were in place were lost when the project was ended. We would have expected CCM documents to be included with the mixed inventory of hardware and software held for demonstrations and a possible project re-start.

Four key items would have benefitted the assessment and integration work. First was an F&OR document, which would have described the features and functions the planners felt were needed for the target application. Second, we needed to know how various versions of the sub-systems and integration software evolved. In fact, we received no version control information. Among the huge number of duplicate files, there was no way to tell which were actually working versions, which were "works in progress," and which might be archives of older versions. Third, although working versions of the CMS prototype were demonstrated twice, no reliable descriptions of the features implemented for those events were provided. Again, this is baseline information we would have expected to be retained in the CCM documentation. Finally, no annotated source code listings were provided. Annotation with extended commentary would help understand how a particular feature was implemented, and why a specific approach was selected. (The assertion that "SmallTalk code is self-documenting" is neither accurate nor useful.)

Architectural Standards Non-Adherence According to the *Summary System Description*, IBM OS/2 was to be the architectural standard for the CMS. We need to emphasize that our concerns here are separate from questions about the specific standard selected. As noted before, early in our program MS Windows was cited as the preferred standard for continued work in this area. OS/2 was judged to have far too small a market presence to compete with Windows NT as a multi-tasking operating platform.

The developers certainly *meant* to stay within their OS/2 standard, and most of the individual component descriptions assumed that would be the case. In practice, however, major deviations from that standard were allowed: Various portions of the system were delivered as DOS applications, MS Windows packages, or different versions of OS/2 software. Yet the *SSD* noted that these programs would "benefit from conversion to 32-bit OS/2 applications." The plan was to run these applications as sub-processes under OS/2, in dedicated windows. As we will see, in at least one case the OS/2 driver used with CMS was a "non-product." The vendor did not support OS/2 and had no plans to ever do so. This mis-match would increase the probability for inter-process conflicts and aggravate the problem of communication between sub-systems. In some cases, the sub-components used in the prototype were themselves prototypes ... development-grade units that were not necessarily ever going to be part of a viable product line.

Based on our discussions with the developers, we understood that these deviations from the architectural standard were allowed in the interest of getting demonstration prototypes up and running. And, basically, this strategy did work. As noted above, CMS was twice demonstrated "in operation." But using non-standard components to provide required features has both a short-term and a long-term cost. In the short run, developer time would be spent to interface the non-standard components, with the almost certain knowledge that the work would need to be re-done when a standard equivalent becomes available.

Two longer-term possibilities then arise, only one of which is "good." The not-bad alternative is that the vendor will indeed provide an OS/2 version of their program. However, this would mean that the CMS would need to be modified to handle the new version. Depending upon the component, this could become

a significant cost. On the other hand, the vendor may *never* convert the application to run under OS/2. Experience shows that the version running in the non-supported environment will soon become an "orphan." From then on, no bug fixes or other maintenance support would be provided, and copies would no longer be available for use with additional CMS installations.

The issues created by non-adherence to a standard impacted two of our other areas of concern: First, the poor CCM implementation would make it very difficult to convert everything in the system to a common standard. This would be even more work if everything had to be converted to run on Windows NT. Second, the presence of non-standard components would make it that much more difficult to improve the top-level error tolerance of the system. Clearly, the more internal boundaries data and instructions must cross, especially at the operating system level, the more chance there was for something to go wrong.

No Error Tolerance Provisions A complex hardware-software product like the CMS has numerous potential sources of error. To begin with, we know that software can have bugs. We hope that the most serious coding errors will be caught during the development cycle. Yet even logical errors – oddities that result from unforseen combinations of code and data – do appear in supposedly finished programs. Then there are entire lists of typical run-time errors: file not found, illegal function call, mathematical overflow, subscript out of range, device time-out, and many more. Other problems appear due to erroneous device or user input. While good programs trap the more predictable input errors, it is very difficult to catch them all. Finally, this system seemed likely to be susceptible to errors in memory management and multi-process resource clashes. This last problem was acknowledged in the *Application Architecture Document*: "Is there a point at which the individual software components start to interfere with each other?" Finding answers to this and related questions was deferred to "future phases" of the project.

Despite this acknowledgment, none of the documents sent to us contain any reference to systematic exception handling or error recovery. The only mention of error messages was among the Uses Cases of the Security Log-On sub-system. (Entry of an invalid user identification or password causes an error message display.) Thus, any instance of fallacious user or device input, bugs in the SIS or legacy software code, or other run-time error could cause major or minor system problems. And, in fact, we experienced numerous system failures (including many complete lockups) as we tried to re-assemble a working combination of components. We realize that the developers, to save on time and cost, might not implement extensive error checking for a demonstration prototype. However, the design documents should at least mention some approach to including such features, even if adding them to the system was to be deferred to a "later phase."

Object Orientation Weaknesses The project documents available to us present a puzzling view with regard to the developer's approach to OOP. On the one hand, they used a pure SmallTalk programming environment, and SmallTalk is renowned as *the* language for object-oriented development. On the other hand, their design seemed to be a curious overlay of procedural thinking onto OOP tools. That is, while they used an OO language, their documentation never really showed any commitment to an OO analysis and design (OOA&D) approach. Some key deficiencies are described below. To help put these remarks in perspective, additional aspects of the OO approach, including its advantages and disadvantages, are described in Appendix A.

First, OOP is about *objects* and *object classes*. In this context, a software *object* is a construct that is meant to represent some item relevant to your "problem domain," i.e., your particular application. A *class* is a groups of objects that have the same attributes, associated operations, and relationships. In OO terminology, objects are *instances* of the class. *Methods* are an integral part of the object. As with procedures and functions in traditional languages, methods are used to make things happen, i.e. perform operations. *Encapsulation* is considered a key generic property of objects. This means that you never

directly access the information hidden within the object, it is accessed only by the methods associated with the object.

The early stages of OOA&D involve converting the features and requirements of the real world application into object classes. Eventually, a design document is generated containing everything the programmers need to represent the problem domain on the computer. As noted in Appendix A, the OOD should include, in some form, three features: An *object model* (usually described by a class diagram), a *dynamic model* showing the time-variant behavior of the system, and a *functional model* to show data transformations made within the system.

Class diagrams describe the basic structure of the objects used to represent the real world application. Class diagrams show the features of individual object classes, and how each class relates to others in the same parent-child hierarchy. *All* OOP systems use class hierarchies, because all OO development environments (including VisualAge) contain standard packages called class libraries. The entire thrust of OOP is to use object classes in these libraries to create a computer representation of the problem domain. Object classes may be used directly, or modified to better meet the application requirement.

Without a doubt, the most complete information in the *Application Architecture Document (AAD)* was in the "Use Cases" parts for each sub-system. This should have provided a good start on creating a complete object model of the application; that approach is widely used in the OO community. Yet that next step was, to put it charitably, very poorly executed. The *AAD* does have a section entitled "Object Model;" however, the material here was quite incomplete, and did not seem to match up very well with the Use Case descriptions. The Object Model section had charts for around 50 objects, while our assessment of the VisualAge source files found around 320 classes specified for the CMS. (The names of these object classes are listed in Appendix B.) The developers did use a reasonably well-defined set of naming conventions to suggest which sub-systems the classes were meant for. The number of classes in each sub-set are roughly as follows:

SL	Security Login	6
UI	User Interface	6
EA	Event Assessment	60
MF	Maintenance Function	2
DM	Data Management	10
DT	Data Transfer	13
DA	Data Acquisition	33
DB	Database	3
SC	System Configuration	38
OM	Object Model	110

Note that the final, and largest, entry referred to the "object model," which was not a sub-system identified in the *AAD*. One might presume that these represented abstract classes that cut across sub-system boundaries, but we had no way to tell without digging deep into the VisualAge code files. Also, there were no classes listed for the final two sub-systems described in the *AAD*: The Scheduler (SH) and Compression/Decompression (CD).

The charts shown in the *AAD* did generally show the attributes and operations for the small subset of objects identified. (One chart is shown Figure 5). The ALL-CAPS notations on the left were presumably "variables" defined for the object, while the mixed-case items on the right were the associated methods.

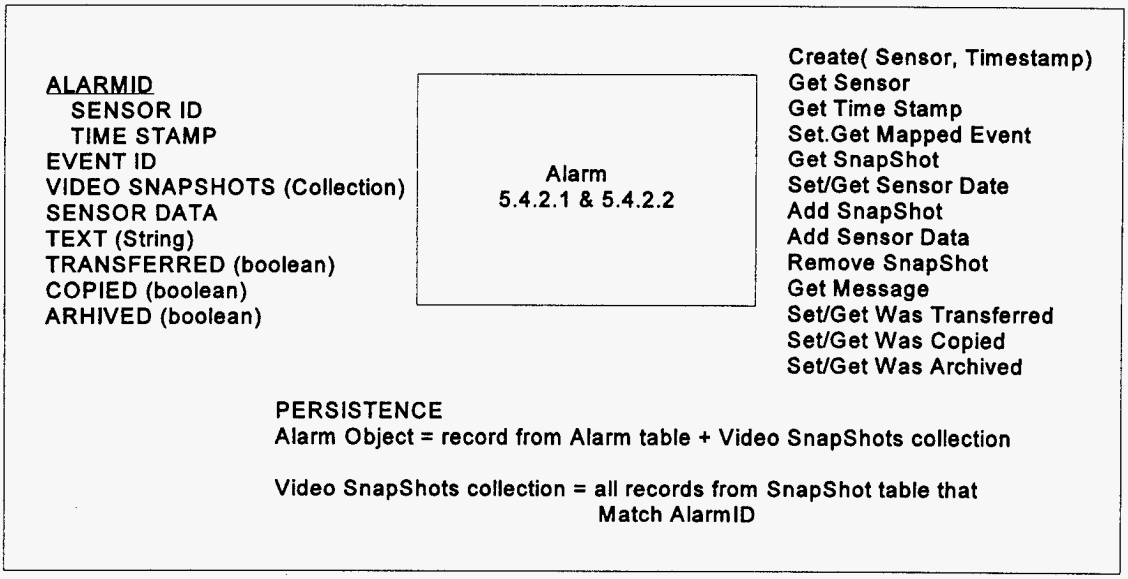


Figure 5. Alarm Object model

The text below the object box did give information about how this entity was connected with some other items in their design. But this by no means constituted a normal class relationship chart. Four of the charts did show a kind of tree structure. However, we could not tell if they were meant to be hierarchical relationship diagrams; there was no hint as to any inheritance of properties down the chart. We could not even be sure from these charts which sub-system, or sub-systems, an object was meant to be used in. Some of the titles and sub-headings provided clues, but it was often very hard to be sure. At least a cross-index to the class names we found and listed in Appendix B should have been provided. (Recall again that our document set was clearly incomplete; this information may have existed somewhere at some time.)

There was also a problem of scope. Some charts were clearly high-level objects: Showing the "Station" and "ROW" and "LOW." Others, like the Alarm Object above, were just as clearly very low level, and some seemed to be mid-level objects. Nowhere did the section show which object might have inherited features from a high level object. As illustrated in Appendix A, a more informative object model (class diagram) should show exactly how classes relate to one another. We would know which classes inherited traits, what traits were inherited but modified by the developer, and which had to be added. From the documentation, we could not tell which objects were adopted from the VisualAge library, and which had to be created more or less from scratch. It is possible, although highly unlikely, that the smaller set of objects shown in the *AAD* were those that the developers had to create, and other activities were all accomplished using standard VisualAge objects. (We do not think this is really the case.)

In the end, it must be said that the object model within the CMS design was simply inadequate to support a project of this scope. It was not *just* that there was no way to tell how well system requirements had been mapped onto the object structure. Because we could not tell which objects did what, there would be no reasonable way to isolate parts of the code for re-use in some other application.

We also could not find any reasonable equivalent of a *functional model* or a *dynamic model* for the CMS. It was hard to tell if any significant computations took place in this system, so it was difficult to judge whether or not the functional model was important. The decompression algorithms for the Fractal Image Format files were probably non-trivial, but those were surely supplied by the vendor. In a system with a strong OOD, these and other major data transformations would be represented in the documentation.

One approach to converting the use cases into a dynamic model would have been to develop object interaction diagrams (OIDs), at least for the non-trivial use cases. An object interaction diagram specifies how the objects communicate with one another to carry out a specific scenario for a use case. Generally, inter-object communications are the time-dependent behavior of the system. No OIDs were found in the documentation provided to us, nor any other information that might have provided comparable content. There were some charts that resembled data flow diagrams, but none of these gave any idea of when, or under what circumstances, such flows were supposed to take place. Actually, if the authors of the *AAD* meant for these diagrams to be data flow diagrams, there were some things missing; e.g., data flows should be labeled so the reader can tell what data is flowing between sources, sinks, and processes.

As a matter of fact, two of the largest parts of the documentation were these (incomplete) data flow diagrams, and a set of functional decomposition charts. Although data flow diagrams may indeed be used during an OOD process, the heavy use of these two formalisms showed a strong bias toward procedural design and development methodology. These points, and other general features of the documentation support our contention that a hybrid procedural design/object encoding approach was being attempted in this project, an approach that would have almost certainly caused problems in the long run. This might also be the basis for the very poor CCM process discussed earlier.

