SANDIA REPORT

SAND2000-0729
Unlimited Release
Printed August 2000

A Configurable, Object-Oriented, Transportation System Software Framework

Suzanne M. Kelly, John W. Myre, Mark H. Price, Eric D. Russell and Dan W. Scott

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND2000-0729
Unlimited Release
Printed August, 2000

A Configurable, Object-Oriented, Transportation System Software Framework

Suzanne M. Kelly and John W. Myre
SECOM Systems Department

Mark H. Price
Communication Systems Department

Eric D. Russell
Scalable Computing Systems Department

Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM

Dan W. Scott, TECH REPS INC
Albuquerque, NM 87110

ABSTRACT

The Transportation Surety Center, 6300, has been conducting continuing research into and development of information systems for the Configurable Transportation Security and Information Management System (CTSS) project, an Object-Oriented Framework approach that uses Component-Based Software Development to facilitate rapid deployment of new systems while improving software cost containment, development, reliability, compatibility, and extensibility. The direction has been to develop a Fleet Management System (FMS) framework using object-oriented technology. The goal for the current development is to provide a software and hardware environment that will demonstrate and support object-oriented development commonly in the FMS Central Command Center and Vehicle domains.

Acknowledgment

The authors thank the other members of the CTSS development team Robert N. Cook and Karl L. Green as well as our managers for their efforts in helping realize the CTSS software product.

Contents

LIST OF FIGURES

LIST OF TABLES

This page left intentionally blank.

Executive Summary

This document defines the software development for the Configurable Transportation Security and Information Management System (CTSS) project. It describes the CTSS Object-Oriented Framework and the CTSS Component-Based Software Development approach.

CTSS is being developed to provide the center with an information management infrastructure suited to transportation surety that can support production system upgrades and allows for the rapid development of applications suited to new transportation customers. The product is a library of components that developers can use, extend, and customize for a specific transportation application.

Goals

The CTSS project has selected an Object-Oriented Framework architecture and a Component-Based Software Development approach to meet the following goals:

- rapid development of highly customized applications
- increased developer productivity
- reduced costs for software development and maintenance
- increased software quality (i.e., enhanced reliability and robustness)
- improved software maintainability and modifiability
- computer platform independence
- better integration with legacy systems

- easier extendability to new transportation applications and other applications.

Rationale

To maximize the benefits of Object-Oriented Technology, the project is

- developing an Object-Oriented Framework—a set of prefabricated software building blocks (objects) that programmers can use, extend, or customize for specific computing solutions
- using a Component-Based Software Development approach, which builds systems by means of combination, aggregation, and integration of pre-engineered and pretested software objects.

As a result, developers benefit from a higher level of code and design reuse than what is practical with other design approaches. Developers using other approaches, which rely on procedural programming techniques, cannot easily find the infrastructure and design guidance that is built into framework components.

Framework Advantages

Key advantages of the Object-Oriented Framework approach include:

- **Infrastructure and architectural guidance.** Much of the needed functionality already exists in the framework, thus reducing coding, testing, and debugging efforts. Applications developed using an Object-Oriented Framework approach tend to be smaller, as well as more robust, maintainable, and reusable.
- **Mechanism for reliably extending functionality.** Applications are developed by using the framework as a starting point and writing smaller amounts of code to modify or extend the framework's behavior. Compatibility and interoperability are not sacrificed because the interfaces are well defined.
- **Reduced maintenance.** When a framework bug is fixed or a new feature is added, the benefits of those changes become available more quickly to the derived classes. Also, changes are made only in one place, thus, the chance of introducing additional errors in the code is minimized.
- **Code reuse.** By developing code as decoupled, stand-alone units, other developers or customers may be able to make use of the code developed for their own products.
- **Robustness.** Objects by nature can be tested much more completely than procedural programs, resulting in a more robust component.

It is important to note, however, that benefits are not necessarily immediate. The Object-Oriented Framework requires a long-term investment in learning, documentation, maintenance, and support, as Figure 1 shows.

Component-Based Development Advantages

Component-Based Development is the process of building systems by combining, aggregating, and integrating pre-engineered and pretested software objects. The Object-Oriented Framework approach described above is one manifestation of component-based development. However, the CTSS project is not limited to framework components and the objects they incorporate. The CTSS project is also integrating complete application programs for pre-anticipated needs. CTSS provides application programs for such typical functions as communication control and message handling.