Feature Set Implementation

One feature that stood out in the CMS design was the degree too which functions were embedded in the workstation: On-board video motion analysis, image capture, video switching, image compression, and so on. The distributed control philosophy of the LonWorks approach pushes those functions "out into the field" to reduce the processor load on the central computer, provide more independent security zone functionality, improve the scaleability of the applications, and provide for better localized command and control. "Scaleability" here refers to the range of application sizes the system can readily handle ... from a small security enclosure around a single building up to a large site with many buildings, tanks and sheds, access control areas, internal fences, and the like.

Because practically everything about CMS was embedded in the workstation, everything had to work together properly for us to be able to get any kind of system up and running. This turned out to be very difficult. Some individual equipment items could be operated, but these were not much use. A few of the problems encountered are outlined below.

During his visit, David Levy (former CMS Project Leader for Raytheon Service Company) helped find four drives that could be used successfully as operational drives. These drives were used to try to interact with the DAVID 300 cards. However, although we saw a likely-looking executable ("David.exe") that did seem to activate the DAVID 300 when used directly, it could not be launched from within CMS. The same file appeared elsewhere, but did not seem to work at all from that configuration, nor could it be launched from within CMS. Eventually, by systematic disk swapping, we actually got the system to log video alarms, which then showed up in the CMS transcript window. Even so, there were still problems: For no good reason, debugger windows popped up intermittently with error messages, sometimes the system tried to do framegrabbing, and at other times tried to contact the ROW to establish a communications link. At no time could a reasonably complete array of CMS components be made to work together. Some of the reasons for these difficulties have been identified in prior sections (loss of version control being the primary one). Others may be rooted in problems with individual sub-systems; these will be covered shortly.

Because of the rapid-prototyping mode in which this project was run, the CMS System Integration Software (SIS) was embodied in development and runtime images of the VisualAge for SmallTalk system. Those "images" contained the objects and sub-applications needed for the overall system. Levy noted that

various "team development repositories" were maintained throughout the project cycle. As has already been suggested, the presence of various development versions, at different stages of completion, was what ultimately made it impossible to re-create even the demonstration systems that were run at two stages of the project. The final conclusion was that it would be essentially impossible to now implement and evaluate many individual features planned for the CMS; this conclusion was explored thoroughly in prior sections. The sub-system assessments are reported in the next Section.

Sub-system Breakdown

The sub-systems discussed below are those identified in the *Application Architecture Document (AAD)*. The individual discussions are each divided into three units: The first part is a (usually brief) description of what the sub-system is supposed to do. Next is the assessment itself, covering the apparent status of the sub-system (how mature and usable it seems to be), how well what we received worked, whether or not we think its features might be unique or special, and perhaps some idea of a possible COTS substitute for the capability. The final unit is a general discussion of our experiences with the sub-system, if any. Some areas we spent a great deal of time with, while others were barely touched.

User Interface Sub-system

Description The User Interface sub-system provided a GUI and mouse functions so the user could view the application and navigate in a logical fashion through the other subsystems. The application was started by selecting the VisualAge icon from the OS/2 task bar, using a C: drive configured as a LOW or ROW, and logging on to establish a user profile. The user could branch to available sub-systems by selecting a menu option. An example of the GUI, with the Security LogOn sub-system activated, is shown on the following page.

Assessment The top level CMS user interface seemed to be reasonably complete. Dongle and password check windows were presented successfully to steer the user into the software, and the branches to the various other parts of the system seemed to work properly. Overall, the user interface was neither a major hindrance, nor did it have any unique or exciting features. Had any form of system-level error recovery been implemented, it was within this User Interface we would have expected to find it. This would have helped a great deal during our attempts to run the system. In the final analysis, we considered this sub-system marginally adequate, but not anything special. Today, numerous toolkits are readily available to help build as good, or better GUI screen sets.

Discussion We had no special problems, nor any memorable successes with this part of the system.

Security Log-On Sub-system

Description The Security Logon sub-system controlled access to the underlying application data and functions by creating a user profile, which told the User Interface sub-system what areas a user was authorized to access. Logging on as a guest provided access only to the menu item "Archived Events." Logging on as an authorized operator allowed access to all capabilities except site configuration and user authorization levels. Supervisor authority gave access to all capabilities.

Assessment The Security Logon sub-system worked basically as expected. It was not clear, however, whether or not a security dongle check had actually been performed. There was no visual indication of any kind to show that the check had occurred. As we'll see shortly, we had a lot of trouble with this whole dongle approach. In any case, there was nothing special about the security setup. Much of it was exactly what would be expected from a standard database password protection system. Certainly this is not a unique capability; there are many other good ways that such a system can be implemented

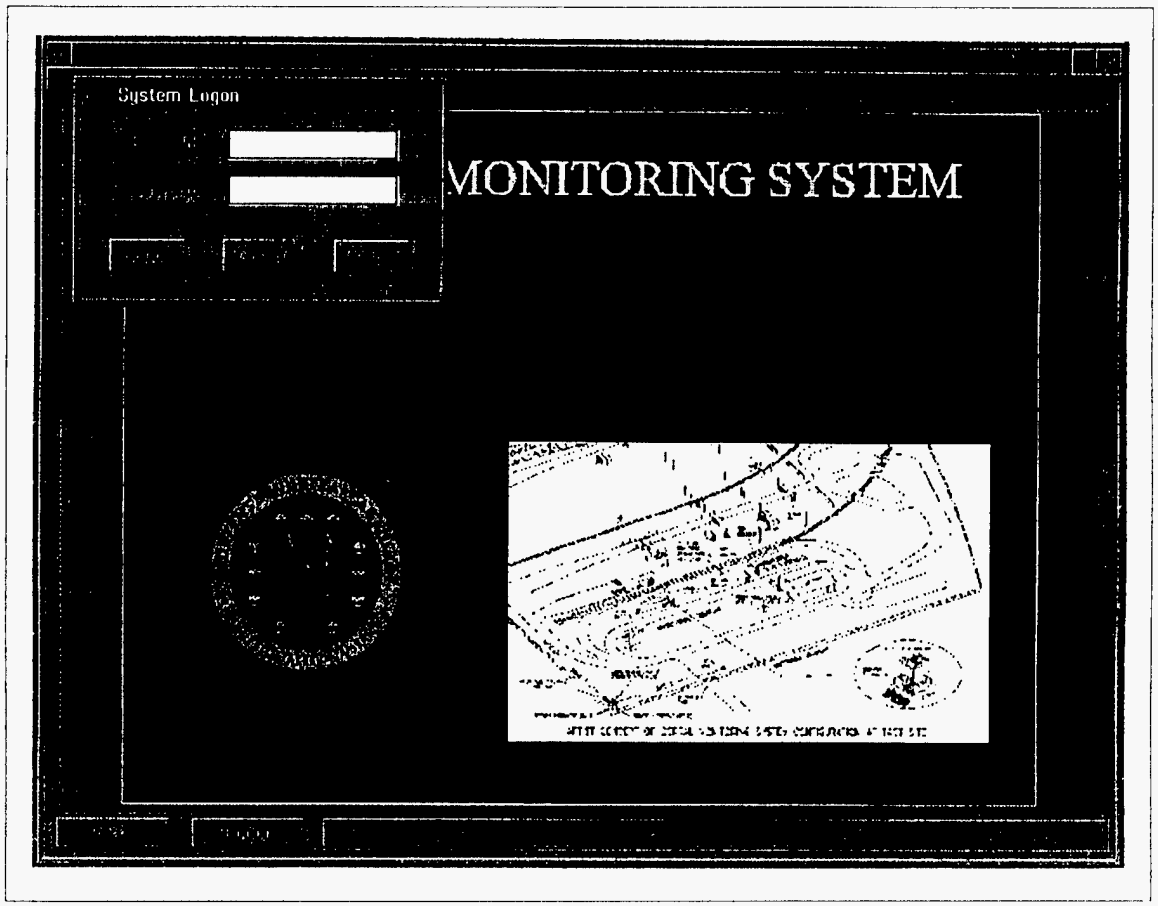


Figure 6. GUI with Security LogOn window

Discussion For a while early in the setup phase, we were unable to get past the user id and password check required by the system. We had received no information on what the current entries were, and had no idea how they might be bypassed. Obviously this was very frustrating, but eventually we (or rather, a representative of the program sponsor) made some good guesses and moved through the security layer. When David Levy visited the INEEL, he added a generic userid/password pair

Eventually we discovered that the User ID and Password requirements associated with the CMS SIS application were actually DB2 database access control features. As such, the user names and passwords were stored in the database, and could be queried using the DB2 Query Manager. It was necessary to know how to log into the Query Manager and how to navigate the database tables in order to find this information for a given system. The most common User ID / Password combination to obtain supervisor-level access turned out to be <Cal / Sup>.

Event Assessment Sub-system

Description The Event Assessment sub-system was designed to give the user the tools to group sensor data into packages called "events," assess what those events might mean, and move the events into various distinct categories. It gave a site administrator or overseer the ability to review the assessed events, and dispose of them, and to manage declarations. It allowed guests to view archived events, and users to focus on specific data.

Assessment This sub-system was considered the "heart of the application." It was meant to display alarm data in a fashion that enhanced the ability of the operator to assign alarms to specific events, and to aid in the subsequent assessment of those events. It was clear from early in the assessment program that this was potentially one of the best features of the CMS. At any reasonably busy facility, monitoring personnel are apt to be overwhelmed by the mass of raw data being logged into the system. Having the ability to summarize and categorize that data, to have help in "making it make sense," would be an extremely useful capability. Unfortunately, we experienced a great many debugger error messages as we worked with this sub-system, and encountered a number of unexplainable lockups and other problems. We therefore had to conclude that this sub-system was simply not a finished product.

This CMS feature was somewhat similar to an "alarm filtering system" developed previously here at the INEEL. We knew the power of such an approach and were looking forward to seeing how the CMS accomplished the function, and were disappointed when it proved to be unavailable. One feature we would like to see include in any future implementation of a similar system is the ability to run algorithms, and other functions, on the data. This should not be difficult to do, and would greatly enhance the usefulness of the feature.

Discussion At one point during the assessment, we were able to make the DAVID 300 system work, and to have the logger record the alarms it generated. We used these as current events to test this sub-system. During assessment of "List Events" from the alarm log window, an error occurred that stated that, "An event has been logged which appears to have no associated alarms". This appeared for each event in the alarm log, and each error had to be recognized by the user before the "Event Log" window appeared.

In another case, we tried to "Accept" of an "Assess Event" menu item but the VisualAge Debugger appeared with another error message. The "Video Snapshots" function under "Details For Events" window was also found to not work as specified. It seemed to work only for the "Alarm Details" window. During "Accept Final Assessment- Archive Event," the debugger window again appeared. The "Declarations" function did not work under the "Details for Event" window. "Reviewing and Associating Declaration", did not work either. Selecting "View Alarm" for the mini-Multichannel Analyzer (mMCA) was supposed to load legacy software to view spectral data, but there was no indication that it did.

System Configuration Sub-system

Description The System Configuration sub-system allowed the site administrator to set workstation defaults, event assessment defaults, user preferences, data management defaults, WAN definitions such as data transfers, remote control, and targets for data transfers. The supervisor could create or maintain a site view and maintain security profiles of users, such as guest, operator, or administrator. It was also meant to provide the functionality to configure individual sensors via sensor legacy software.

Assessment After considerable effort, we found that the only site view and sensor configurations that could actually be used were ones that had been essentially hard-coded onto a working set of disks. A new visual site tree could be generated, such as one for Idaho Falls, but new sensors could not be linked to the database. The only way to generate a new working site configuration was to go into the VisualAge system and modify the SmallTalk code itself. This is not an acceptable way to configure new monitored sites. Clearly, a great deal of work remained to make this sub-system usable.

The *plans* for the System Configuration sub-system looked workable. However, the CMS approach did not really seem to offer any novel ideas. Several COTS packages appear to offer much the same features or capabilities, so this is probably not a fruitful area to consider for any follow-on project.

Discussion When David Levy was at the INEEL, a demonstration was undertaken using the CMS ROW disk, along with the data drive containing the "canned scenario" database, as previously demonstrated at the Center for Verification Research (CVR) in October 1996. After demonstrating the configuration features and navigating the site/sensor tree, a review of "current" monitoring events was undertaken, which involved comparison with activity declarations entered into the database, and examination of associated video snapshots, rf tag ID strings, and actual gamma spectra and neutron counts taken by a mMCA.

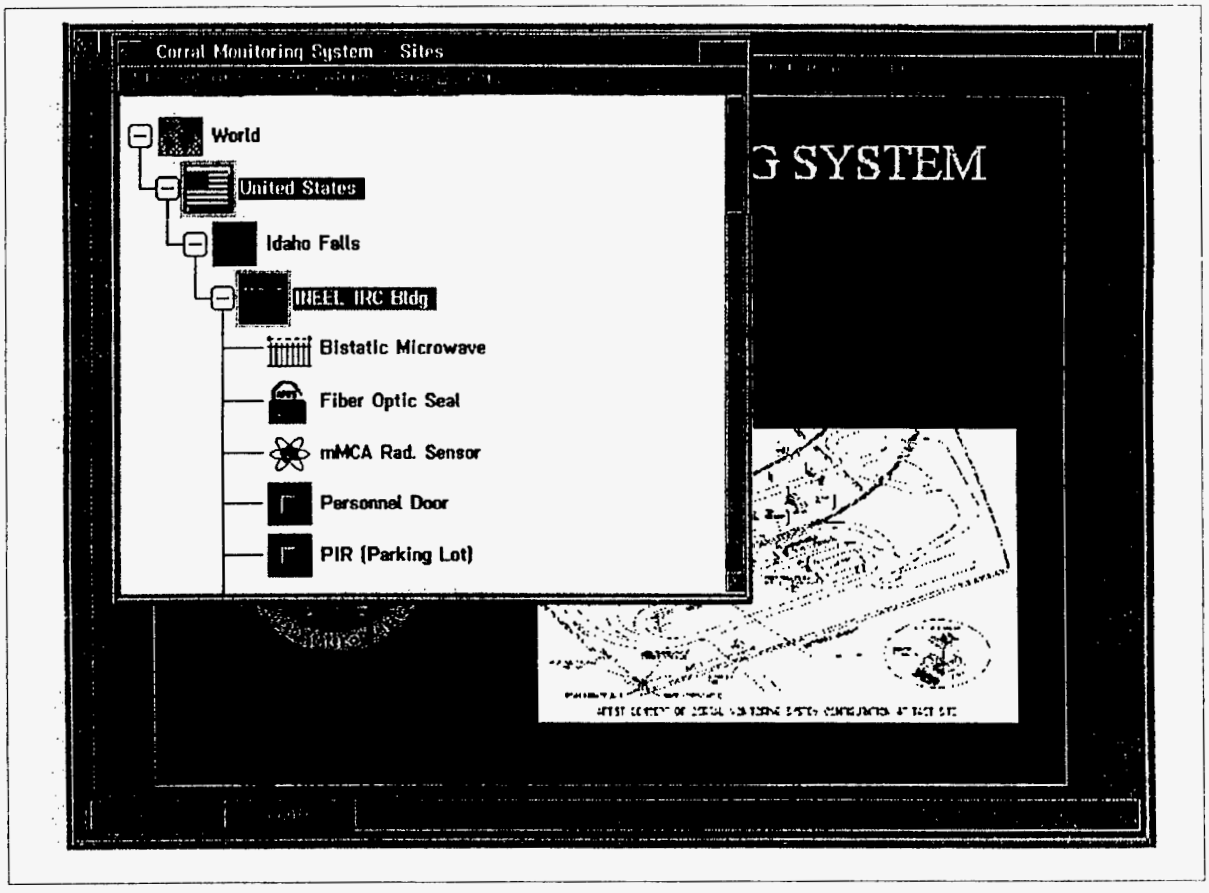


Figure 7. Idaho Falls Site Tree

Levy was also asked to show how to modify the Site Tree configuration to tailor a CMS LOW to monitor data from a suite of sensors to be set up at the INEEL. Using the Configuration sub-system features, a new monitoring location (Idaho Falls) was added to the LOW Site Tree at the same level as Albuquerque. Two monitoring areas (one indoors and one outdoors) were then added at the next level, with a number of different sensor types in each area. Sensors added included door switches and a bi-static microwave sensor (both types annunciated via AIMS), as well as an mMCA radiation analyzer and an rf tag reader.

The DB2 database query manager was then used to modify the database attributes for the new configuration. After the reconfiguration was completed, the new Site Tree for the INEEL monitoring area was displayed, printed out, and subsequently FAXed to the program monitor for review. A correspondingly tailored Map Display capability could have been created using bitmap files that depict the INEEL facility and monitoring areas associated with the CMS demonstration.

In the course of this demonstration, Levy mentioned that a similar process had been used to set up a new monitoring configuration in Alburquerque for the November 1995 CMS Test. Although they had hoped to

perform site-specific tailoring of the CMS monitoring areas using just a simple GUI, this level of configuration transparency had not yet been achieved by the end of the CMS development cycle. Since these modifications were originally performed by the CMS programmers, it was unknown at that time whether revisions to the SIS SmallTalk code itself were required to support these site configuration changes. Later, Levy contacted members of the former CMS programming team and confirmed that such changes were indeed required.

Levy noted that the SIS database on the Albuquerque LOW was never configured to add LonWorks sensors to the runtime image having the corresponding sensor interface code (using the Paragon TNT SmallTalk API). However, the Paragon TNT for OS/2 LonWorks Process I/O (PIO) software module and interfaces were successfully tested on a standalone basis in Albuquerque. Furthermore, the ability to annunciate alarms from a SmartControls LonWorks sensor interface via the CMS SIS was fully demonstrated in the development laboratory toward the end of the project; Levy speculated that this configuration might still be present on one of the LOW drives present at the INEEL at that time. Because the INEEL equipment set did *not* include the Paragon dongle (more later), this was a moot point.

Maintenance Functions Sub-system

Description The Maintenance Functions sub-system allowed the user or supervisor to view sensor "Maintenance History" and "On Demand Video" and confirm the correct camera and sensor operations. This could be done manually or via automatic maintenance data collection.

Assessment Some fairly serious anomalies were observed in trying to run this sub-system, so serious that we were not able to study its operational functionality very much. Based on the design documents, this seemed to be a well-conceived set of functions. However, as with many of the other units, the design did not offer much that was particularly innovative or unusual. A database system built on virtually any COTS platform and designed to capture the equipment maintenance histories could offer similar functions.

Discussion This sub-system "sort of worked," but exhibited some strange abnormalities. The "Sensor Maintenance" screen showed the DAVID 300 alarm as an off-line sensor, when in fact the operator was logging alarms with it. The "On Demand Video," using a DAVID 300 alarm, periodically sent a request to have the framegrabber capture a video snapshot. However, this activity was meaningless because the framegrabber function was not available. The system then displayed a "canned" snapshot called up from disk storage. We were unable to determine why the incorrect sensor status message was displayed. The "On Demand Video" was clearly set up only as part of a pre-configured demonstration event.

Data Transfer Sub-system

Description The Data Transfer sub-system was designed to provide remote communication for data transfers from a LOW to the ROW. Transfers could be either automatic for un-attended workstations, or on demand for attended workstations. Alarm, sensor, and compressed video information were to be the data sent from the LOW to the ROW.

Assessment During our review, we learned that LOW-ROW communications were handled using the third-party OS/2 application called Remote Service Manager (RSM), in conjunction with some CMS-specific customized session scripts. For the equipment shipped to the INEEL, access to the sub-system was not really possible. The custom scripts apparently worked for the developer under carefully controlled conditions, but we had very little success. Since the core software was a COTS package, we decided that these features could be included in any application where we were willing to build custom links. Conversely, the obsolete OS/2-specific links created for this program probably have little to offer for new configurations that might be planned. Although the *Summary System Description* asserted that the LOW-ROW data stream was to be

authenticated, this design feature was not considered among the use cases for this sub-system in the *Application Architecture Document.* It appears that, aside from the AIMS components, the developers were unable to implement data authentication in the working CMS.

Discussion For at least one disk combination, we were able to identify the RMS software on the LOW and ROW, and check for configuration under "System Configuration-Workstation" menu. The ROW was configured with a recognizable phone number, but the LOW did not have that information. This might have been configured had we had modems to test it with. These servers were meant to be used with SatCom links, not modems. A second communications software package, called Talkthru, was present on the LOW and ROW drives. It appeared to us that his software was probably not used, and certainly we had no way to determine its function.

When we forced a "Pull" (a request for data from the ROW to the LOW), a window appeared that stated "Starting PolyPM2." This was, in fact, the communications software. Nothing happened at this point, and it seemed like the software did not load. Similarly, during an alarmed event, the communications software loaded and tried to connect to the ROW. However, without modems or a ROW that would load software, it was impossible to tell whether or not the transfer would have been successful. These attempts probably failed because, despite appearances, we were still not using the right disk combination in the ROW and LOW. This was a general problem throughout the assessment work in trying to operate the various hardware interfaces. According to the developer, RSM can establish communication sessions in a number of different ways. Although "direct connection" was used with SatCom links, LOW-ROW sessions were usually conducted via asynchronous dial-up modems, with or without the use of TCP/IP.

Other SIS images and repositories with new and relatively untested features were present on ROW Drive C:, these had been set up by Dave Levy while he was at CVR to provide a demonstrable CMS populated with canned monitoring data. That demonstration included actual framegrabbed images and radiation analyzer spectra from the Albuquerque site. The new features incorporated in those SIS images included some database modifications, as well as integration of PGP encryption and event filtering enhancements for the LOW to ROW data transfer. However, Levy has also said that these features were never fully tested using CMS workstations. We did note the presence of *PGP for Windows* software on one of our disks, but it did not seem to be operable. Neither the *ViaCrypt PGP* software nor the Modular Video Authentication System described in the *SSD* were on any of the disks we received.

Levy said the data transfers and remote control sessions had been configured to utilize TCP/IP SLIP sessions to improve throughput and reliability, especially for SatCom telemetry. However, this approach required operators at the LOW as well as the ROW to manually launch the necessary software. Plans to develop REXX scripts to completely automate this process could not be implemented before the original CMS development project was terminated.

Data Management Sub-system

Description The Data Management sub-system was used to manage archived data and disk utilization. The sub-system made it possible to delete old un-assigned, un-assessed, assessed, or archived data to make space for newer, more current data. The disk utilization functions determined if a D: drive existed, and if so, it was assigned to be the data disk. However it was done, a D: drive always needed to be installed in the system, whether on a ROW or LOW.

Assessment The only portion of this sub-system that could be tested was the deletion of data using the "Assessment" window, although it did not necessarily process old or assessed data. The disk utilization also seemed to work to the extent that the sub-system knew when a D: drive was not installed. This is a very

important capability to have in a data logging and review workstation. However, it is not at all clear that the CMS design offered any ideas that might be transferable for use in some other application. This opinion might have been different had we been able to utilize more of its intended functionality.

Discussion Actually, the only real indication that the disk utilization feature was working was that the CMS software would not load. The scheduler functions of the Data Management sub-system were apparently controlled by the Legacy "Scheduler" sub-system. We were not able to completely test this subsystem because we lacked a working database of associated alarm "Events." Of course, we lacked a database of events because we could not get that many CMS components to run.

Data Acquisition Sub-system

Description The Data Acquisition sub-system comprised all the interfaces (hardware, and software) needed to collect data from active, passive, and continuous sensors. This interface was responsible for converting the alarm information generated by the individual sensor hardware and software, into data that could be used by the CMS SIS application.

Assessment We spent by far the most effort trying to make the Data Acquisition sub-system work. (Note the long Discussion section that follows.) Unfortunately, these efforts were largely unsuccessful. Only minor pieces of the acquisition components could be made to work, and even then they did not run very reliably or consistently. In the final analysis, the device interface part of the CMS was not something we felt had any promise for future applications.

A combination of several problems seemed to be at the root of this failure. First, despite all the equipment that was shipped to the INEEL, we still lacked a number of key components (like the security dongles). Second, there were clear incompatibilities in the software ... Either the CMS module was incomplete or we had the wrong version, or the software supplied with a component had something wrong with it. And finally, many equipment items came with no documentation whatsoever (no specifications or users manuals, for example). Even when we were able to contact the vendors, we often found that they no longer supported the particular component we had on hand. As for the potential for transferring this CMS functionality to some other project, we judged this to be nil. This highly centralized approach was just too complicated and had too many "show-stopper" problems.

Plans called for the use of a COTS Alarm Assessment Sensor Processor (AASP™) neural network algorithm with CMS, but it was not implemented. In general, the neural network approach is not amenable to the kinds of validation and verification exercises we normally would want to apply to a security-related component. Issues raised by the possible inclusion of such an approach in any future monitoring system like the CMS should be carefully examined before proceeding.

Discussion We had many problems trying to integrate additional components into the CMS sensor suite. Although much of the work moved forward in fits and starts, this section has been organized so as to consider each component area in one block

We received an AIMS Receiver Processing Unit (RPU), but no receiver and no sensors. According to Levy, the internal software for this RPU was a version specifically created by workers at Sandia National Laboratory to work with a certain CMS configuration. It was Levy's opinion that, if we were to replace the RPU with a new receiver/RPU combination, the new setup would almost certainly have the wrong software version for communicating with the CMS workstation. The same version-matching problem might also happen if we tried to interface just a new receiver to the old RPU. Much of this discussion was moot at the time since we had no AIMS-compatible sensors. Later, a search of the storage areas at Kirtland AFB turned up a number of (contact closure) field sensors and associated AIMS Authenticated Sensor Transmitter

(ASTX) packs. These were not shipped to the INEEL, because problems with the RPU firmware would have prevented any further progress in this area anyway.

Although the *SSD* mentioned a bi-static microwave motion sensor, it was only after the search at Kirtland that we received any of this hardware. Levy noted that the CMS demonstration in Albuquerque had preempted the integration, testing, and training of the AASP with the bi-static microwave sensors. The AASP was one of the COTS software systems proposed for inclusion in the final CMS configuration. Because the integration work was not completed, we could not evaluate this component in actual operation.

Trying to make the DAVID 300 VMD boards operate properly was a major source of frustration. During CMS startup, the operator was asked if the DAVID application should be started. Upon a "yes" response, the CMS seemed to load normally. But the top level CMS task window did not show any kind of DAVID 300, or VMD application to be up and running. We did find the file named "David.exe," and were able to cycle out of the CMS process and manually start the application. At that point, it showed up on the task list, and seemed to work fine. But clearly, the DAVID 300 system could not be started from within the CMS. Nevertheless, with a camera and video monitor connected to one of the DAVID cards, we were able to create video motion alarms that were annunciated both via the CMS alarm log display *and* the alarm logging features of the standalone DAVID configuration/logging DOS application.

Sometimes a VisualAge debugger window popped up with an object error that seemed to be related to the VMD. These did not seem to correlate with the video events, so things were perhaps all right. As we continued our assessment, however, we realized that the system was also trying to trigger the (non-existent) framegrabbing hardware, and to contact the ROW for data communications. None of this made any sense, but again we terminated our hardware efforts before we could figure it out.

We received a color framegrabber card, an AM-CLR system from Imaging Technology, with the CMS workstation. Unfortunately, what documentation we had indicated that the CMS was set up to run only with a monochrome video board. After much discussion with the sponsors and Dave Levy, we confirmed that this was indeed the case. In fact, the only combination that appears to have ever worked employed a monochrome (shades of gray) card installed on a daughterboard "on loan from the vendor" and even then they achieved only "partially successful experiments" with it. What appears to be a night time snapshot, obtained from the *User Interface Design Document,* is shown in Figure 8.

We also found that the OS/2 drivers and libraries for the monochrome version were not actual vendor products. They were apparently provided just as a favor to the developers. Levy said the color framegrabber had "never been successfully integrated with CMS." There would have been two major hurdles to overcome to integrate these components: First, the hardware we had was obsolete, so the vendor could not provide any useful help. Second, their upgraded version runs under MS Windows, not OS/2. (The vendor says they do not support OS/2 versions for any of their products, and claim that they never have.)

The Coreco OCULUS-CS was proposed to select banks of camera channels via register-level programming over the EISA bus. At one point we thought we had received the expansion chassis, but the item involved turned out to be something else so we could not test this component. We later learned from Levy that a standalone OS/2 application was successfully developed to test this capability. In this stand-alone mode, successful control of the switcher and RAM image buffers was demonstrated. This made it possible for them to use up to 16 cameras with a single IC-PCI framegrabber card and AM-VS (monochrome) acquisition module. However, Levy also reported that integration of this video switcher with the framegrabber code was never fully tested.

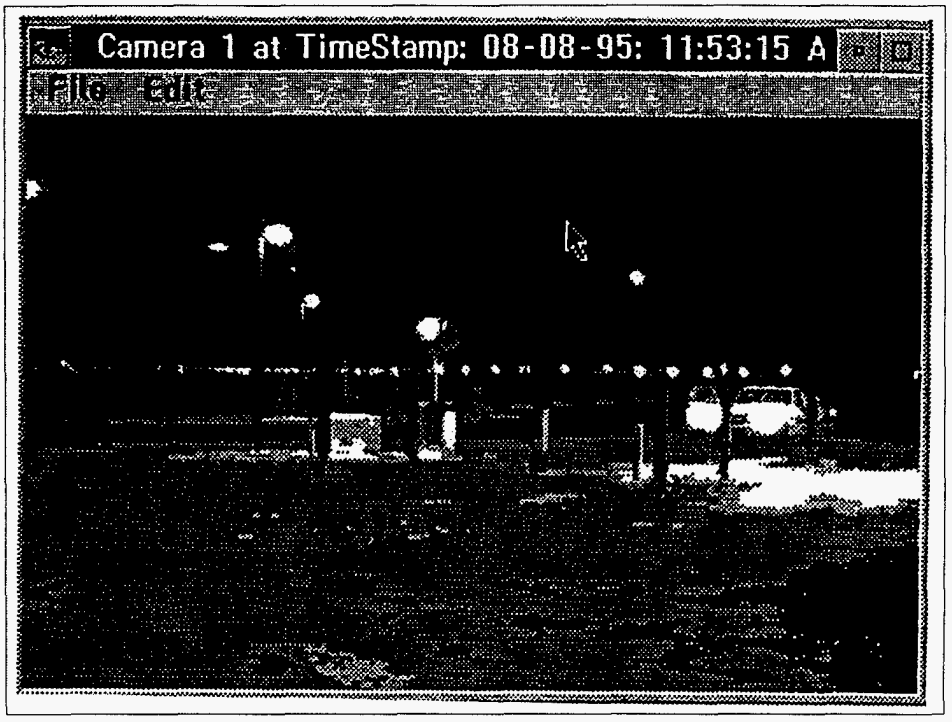


Figure 8. Video Snapshot on Screen

Because of the many problems encountered with the all these video components, essentially all of the video assessment and integration tasks were deleted from the work scope in late August 1997.

The Amtech rf tag reader and tags were two more components discovered in storage at Kirtland. Because of all the other interface problems we were having, we decided it would not be worthwhile to bring this component here for further work. In addition, Levy reported that, during the Albuquerque demonstration, the tag reader occasionally sent back two tag ID numbers when polled only once. They could not determine whether this was a hardware, firmware or software problem.

The mMCA radiation analyzer also became available later in our project. We felt we might be able to run the mMCA, if we could overcome some minor hardware problems, and determine how to run the software. It appeared that we could connect it to a communication port set up with a "commport" configuration. Unfortunately, it also looked like we might need an alarm as a trigger to start the gamma scan. The little documentation we had was not clear (we had no technical specification manual), and Levy was unsure how the system really worked. He did report that the analyzer had continually malfunctioned during the Albuquerque demonstration, despite prior re-work at the factory to correct a known design flaw. Its calibration was also suspect because of the often-strange appearance of the low end of the gamma scans. We terminated most of our hardware studies before these issues could be resolved.

Parallel to the video component work, we also explored the software-hardware situation involving the use of LonWorks devices with the system. To run LonWorks with the CMS, we needed to work through the Paragon Engineering interface screen outside of the CMS. (This seemed rather like the situation with the DAVID card.) However, to run Paragon, we needed to replace the lost copy-protection key or "dongle." While we did eventually work out a way we could have replaced the dongle, there were also other, more serious, problems. To begin with, we could not be sure which combination of disks would allow us to run a LonWorks-connection version. Unfortunately, it appeared that whatever that combination was ... it might not be one where we could also run other pieces of the CMS at the same time. We finally decided that trying to re-link a LonWorks network to the CMS was not worth the effort.

Database Sub-system

Description The Database sub-system provided management of the CMS SIS data files. It was logically represented by a Relational Database Model (RDM), and specifically implemented by the IBM product DB2/2, V1.2.

Assessment The DB/2 database was used during logging of David 300 alarms within CMS. The alarms were viewable as current data, and some assessments of the data were possible. A reasonable range of database functions were tested during our work with the system. We finally concluded that the database software was fairly well integrated into the CMS SIS. However, one of our reviewers asserted that the use of a relational database with an object oriented software product was a fundamental mistake; this mis-match would make the integration more prone to maintenance problems and add to the overall development cost.

Yet there does seem to be some disagreement about this assertion both here at the INEEL, and in the OO and database communities at large. More to the point, we do not feel there was that much to learn from the work done on this part of the CMS. The selected COTS database platform is widely used, and many other actual working applications are available for study.

Discussion The CMS design document contained a relational model (entity relationship diagram) for the application. Actually, the *Application Architecture Document* contained both a prototype RDM and a "projected" RDM. A portion of the projected RDM linkage diagram is reproduced in Figure 9. The use of such models is is a standard way of approach the design of a relational database management system (RDBMS). However, many people in the 00 community feel that the use of an RDBMS with an object-oriented language increases the project cost and schedule 10% to 30%. This is because the application must provide code to map tables to objects and vice versa. They feel that a more optimum solution is to use an object-oriented database management system (OODBMS). An OODBMS is a persistent store of objects created by an OOP language. The OODBMS allows objects to persist beyond the confines of program execution. The use of an OODBMS with an OOP is relatively transparent.

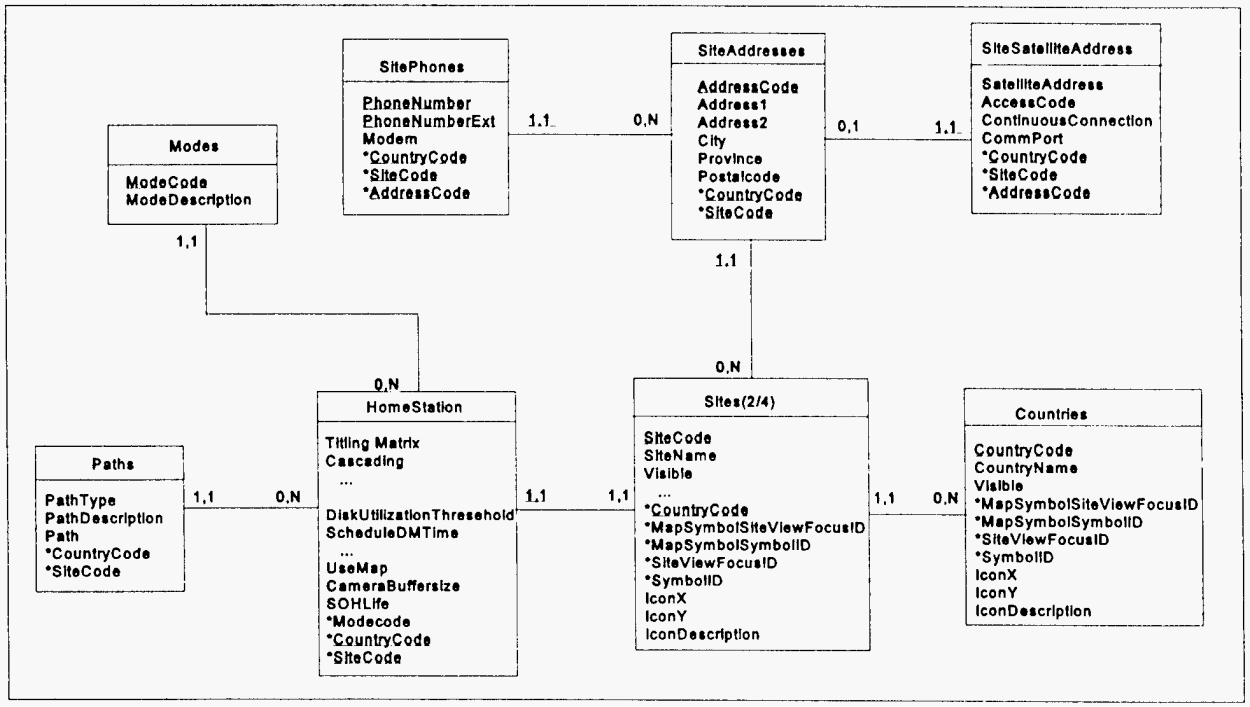


Figure 9. Portion of Relational Data Model

According to Levy, the interface between the CMS database and the SIS application was subject to modification and fine-tuning up to the very end of the CMS development cycle, partially as a result of a DB2 upgrade. Some database versions were consequently not compatible with the SIS, and would result in "empty" datasets when viewed via the SIS user interface. We think we encountered such a situation several times as we worked our way though the database-handling menus. Apparently, the "correct" (i.e., least risky) reply to the "Install/Abort/Continue" query in the user interface was usually "Continue." We later learned that the "mixed bag" of removable hard drives we had almost certainly contained several different "canned" datasets and CMS SIS development images.

Legacy Sub-system - Scheduler

Description The Legacy Scheduler sub-system provided a means of scheduling background processes such as data transfers from an unattended LOW, archived data management requests, and to control automatic scheduling. According to what documentation we had, the COTS package called CHRON was being used for the CMS scheduler.

Assessment Basically, it was impossible to determine if the Scheduler sub-system actually worked, or was even a part of the CMS SIS. With no documentation beyond the name of the product, it was impossible to assess this part of the design.

Discussion We found that the CHRON software was indeed present on the CMS LOW, but it could not be started from the Filestar utility program, or executed using any of the CMS menu options. No further effort was expended in this area.

Legacy Sub-system - Compression/Decompression

Description The Compression/Decompression sub-system was used to compress video snapshots for reduced use of disk space, and to provide shorter data transfer times. The compressed video could then be decompressed later for viewing, from either the LOW or ROW. Apparently a COTS component from a company called Iterated Systems was to be used to provide this capability

Assessment As it turned out, we had no way to assess this part of the CMS. Anyone who has a need for this capability must evaluate the unit on its own merits, without regard to any interactions with other planned features of the CMS.

Discussion Iterated Systems COTS hardware did exist on the LOW, and software was present on the LOW L1 drive as a Windows/OS2 software package. We could not test to see if it was part of the CMS SIS. Trying to run the compression software in a stand-alone mode caused the system to lock up, so we had to reboot. The ROW server contained no Iterated Systems hardware. Actually, without framegrabbing capabilities, the compression/decompression hardware and software was useless because there was no video input to compress.

IV. CONCLUSIONS

The earliest conclusion reached in this effort led to termination of the original second phase of the project. That is, the sponsor agreed with our determination that the level of effort required to prepare the CMS prototype for integration with a distributed network was simply too great. That led to a re-direction to focus on the features and functionality of the system.

Using mostly the *Summary System Description (SSD)* report as a basis, we assessed how well the CMS addressed the target application, and might address other nonproliferation applications. We concluded that the *SSD* description was adequate as a broad perspective, but it fell short in mapping those broad requirements into a coherent operational concept. For example, we could not determine their preferred or expected site manning approach. In one section they emphasized the presence of an operator at the monitored site, while in another they seemed to assume that the most critical functions would be performed at the Remote Data Center. We were left not really knowing what the developers had in mind. The lesson to be learned here is that flexibility, with a broad "shopping list" of capabilities, is probably more important than any one specific feature for this kind of system.

At the next assessment level, our intent was to determine what lessons could be learned from how they approached their development tasks. Unfortunately, these lessons were taught more by what was *not* done than vice versa. We concluded that the weaknesses in their overall development approach were all directly or indirectly related to the "rapid-prototyping" mode in which the original development project was run. One indirect consequence was that little or no provision was made for trapping out errors or other exceptions. None of the design documents contained any reference to systematic exception handling or error recovery. All too often, the only way we could recover from an error was to re-boot the computer. This was not only frustrating and time-consuming, it increased the risk that data and program files might become corrupted or be lost. The lesson to be learned here is that at least a top-level error over-ride or restart function should be implemented early in a programming project.

The other three weaknesses were inter-related: The absence of a formal Configuration Control and Management (CCM) system, failure to stick with a standard system environment, and apparent use of a hybrid procedural design/object oriented encoding approach. Basically, none of the documentation we would normally expect from a formal CCM system was provided: No formal Functional and Operational Requirements (F&OR) document, and no history files to show how various versions of the sub-systems and integration software evolved to the state in which we received them. Worse yet, no annotated source code listings were provided to explain how specific features were implemented, and why. Finally, although two separate demonstrations had been presented as formal events, no reliable descriptions of the system states during those activities were provided.

The problem of poor version control and documentation was compounded by the failure to adhere to their selected IBM OS/2 operating environment. No only would they have now needed to track versions of their own software, they would also have needed to stay current as vendors changed how they supported COTS components intended for any of three different operating systems. Of course, the use of multiple platforms increased the probability for inter-process conflicts and aggravated the problem of communication between sub-systems. This combination would create a kind of negative feedback loop. The use of multiple environments would make it much more difficult to maintain accurate CCM documents, and poor CCM documents would make it very difficult to evolve everything toward a common standard. Of course, it seems likely that they never had to deal with these extra problems because no formal documentation was attempted, and they never tried to pull everything together onto one operating platform.

We found that the hybrid procedural design/object oriented encoding approach led to a CMS design package that was simply inadequate to support a project of this scope. It was not just that there was no way to tell how well system requirements had been mapped onto the object structure. The problem was also that, because we could not tell which objects did what, there was no reasonable way to isolate parts of the code for re-use in some other application. (This was particularly unfortunate because one of the claimed strengths of the OO approach is the supposed ability to readily re-use code.) This mixed approach to analysis, design, and coding also almost certainly contributed to the general lack of formal CCM structure.

Several lessons can be (re-)learned from these problems. First, selecting a standard, or set of standards, for a complex project will simplify the documentation, shorten the learning curve as new components are added, and reduce the potential for component incompatibilities. Second, an appropriate CCM approach should be established early in a project. It can then be allowed to evolve as the project proceeds and needs change. Finally, the choice of standards should also include a reasoned selection of a coherent approach to analysis, design, and coding. Whatever approach is selected, it should feed into, and benefit from, the structure of the CCM system. One caveat: It is not at all clear how to best adapt a formal CCM system to the currently preferred approach to OOP, which involves rapid prototyping in short iterative cycles.

The final assessment level was directed at the eleven individual CMS sub-systems. In general, this had to be done in a rather piecemeal fashion because so few of the components were found to be operable, much less being able to run as integrated packages. Far and away the most effort was expended in trying to make parts of the *Data Acquisition* sub-system work. Conversely, very little (or no) work was done with the *Maintenance, Scheduler*, and *Compression/Decompression* sub-systems, while a moderate amount was done with the *User Interface, System Configuration* and *Data Transfer* sub-systems.

Only two components seemed to run adequately and consistently: the *User Interface* and *Database* sub-systems. The software parts of the *Security Log-On* sub-system ran all right, but the hardware security devices were either missing or did not work properly. Unfortunately, the other sub-systems were mostly unusable. We encountered numerous bugs (some caused the computer to lock up), serious and unexplainable anomalies in behavior, or inconsistent and unreliable operation. Sometimes a sub-system would work one day, and then fail the next time it was accessed ... even though there had been no apparent change in the system setup. The *System Configuration* sub-system had a special problem. It seemed to run, marginally, but required changes in the actual source code to perform its stated function. This was not an acceptable approach, particularly with the poor state of the documentation. Ultimately, we had to base the sub-system assessments on fragments of observed operating behavior and a reading of what documentation we did have.

We found that most of these sub-systems really offered very few features or capabilities that were new or unique, even at the conceptual level. For example, the *Security Log-On* sub-system pretty much used a standard approach, even though it included checks of some hardware-based security devices. Similarly, the *User Interface* sub-system was a conventional GUI, one that could easily be duplicated today using any one of several readily available "toolkits." The *Event Assessment* sub-system seemed to have possibilities, but it was simply not a finished product, so we were unable to assess its actual operation. The *Data Acquisition* sub-system was hampered by its highly centralized approach and unreliable operation.

Our observations yielded three broad conclusions: First, the match between the system design and the target application was not very good; several issues needed to be evaluated more thoroughly. Second, weaknesses in the overall design and development approach would almost certainly have caused significant problems had the original project continued, and played a major role in crippling our re-start attempts. Finally, at the basic functional level, we concluded that the system had no special features that would be worth extracting for use in another system. A completed CMS might have offered a unique *combination* of features, but this level of integration was never realized.

APPENDIX A - DISCUSSION OF OBJECT ORIENTED PROGRAMMING

The purpose of this Appendix is to describe some general features of object oriented analysis, design, and programming (OOA, OOD, OOP). As noted in the report body, OOP works with *object classes*, which have attributes, operations, and relationships. Properly designed objects provide *encapsulation*, hiding features and methods from external access. Another often-mentioned OO concept is that objects communicate only by sending and receiving *messages*. This terminology is presumably used to emphasize the need for effective encapsulation. In practice, however, object messages look and act basically the same as function and procedure calls in more traditional language systems.

Two other key concepts of OO are *inheritance* and *polymorphism*. Inheritance occurs when a new ("child") class is created by adding new features to those of an existing class ("parent"). All OOP systems allow many layers of inheritance, potentially leading to complex hierarchies of classes. Some language systems allow for multiple inheritance, in which the child inherits features from more than one parent class. Polymorphism occurs when a new class is derived from an existing class by over-riding (internally re-defining) one or more inherited traits. It is entirely possible to build a new class by adding new features and over-riding some inherited features. These two capabilities, along with encapsulation, are critical to the success of the object oriented approach.

Advantages and Disadvantages of OO

Numerous advantages are claimed for OO compared to traditional approaches. First, OOA and OOD are said to be a "more natural" way of looking at the world. Second, the use of OO supposedly improves the communication process, both within the development group and between the developers and the client. Third, OO code is said to be highly "re-usable" in the sense that existing modules can be used to provide comparable functions in a new project. Finally, it is asserted that OO leads to better-quality software: Programs that work better, with code that is easier to maintain and more readily modified. There appears to be some validity to these assertions, yet the situation is not nearly as positive as early proponents hoped.

To begin with, the "naturalness" of viewing the world as a collection of objects is not supported by any empirical data, nor even by any reliable anecdotal information. Plus, even OO advocates agree that applying it to "ill-structured" or poorly defined problem areas is difficult and time-consuming. Moreover, practitioners agree that, ultimately, what seems natural to you may not necessarily look natural to me. The community obviously agrees that OO has numerous advantages, but a general feeling seems to be developing that being a universally "natural" way of looking at the world is not one of them.

The assertion that an OO approach "improves communications" is supposedly bolstered by case studies reported in the computer press. It does seem likely that communications within the development team could be improved. This is especially true if, as is normally the case, everyone on the team uses the same development platform with a standard "class library" (set of pre-defined objects). This *enforced* standardization makes intra-developer agreement more workable and likely. A degree of standardization could be enforced on traditional platforms, but their stock of pre-defined functions and procedures is often much more limited.

The belief that OO also improves developer-client communications is less plausible. With suitable coaching, a developer and the client may indeed be able to "think in objects." The question is: Are they the same objects? There is compelling evidence in other case studies that, in fact, they are not. The client almost always thinks in terms of the "narrative" objects describing the real world features. Developers see the

abstract computer objects that are meant to correspond to the descriptive objects. This is not necessarily a problem. In fact, it's how the system is *supposed* to work, if the mapping between the two object views is valid. Unfortunately, experience shows that several, sometimes many, revisions of the computer objects may be required before the correct mapping is accomplished. (We'll re-visit this point in a moment.)

Code re-use is the most cited benefit of OOP, with some justification. Once an object has been designed, coded, and tested for a particular feature set, it can be used and re-used whenever those features are needed in a new application. Encapsulation means there are no unexpected interactions for the object; you know what the object can do and exactly how you must go about doing it. Of course, older programming language also re-use code; the extensive libraries associated with FORTRAN are just one example. Inheritance and polymorphism significantly enhance this advantage in OOP by providing a structured means for modifying existing, almost-ready objects for a new application. Faster development is often claimed as a secondary benefit from this approach, and there are certainly examples where this is true.

On the other hand, code re-use can be over-done or mis-used. That is, the idea of code re-usability can lead developers away from performing a thorough systems analysis so they can devise classes that efficiently and effectively address the application. Given the normal "deliver-a-product" drive in development, it seems *more likely than not* that the developer will grab the first class object that seems to meet a need, or that looks like it can easily be modified to meet the need. Will it be the best way to meet that need? Perhaps, but in the real world of program production, people will simply move on to the next problem. We may never know any different unless problems appear later on. Programs generated when this happens tend to be bloated and inefficient. The worst case occurs when the developer subtly or drastically *modifies the application* to fit into an existing set of classes. The product no longer accurately represents the initial problem domain. While a good testing phase should catch such alterations, there are cases where the client went along anyway – just to get a product out the door.

Possible problems such as these cast doubt on the claim that OO, *but its nature*, leads to the production of better-quality software. First, it is by no means automatic that OO programs will "work better" than those produced by other approaches. In fact, if "use system resources most efficiently" is the criteria, there is good reason to believe that OO program will not work as well. Several case studies have demonstrated that real-time applications are hampered by performance problems inherent to the OO approach. Because OOP forces encapsulation, object oriented code may indeed be more maintainable than would otherwise be the case. Similarly, inheritance and polymorphism should, in principle, make it easier to modify OO code. Nevertheless, poorly-designed OOP can still be hard to maintain and modify, while proper design can make it easier to maintain and modify *any* computer program.

Above, we noted the fact that some of the claimed benefits of the OO approach are not actually realized, or are less clear-cut than proponents would hope. Besides these, the approach has some clear problems. To begin with, its weakness in handling ill-structured or poorly-defined systems seems to be a *general* problem. It can be much more difficult to find re-usable object classes that apply to such situations, and they significantly heighten communications problems between the client and developer. More specifically, case studies have shown that OO systems seem to have no reliable way to handle "concurrency" (when several real-world activities happen simultaneously). Also, several facets of the OO approach can hamper performance – sometimes quite severely, as in the case of real time systems.

Finally, at the project level, the OO approach has two known weaknesses. First, we know from personal experience that the "learning curve" is quite steep for understanding and adopting OO methods. Second, there is still seems to be no generally-accepted process for project-level management of OOA, OOD, and OOP. Several different methods are used, most of which are quite different in philosophy from approaches used in the procedural/functional world. One OO author has even asserted that, "traditional Life Cycle approaches" are "un-usable" for OO projects.

Overall Design Philosophy

Of course, at the broadest level, analysis and design are still analysis and design. They proceed through the usual general steps: Establish requirements and model the behavior of the application, design computer constructs that will meet those requirements, and implement the design in program code. Two behavioral patterns distinguish the OO implementation of these broad strokes from non-OO approaches. The first difference should be obvious: Features and behavior are modeled in terms of *objects* rather than functions, procedures, and data flows. As it turns out, the second is also highly characteristic. Virtually without exception, leaders in the OO field advocate an incremental, iterative approach to the entire process. They see the process as "evolutionary" and "situational" in nature (whatever that means).

One of the most influential authors, Grady Booch[2], promotes what has been called a "round-trip" iterative approach. That is, select a piece of the total project, quickly analyze it to build what appears to be a workable design, hack some code to implement the sub-system, and then test it. At this point, you basically fold the analysis of the outcome back into the original analysis, and re-design it to address any flaws. The cycle continues until the computer sub-system meets the requirements of the real-world application. Booch does say that an overall system perspective must be kept in mind, but also advocates running the cycle with very small chunks (according to one report, his software-development company does *weekly* internal "releases"). Other leaders in the field differ in the details and scale of the work, but their general design philosophy is basically the same.

Note how this cyclic analyze-design-program approach relates to the client-developer communication problem discussed above. In any complex problem domain, its seems almost certain that these two will not start out understanding each other. After all, the client is an expert on his or her real world domain, but ignorant about complicated class libraries and other OO "stuff." Conversely, the developer knows the computer stuff, but is ignorant about the client's application area. Particularly for ill-structured problems, the probability seems pretty small that the "first cut" object definitions will actually meet all the client's requirements. The iterative cycle makes it possible to show the client some results and ask, "Is this really what you want? Does it do what you need it to do?" Given enough time and effort on both sides, this should, sooner or later, converge on a mutually-agreeable solution.

A variety of design tools have been suggested to aid the process, some having their own notation and detailed sequence. One approach is known as the Object Modeling Technique (OMT). The OMT methodology uses three complementary models to express aspects of a problem and its solution - object model, dynamic model, and functional model. Each model, incomplete by itself, presents a different perspective of the system. The models combine to really describe a system. The *object model* characterizes the static structure of the system. A class diagram is used to show the classes, their attributes and operations, and their relationships to other classes. The *dynamic model* represents the time-dependent behavior of the system and the objects in it. State transition diagrams or Harel state charts are the most common approaches used to describe the dynamic behavior of the objects in the system. The *functional model* defines the computations that objects perform - how output values are computed from input values. Data flow diagrams which consist of processes, data flows, actors, and data stores are used to show how input data values are transformed into output data values. Whether a developer uses this specific model set or not, his or her approach must address the same issues.

---

[2] Grady Booch, *Object-Oriented Analysis and Design with Applications*, 2nd edition, Benjamin Cummings, Redwood City, CA (1993). ISBN 0-8053-5340-2

Example Sequence

The early analysis part of the sequence is not really unique to OO:

- Define the real world situation the program is expected to handle. The more clearly defined and reasonably bounded a scenario is, the greater the likelihood of a successful project.
- Define the major interfaces within the system, and with situations on the outside. Often the user interface is the most important of these.
- Define the abstract concepts the computer program will be expected to capture. Modeling the opening of a door should be easy, but you must be much more careful when modeling the approval process that allows a specific person to open the door.

Defining the object structure is a two-step problem. First, you must devise an object-based model of your real-world problem. Describing physically distinct elements as objects is not as easy as it may sound, and less concrete situations can be even more difficult. Moreover, a desire for OO efficiency dictates that common features of these lower-level objects should be combined into abstract "super-classes;" this is by no means an easy task. At this point, the objects will probably be described in a concise narrative form, perhaps as a table of descriptive names and features. Once we have a "book" of requirements, physical and abstract components, and interface descriptions, we can really begin the object modeling sequence.

Suppose we want to model the door situation suggested above. Our overall system perspective is that the door is one of many access control points at a large facility. Components include an electronic door lock with a personnel identification system, perhaps a keypad with small display. The keypad/display is our user interface. This equipment must interface to a central monitoring station, which downloads authorization data to access points, and collects site-wide entry data. For this overview, this is all of the system "book" we will use. Our brief summary continues from here with the object-building sequence.

We will treat the central station as a "black box," defining no more of it than we have to. The object, or objects, we end up with must model the entire access authorization process, not just the opening of the lock. Our narrative summary would be:

Physical components: Keypad/display, electronic door lock, link to central station.
Operations: Update authorized-personnel list, show instructions, accept user input, compare to authorized-entry list, allow or deny entry, inform central station, (conditional) unlock door.
Interfaces: Data link to central station, keypad/display for user input-output, link to lock mechanism.

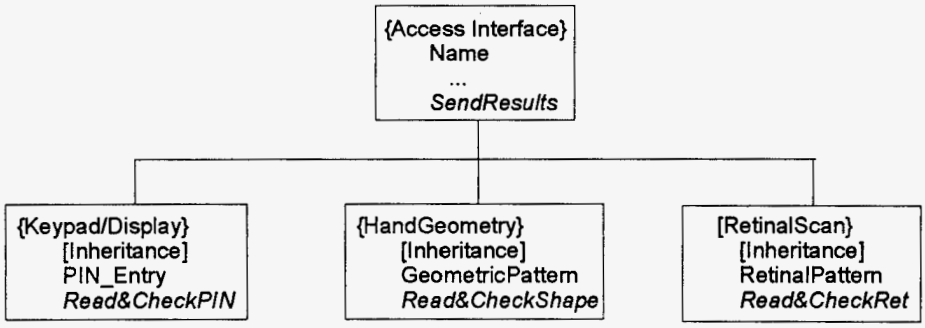
We immediately encounter a fundamental question: Should we model the entire access point as one *object class*, or more than one? We could certainly do so with one, and later "instantiate" other door-access points as objects of the same class. However, this facility will surely have other controlled-access points with different physical hardware and entry methods: Gates with chain-drive motors, entrances manned by a guard, biometric personnel id systems, etc. We should consider those system-wide needs in our plan for this particular sub-system. For this informal example, possible classes are enclosed in "curly" brackets, {}, operations (methods) are in *italics*, and attributes are in plain text.

{Access interface}	{Barrier Control*}	{Central Workstation}
Name		Name
Personal ID data		Personal ID data
Clearance level		Clearance level
Instructions		
AccessYN		
Request information		*Send information*
Show instructions	*Release barrier*	
Check authorization		

Note how the electronic lock component has been generalized to cover a wider range of potential real-world situations where {Barrier Control*} processes occur. Of course, at this point the example is too simple to provide an interesting mix of object classes (especially since the most complex component, the central workstation, is being treated as a black box). However, as we begin to use these general classes to represent more access points, we began to "grow" our object model to match the increased complexity.

(General) class	{Access interface}	
	Name	
	Personal ID data	
	Clearance level	
	Location	
	Date & time	
	Instructions	
	AccessYN	
	Request information	
	Show instructions	
	Send results	
Child class	{Keypad/display}	
	Inherits {Access interface}	
	PIN entry	[new attribute]
	Read and check PIN	[new method]
Child class	{Hand geometry reader}	
	Inherits {Access interface}	
	Geometric pattern	[new attribute]
	Read & check hand shape	[new method]
Child class	{Retinal scanner}	
	Inherits {Access interface}	
	Retinal pattern	[new attribute]
	Read & check retina pattern	[new method]

Initially, the general class was designed to have a procedure, *Read & check identification*, but our design iterations showed that the type of ID reader was the major distinguishing feature of each child class. (We kept having to polymorphize the method.) That is why this generation of classes has individual methods defined within the child class. Our *object model* would show the three child classes in a hierarchy below the parent {Access interface}. This is shown on the next page

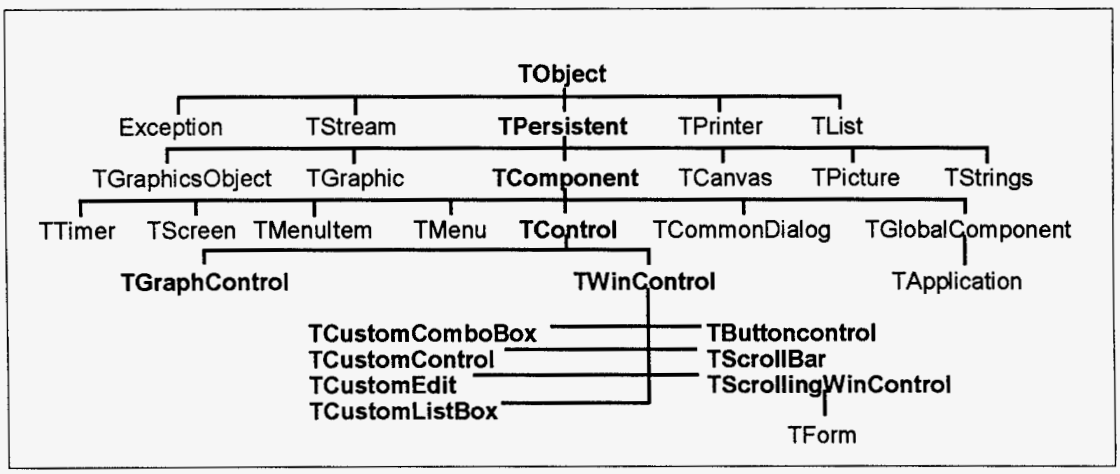


Obviously, the model for an full-fledged system would be much more complicated; at the bottom of this page is the class library structure for the Visual Component Library (VCL) of the OO Delphi™ program development environment[3].

The *dynamic model* for this system is also rather simple, but non-trivial. We might view the process as "beginning" when an access point requested updated personnel identification data from the {Central workstation}. The workstation would respond by downloading a list of persons authorized to enter the area protected by that specific access control unit. Dynamically, the system would see the request for authorization "internally," respond with instructions, and the *Read & check ...* method would collect personal identification in real time, match it against the stored ID information and "Clearance level," and set the "AccessYN" to the appropriate value. The *Send results* method has two message destinations: Whether access was denied or allowed, a time-stamped message would be sent to the {Central workstation}. If access is allowed, a message would also go to the {Barrier control} object, which would initiate its *Release barrier* method.

Even though this is a really simple system, the *functional model* might still contain some fairly complex pattern-matching algorithms for the hand geometry reader and retinal scanner.

Clearly, a program like the CMS would have a vastly more complex design package than this. Nevertheless, this example should give you some idea of what the OOA&D process looks like.



Delphi VCL Structure

[3] *Delphi Component Writer's Guide*, Borland International (1996)

APPENDIX B - CMS OBJECT CLASS NAMES

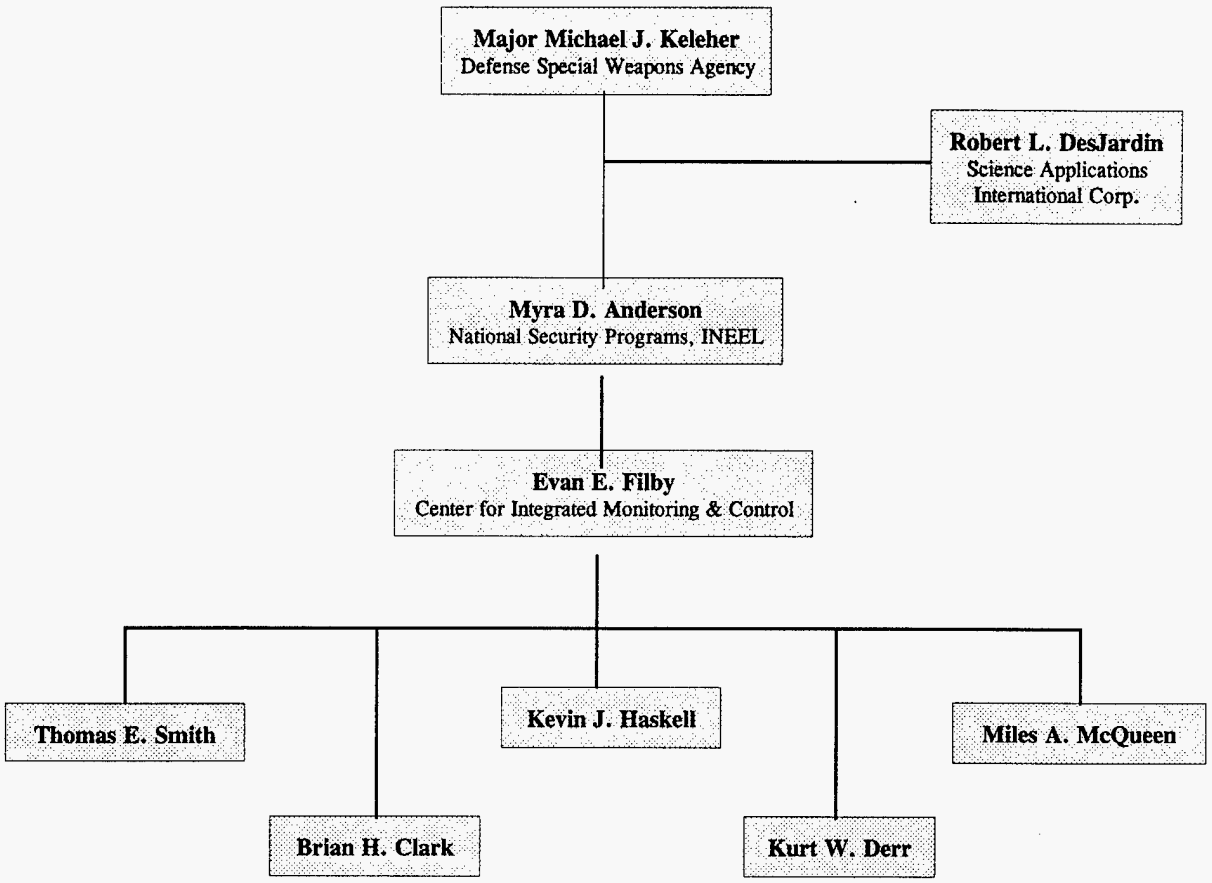
As noted in the report body, we found on the order of 320 object classes defined for the CMS project Although no class diagram was provided, they did try to follow a reasonably well-defined naming convention. It appears that the naming convention was adhered to in most cases. CMSxx is the first five letters of the names of most classes, where xx is the sub-application designator (such as DA = data acquisition, DB = data base, and so on). Then they tried to indicate the role of the object with the text that trailed. Sometimes this attempt seemed to work, many other times it did not.

CMSAutoGenEvents
CMSCIA
CMSCLegacyAppAgent
CMSDACAASPAgent
CMSDACAASPDongleConfiguration
CMSDACAASPDriver
CMSDACAimsAgent
CMSDACAimsAlarmRecord
CMSDACAimsTestStream
CMSDACAlarmAgent
CMSDACAlarmDispatcher
CMSDACAlarmLogger
CMSDACAlarmRecord
CMSDACBusyAgentDialog
CMSDACComStream
CMSDACCswlDll
CMSDACDataBuffer
CMSDACDataLine
CMSDACDavidAgent
CMSDACDavidAlarmRecord
CMSDACDDEClientWrapper
CMSDACDDEServerWrapper
CMSDACLonWorks
CMSDACLonWorksAgent
CMSDACMicrowaveInterface
CMSDACMicrowaveRequest
CMSDACSNMAnalyzerDriver
CMSDACSOHLogger
CMSDACStream
CMSDACSwldll
CMSDACTagAlarniRecord
CMSDACTagReaderAgent
CMSDACTagTestStream
CMSDACTestStream
CMSDACVideoFile
CMSDACVideoServer
CMSDataAcquisition
CMSDatabaseDDL
CMSDataManagement
CMSDataTransfer
CMSDBDataAccess
CMSDBDemoSupport
CMSDEDemoSupportAbtPackage
CMSDMCCopy
CMSDMCDatabaseBackupInterface
CMSDMCDataMgtOnDemand
CMSDMCDataMgtProcessor
CMSDMCDataMgtSchedProcessor
CMSDMCDiskInformation
CMSDMVBackupDatabase
CMSDMVCopy
CMSDMVCopyPrompt
CMSDMVDelete
CMSDTCAgent
CMSDTCEventPushAgent
CMSDTCModemMgr
CMSDTCOnDemandTransfer
CMSDTConstants
CMSDTCReceivedDataAgent
CMSDTCReceivedDataAgentEncrypted
CMSDTCTransferRequestAgent
CMSDTCTransferRequestAgentEncrypted
CMSDTCXferAgent
CMSDTNPgpStarter
CMSDTVAaeWorkingMessage
CMSDTVRequestPush
CMSEAAlarmDetails
CMSEAAlarmDetailsAction
CMSEAAlarmDetailsView
CMSEA.AlarmLog
CMSEAAlarms
CmSEAAlarmsOffLine
CMSEABMP
CMSEACBitmap
CMSEACDriftFreeTimer
CMSEACGenericLightTableApp
CMSEACImage
CMSEACImageViewer
CMSEACLightTable
CMSEACLightTableApp
CMSEACLogicalAgent
CMSEACSlide
CMSEACSlideInfoArea
CMSEACSnapshotFilter
CMSEADatabaseAccess
CMSEAMenu
CMSEANAutoAssessmentEngine
CMSEANAutoEventCategorizer
CMSEANAutoGenEngine
CMSEANEventDeclarationMatcher
CMSEANSNMRadiationSignature
CMSEASystemGreeting
CMSEAVAlarmsOffLineView
CMSEAVAnnotationBox
CMSEAVAreaID
CMSEAVAutoEventWorkingMessage
CMSEAVDeclaration
CMSEAVDeclarationActivityView
CMSEAVDeclarationICM
CMSEAVDeclarationICMView

CMSEAVDeclarationList
CMSEAVDeclarationListPart
CMSEAVDeclarationListView
CMSEAVDeclarationPurpose
CMSEAVDeclarationSelectionView
CMSEAVDeclarationView
CMSEAVEventCategoryButtons
CMSEAVEventCategoryView
CMSEAVEventDetailsComprehensiveView
CMSEAVEventList
CMSEAVEventListView
CMSEAVFinalAssessmentView
CMSEAVIcmDetailView
CMSEAVICMList
CMSEAVIcmTagEntry
CMSEAVMapViewForAlarms
CMSEAVMapViewForArchive
CMSEAVOnDemandSnapshot
CMSEAVSensorHistory
CMSEAVSensorHistoryView
CMSEAVSiteTreeContainer
CMSEAVSiteTreeView
CMSEAVSiteTreeViewForArchive
CMSEAVSiteTreeViewForSensor
CMSEAVTileSnapshots
CMSEAVTreeTestbed
CMSEnvironment
CMSEventAssessment
CMSGUCDriftFreeDelay
CMSGUCDriftingDelay
CMSGUVMessage
CMSGLTVProgressBar
CMSGUVProgressDialog
CMSGVCRubberBandTracker
CMSINCInstallation
CMSINCTableBuilderProxy
CMSInstallation
CMSINVErrorLog
CMSINVInstallScreen
CMSLegacyAppManagement
CMSLegacyAppManager
CMSLightTable
CMSLonWorksParagonTNT
CMSLonWorksParagonTNTAbtPackage
CMSLTVProgressDialog
CMSMaintenanceFunction
CMSMFVOnDemandSnapshot
CMSMFVOnDemandSnapshotNoProgressBar
CMSObjectModel
CMSOMCAbstractDatabase
CMSOMCAbstractDatabaseWithIndex
CMSOMCAbstractDbAlarm
CMSOMCAbstractDbCamera
CMSOMCAbstractDbConnectedStation
CMSOMCAbstractDbCountry
CMSOMCAbstractDbDatabaseVersion
CMSOMCAbstractDbDeclaration
CMSOMCAbstractDbEvent
CMSOMCAbstractDbEventLogic
CMSOMCAbstractDbHomeStation
CMSOMCAbstractDbIcm
CMSOMCAbstractDbIcmDeclaration
CMSOMCAbstractDbIndexList
CMSOMCAbstractDbMaintenanceDatum
CMSOMCAbstractDbMap
CMSOMCAbstractDbMonitoringArea
CMSOMCAbstractDbSensor
CMSOMCAbstractDbSite
CMSOMCAbstractDbSnapShot
CMSOMCAbstractDbSwitch
CMSOMCAbstractDbUserProfile
CMSOMCAbstractTreeobject
CMSOMCAbstractViewer
CMSOMCAimsSensor
CMSOMCAlarm
CMSOMCAlarmDistinctAreas
CMSOMCAlarmGroupId
CMSOMCAlarmId
CMSOMCAlarmLog
CMSOMCAlarmMessageBinding
CMSOMCArealdentifierDbHierarchy
CMSOMCBitMap
CMSOMCBoundaryDeclaration
CMSOMCBoundaryMicrowave
CMSOMCCamera
CMSOMCCameraComplete
CMSOMCCardReader
CMSOMCClause
CMSOMCCountry
CMSOMCDatabaseVersion
CMSOMCDeclaration
CMSOMCEntryExitDetector
CMSOMCEntryExitTagReader
CMSOMCEvent
CMSOMCEventLogic
CMSOMCFiberOpticSeal
CMSOMCFieldElement
CMSOMCFieldElementId
CMSOMCGenericSensor
CMSOMCGlobalMap
CMSOMCIcm
CMSOMCIcmDeclaration
CMSOMCIcmDeclarationItem
CMSOMCIcmId
CMSOMCIcon
CMSOMCIconList
CMSOMCIconobject
CMSOMCIdentifier
CMSOMCIdentifierDbHierarchy
CMSOMCJunctionBox
CMSOMCListViewer
CMSOMCLonWorksSensor
CMSOMCLowId
CMSOMCMaintenanceDatum
CMSOMCMaintenanceHistory
CMSOMCMap
CMSOMCMessageBindingSubSystem
CMSOMCMonitoringArea
CMSOMCMonitoringAreaId
CMSOMCNewFieldElement
CMSOMCNextIndexList
CMSOMCNotebook
CMSOMCNuclearMaterialAnalyzer
CMSOMCObjectClass
CMSOMCOnDemand
CMSOMCPhysicalGrouping

CMSOMCPortSensor
CMSOMCSelfContainedList
CMSOMCSelfContainedobject
CMSOMCSensor
CMSOMCSensorComplete
CMSOMCSensorDatum
CMSOMCSensorIdentifierDbHierarchy
CMSOMCSite
CMSOMCSiteId
CMSOMCSnapShot
CMSOMCSqlError
CMSOMCStarter
CMSOMCStation
CMSOMCStationConnected
CMSOMCStationHome
CMSOMCStationNotebook
CMSOMCStationNotebookDataXfer
CMSOMCSubSystem
CMSOMCSubSystemAASP
CMSOMCSubSystemAIMS
CMSOMCSubSystemDAVID
CMSOMCSubSystemId
CMSOMCSubSystemLonWorks
CMSOMCSubSystemVideo
CMSOMCSwitch
CMSOMCTamperSwitch
CMSOMCTaskList
CMSOMCTimeStamp
CMSOMCValidationObject
CMSOMVListViewerEditor
CMSOMVNotebookButtons
CMSOMVNotebookXferButtons
CMSOMVObjectEditor
CMSPMContainerExtension
CMSPMContainerExtensionAbtPackage
CMSSCCAlarmLogicModel
CMSSCCAssignComports
CMSSCCBitMapTableObject
CMSSCCClauseValidator
CMSSCCConfigAddressList
CMSSCCEventLogic
CMSSCCEventLogicHolder
CMSSCCPortConfig
CMSSCCStationNotebook
CMSSCCTree
CMSSCCUserListObject
CMSSCImageClass
CMSSCVAlarmLogicSetup
CMSSCVAssignComports
CMSSCVAutomaticEventAssessmentConfigure
CMSSCVEncryptionSettings
CMSSCVF,ncryptionSettingsShell
CMSSCVMapBitMapDrillDown
CMSSCVMapBitMapList
CMSSCVMapCanvas
CMSSCVMapIconList
CMSSCVMapTreeView
CMSSCVSensor
CMSSCVSensorAims
CMSSCVSensorConfigurator
CMSSCVSensorDavid
CMSSCVSensorLonworks
CMSSCVSensorVideo
CMSSCVViewUserList
CMSSCVWanButtons
CMSSCVWanConnectedStationPage
CMSSCVWorkstation
CMSSCVWorkstationDirectories
CMSSCVWorkstationShell
CMSSCVWrkData.Management
CMSSCVWrkLocalPage
CMSSCVWrkSleepMode
CMSSCVWrkUsrPreference
CMSSecurityLogin
CMSSensorConfiguration
CMSShape
CMSSLCConnectedProfile
CMSSLCSecurityListViewer
CMSSLCUserProfileobject
CMSSLDAccessSet
CMSSLManageUserCanvas
CMSSLNDongle
CMSSLNUserDatabaseAccess
CMSSLVViewManageUser
CMSSLVViewSystemLogon
CMSStarter
CMSSystemConfiguration
CMSSystemIntegrationSoftware
CMSSystemIntegrationSoftwareAbtPackage
CMSTest
CMSTestAbtPackage
CMSTools
CMSUtilities
CMSVLegacyAppNotifier

APPENDIX C - PROJECT STRUCTURE AND PERSONNEL



Technical Team

Evan E. Filby, Ph.D.

Dr. Filby is is a Consulting Scientist at the INEEL, with nearly 25 years of experience in fields related to special nuclear materials (SNM) measurement and protection. During his tenure as first a research scientist in mass spectrometry, and later as team leader for instrument develop and support, he helped improve the analytical precision on mass spectrometry for SNM accountability by a factor of three -- more, for some kinds of samples. Later, he worked with advanced instrument control systems, computer modeling of nuclear fuel cycle systems, measurements for reactor materials research, and general radiochemistry. He is now Principal Investigator, and Program Manager, for the INEEL Center for Integrated Monitoring and Control (CIMC). He is a recognized authority on safeguards instrumentation and methods, integrated monitoring for arms control and nonproliferation, information management, and other related technologies.

In recent years, he has acted as the INEEL representative on numerous DOE technical panels, advisory groups, and task teams, including: the *Program of Technical Assistance to IAEA Safeguards*, *Safeguards Laboratory Advisory Group*, *DP Task Force on Impact of IAEA Safeguards on Nuclear Weapons Complex*, *Task Force on Restart of Electromagnetic Isotope Separation Process*, and the *Special Briefing Task Force*

on North Korean Reactor History. In 1994, he organized and was principal lecturer for a short course entitled *Nuclear Research Center Purpose and Function*, a course that was meant to teach U.S. government personnel to detect possible nuclear proliferation activities at such facilities. He has been recognized with a number of awards and citations, including: received a *Lockheed-Martin Excellence Award*, four times received the *George Westinghouse Signature Award of Excellence* (nominated ten times) for significant technical achievements, nominated for an *IR-100 Award* for innovation, and received an *Operation Ivory Purpose* award for technical support during the Three Mile Island emergency.

Dr. Filby has authored or co-authored over 210 technical papers, reports, book chapters, and other publications; made over 110 formal presentations; and prepared four patents or patent disclosures. (In this context, a disclosure indicates that the work is considered innovative, but the commercial potential is insufficient to justify the patent application costs.) A selected list of relevant papers and presentations, and the titles of the four patents, are included at the end of this Appendix.

Kevin J. Haskell

Mr. Haskell is a Technical Specialist in electronics instrumentation at the INEEL, with over 20 years of experience here and at Los Alamos National Laboratory (LANL). His work at LANL included construction, test and evaluation, calibration, and maintenance of electronics instrumentation used for designing and testing U.S. nuclear weapons technology. Not only did he participate in all the activities needed to maintain and improve this equipment, he was also responsible for the day-to-day oversight of instruments being used in critical physics experiments. Some of the instrumentation involved included (among others) complex timing systems and image intensified video equipment.

At the INEEL, he has been responsible for trouble-shooting and maintaining a wide variety of industrial instrumentation and radiation sensing electronics, many of which were crucial to operations at the INEEL nuclear fuel reprocessing facility. Equipment for which he was responsible included several kinds of chemical analysis instruments (α- and γ-spectroscopy systems, mass spectrometers, and others), data acquisition systems, industrial monitoring equipment, and main-frame computer systems and peripherals. Of particular relevance to his CMS work is the *age* of some of the components for which he was solely responsible ... some of these "legacy" units are over 20 years old. In many cases, spare parts are no longer available, any manuals have long since been lost, and sometimes the vendor is no longer in business. Keeping such equipment operational requires resourcefulness, perseverance, and an intimate knowledge of how electronics systems *really* work. Those traits were invaluable in our efforts to re-start the poorly documented and already-outdated components of the CMS.

Brian H. Clark

Mr. Clark is a Staff Engineer here at the INEEL. He has around 15 years of experience as a project/design engineer for projects involving video systems, rf and telephonic communications, and computer-based control systems. Prior to coming to the INEEL, he was a Principal Engineer at Boeing Aerospace, where he was responsible for project management and design tasks for various command and control system projects. His "credits" include the design of a system for both internal and external communications and control for the Boeing 777 Airplane Program. He was also responsible for the overall design and qualification testing of a Voice and Radio Control System built by Westinghouse to be used in a Command and Control System. The system provided touch screen control of UHF, VHF, and HF radios, encryption equipment, PABX, modems, and COMSEC equipment. Mr. Clark was the lead engineer responsible for a board-level design to translate data received from the Boeing AWACS communications processor and reformat the data for recording on an AMPEX tape recorder, and was the design engineer responsible for the integration of a Collins HF radio system into the Boeing E-6 airplane for the U.S. Navy.

Also at Boeing, he was lead video Project Engineer responsible for the design and integration of a Closed-Circuit Television (CCTV) system to be used in a Command and Control System located in Saudi Arabia. He designed rack and console assemblies, integrated over 150 separate audio/video components, and generated factory acceptance test procedures for this project. After joining the INEEL, he continued a heavy involvement with complex video-based systems. He was the Project Engineer/Designer responsible for the overall design of a Mobile Radio and Television Broadcasting System for the U.S. Army, and a video/audio production studio for the U.S. Navy.

More recently, Brian was video engineer responsible for design of a 96-input by 152-output RGB video distribution system installed at the U.S. Air Force NORAD Command Center in Colorado. This sub-task of the large-scale *Granite Sentry* project set up a system for the distribution of broad-band analog video signals over secure fiber links. This was also his introduction to LonWorks technology; a LonWorks-based keypad with mini-display was designed and implemented for video switching and control. Most recently, he was the Lead Engineer on a project that upgraded the real-time SNM monitoring system for a Category I vault located at the Idaho Chemical Processing Plant. In addition to project management responsibilities, he designed a LonWorks-based node to handle the load cell monitoring output, and worked with a vendor to realize that design by adapting one of their standard products.

Thomas E. Smith

Mr. Smith is a Electronics Systems Engineer with over a decade of experience with distributed monitoring and control systems, audio/video distribution systems, and micro-electronics. Before coming to the INEEL, he worked at Signetics and at National Semiconductor, where he was responsible for evaluation and testing of new releases of advanced CMOS logic devices, including new and redesigned circuits for the Signetics FAST logic family. He also helped develop process improvements to maintain and enhance production yields in their chip-fabrication facilities. Among his accomplishments was the performance of yield enhancement analyses that led to the development of formulae to accurately predict the number of possible good die on a wafer. He also developed a user-friendly Man Machine Interface (MMI) for their test engineering program, and concurrently devised a training program for operators who would be using the new MMI.

To some extent, his career here at the INEEL paralleled that of Brian Clark, including work on the video production studio and mobile radio and television station project. For these projects, he worked directly with the customer to develop requirements and specifications, and supervised technicians and crafts personnel during construction, installation and testing at these facilities. He had similar duties on the *Granite Century* NORAD project, as well as tasks involving PLC (Programmable Logic Chip) design and programming. The specifications for that part of the project required fully redundant environmental, security, and projector control. When he joined the CIMC, he received additional LonWorks training at Echelon Corporation and played a significant role in several on-going distributed monitoring and control projects.

Miles A. McQueen

Mr. McQueen has around 20 years of experience with real-time software development, complex simulations, algorithm design, software engineering, systems analysis, and the development of integrated hardware/software systems. He is currently an Advisory Engineer here at the INEEL, where his duties have included the development of C++ software in support of system simulations, technical coordination for the Spatial Analysis and Internet Systems Group, the development of embedded real-time software systems, and work with several distributed monitoring and control projects for the CIMC. He has also performed process evaluations for the Software Process Engineering Department, and organized and executed a broad-based validation and verification effort to support various DoD programs. One of his current research projects involves an examination of how "fault tolerant" principles might be integrated onto a LonWorks

network. Fault tolerant systems have the ability to continue operation in the presence of faults (bad data or malicious tampering), and the development of such capabilities could significantly enhance the security of a monitoring and control network.

Prior to joining the INEEL, he spent a number of years at the Hughes Aircraft Corporation. While there, he was Manager of the Advanced IR Tracker Development effort, which required development of real-time simulations and tactical software in C, Ada, assembly language on a variety of commercial platforms and proprietary hardware. He had previously had the technical lead for a large research and development program in support of their anti-armor missile systems project. This work included the development of real-time, hardware-in-the-loop, 3-D simulations used to guide the formulation of system specifications, and the design of man-aided tracking algorithms.

Kurt W. Derr

Mr. Kurt Derr has over 15 years of INEEL experience as a Project Manager and Principal Investigator for software engineering projects involving the use of object oriented (OO) technology. His OO project experience include resource management systems, data conversion utilities, database question bank protocols, and advanced database access approaches. Other OO activities have involved geographical information systems, a "virtual storefront" project, object-oriented middeware, and graph visualization. His 25-plus year career includes software development and project management jobs at Datapoint Corporation, Datascan America, Computran Systems Corp, NCR Corporation, and Sperry Rand Corporation. Specific accomplishments included development of radar system software, computerized traffic control systems, and text storage and retrieval software for PC databases. He was also project manager for the deployment of a distributed client-server computer network using a UNIX derivative operating system.

In addition to his direct INEEL duties, Mr. Derr is a computer science instructor at the local branch campus of the University of Idaho. He has taught structured analysis and design, OO analysis and design, OO programming, "C" language programming, C programming in the UNIX environment, computer networking, and distributed processing. In addition to various technical papers and presentations, he has had a book on the Object Modeling Technique (OMT) published [see reference list].

Relevant Reference Titles

E. E. Filby, L. J. Hanson, M. A. McQueen, T. E. Smith, "Distributed Auto-Control for Optimum Video Surveillance," *J. Institute of Nuclear Material Management*, Proceedings on CD-ROM (1997).

M. A. McQueen, E. E. Filby, *Support Framework for Agreement and Synchronization Using LonWorks® Distributed Networks*, LDRD technical report (October 1, 1997).

E. E. Filby, R. K. Albano, T. E. Smith, *Cooperative Remote Monitoring Test and Evaluation: Device and Scenario Review*, report INEEL/EXT-97-0021, LIMTCO (February 1997).

E. E. Filby, T. E. Smith, R. K. Albano, M. K. Andersen, R. L. Lucero, K. M. Tolk, N.S. Andrews, "Testing Integrated Sensors for Cooperative Remote Monitoring," *J. Institute of Nuclear Material Management*, proceedings issue (1996), p. 982.

E. E. Filby, T. E. Smith, R. L. Albano, "LonWorks® Device Integration for International Treaty Verification," *LonUsers® International Conference Proceedings*, Echelon Corporation, Palo Alto, California (May 21-22, 1996) p. 139.

C. D. Friesen, N. S. Andrews, D. W. Myers, E. E. Filby, "Modular Integrated Monitoring System (MIMS)," *Arms Control and Nonproliferation Technologies* magazine, DOE/NN/ACNT-95D (Fourth Quarter 1995).

K. W. Derr, *Applying the Object Modeling Technique*, SIGS Publications, Inc., distributed by Cambridge University Press (August 1995).

R. L. Martinez, D. R. Waymire, D. A. Fuess, D. W. Myers, C. I. Frerking, E. E. Filby, "Modular Integrated Monitoring System (MIMS) Field Test Installation," *J. Institute of Nuclear Material Management*, Vol. XXIV, proceedings issue (1995) p. 1100.

D. R. Waymire, R. L. Martinez, E. E. Filby, D. W. Myers, B.J. Wheeler, J. Abraham, *Final Report: FY94 Field Test of MIMS Subsystem Capability at INEL*, Sandia National Laboratory Report (December 1994).

G. J. McManus, E. E. Filby, *Monitoring for Nobles Gases as a Reprocessing Signature*, INEL-SP-349, WINCO (May 1994).

E. E. Filby, *Nuclear Facility Transparency Signatures for the Idaho Chemical Processining Plant*, WINCO-1185 (November 1993).

K. W. Derr, "Object-Oriented Programming," INEL Computer Symposium, Idaho Falls, ID (October 1993).

E. E. Filby, "Safeguards-Related Activities at the INEL," Overview Meeting for the International Atomic Energy Agency, Vienna, Austria (October 13, 1993).

K.W. Derr, "A Proposal for DOE-Wide Compliance Assessment Database for Risk Assessment," Energy Facility Contractors Group Conference (February 1993).

E. E. Filby, J. K. Hartwell, *Control of High-Enriched Uranium at the Idaho Chemical Processing Plant*, Parts 1 and 2, program Report ST129A, U. S. DOE (October 1992).

E. E. Filby, R. A. Rankin, D. E. Yoshida, "Automated Intelligent Assistant for Mass Spectrometry Operation," *Proceedings of AI'91: Frontiers in Innovative Computing for the Nuclear Industry*, American Nuclear Society, Idaho Section (September 1991

E. E. Filby, R. A. Rankin, G. W. Webb, "Diagnostics Aid for Mass Spectrometer Trouble-Shooting," *Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry*, M. C. Majumdar (Ed.), Plenum Publishing Corporation, New York (1987). p. 853.

E. E. Filby, R. A. Rankin, S. J. Petrofsky, G. W. Webb, "Predicting Peak Sample Loads for Multi-Process Analytical Support", *Modeling and Simulations on Microcomputers: 1986*, C. C. Barnett (Ed.), Soc. for Computer Simulation, La Jolla, California (1986) p. 43.

E. E. Filby, W. A. Emel, "Implementation of New Analytical Measurement Techniques within an Operating Process Environment," *Proceedings of the Conference on Safeguards Technology: The Process - Safeguards Interface*, CONF-831106, E. A. Hakkila, M. H. Campbell, N. M. Trahey (Ed.), American Nuclear Society and Institute for Nuclear Materials Management (August 1984) p. 135.

K. W. Derr, *Packet Switching*, Master thesis, University of Idaho (1979). Material also presented at the National Electronics Conference ,1981.

Patents and Disclosures

"Method of Recycling Lithium Borate to Lithium Borohydride through Diborane," No. 3,993,732. Author: E. E. Filby. Issued Nov. 23, 1976.

"Method of Recycling Lithium Borate to Lithium Borohydride through Methyl Borate," No. 4,002,726. Author: E. E. Filby. Issued Jan. 11, 1977.

"Automated Intelligent Overseer for Mass Spectrometry Operation," No. 5,025,391. Authors: E. E. Filby, R. A. Rankin. Issued June 18, 1991.

"Mass Spectrometer Control System," E. E. Filby, R. A. Rankin, G. W. Webb; submitted, January 1986; Patent Committee recommended recognition, March 1986, but did not file.

Report Number (14) INEEL/EXT--98-00303

Publ. Date (11) 199803

Sponsor Code (18) DOE/EM , XF

UC Category (19) UC-2000 , DOE/ER