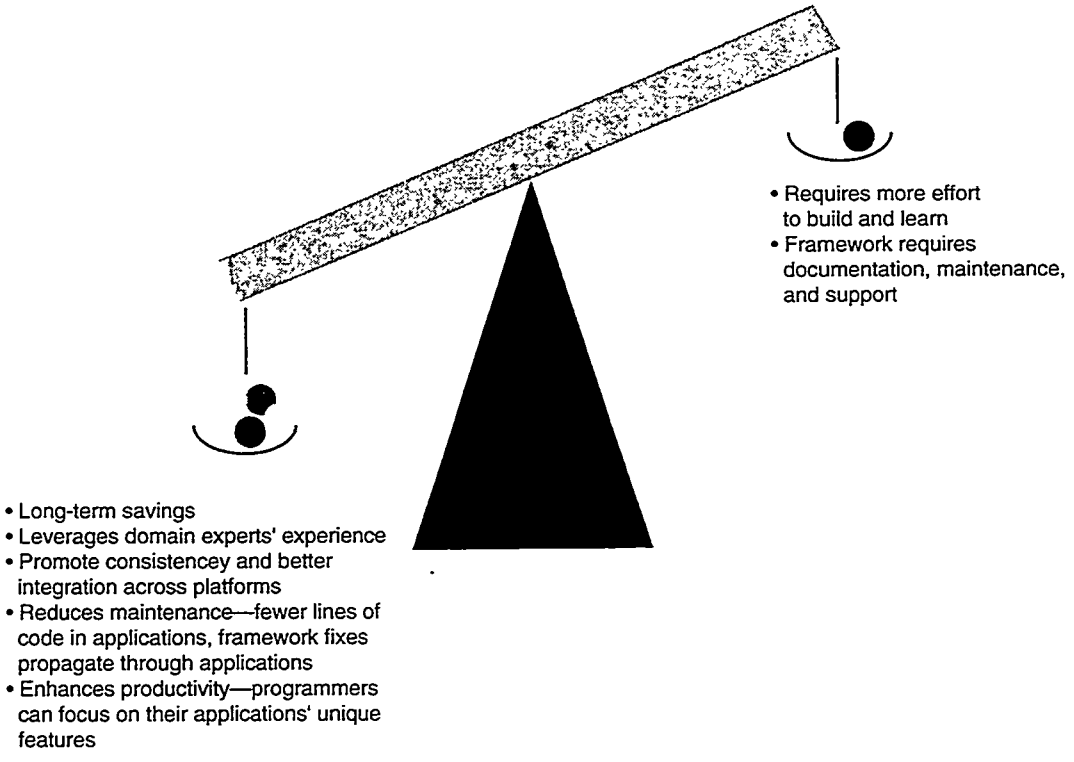

Figure 1 Cost and benefits of Object-Oriented Framework approach.

Architecture

Fleet Management System Domains

The CTSS Framework defines a Fleet Management System as containing some set of the following domains:

- Command center domain (e.g., resource allocation, mapping, vehicle interface, operator interface)
- Vehicle domain (e.g., resource monitoring, driver interface, emergency notification)
- Trailer domain (e.g., sensor monitoring, countermeasures, container interface)
- Container domain (e.g., materials monitoring).

CTSS concentrates on the Vehicle and Command Center domains and provides framework objects for interfacing with the other domains. Software in the Trailer and Container domains have constraints that the CTSS framework layers described below, do not support. Trailers and containers typically utilize embedded processors which have limited functionality operating systems and compilers. In addition, they support only a few, specialized external interfaces.

Framework Levels

The CTSS Framework consists of the following four functional levels:

- **Application Level.** The Application Level consists of high-level applications written for tasks using common objects, application standards, and standard communication between applications provided by the Framework Level services. The application level is the responsibility of the developer using CTSS components. As mentioned previously, CTSS provides a few applications programs that are routinely used by transportation systems.
- **Framework Level.** The Framework Level interfaces to the Middleware Level and the Application Level. It is the core of CTSS development. The Framework Level consists of transportation-specific objects that an application developer can utilize to build a tailored, customer-specified application.
- **Middleware Level.** To provide support for the maximum number of vendors, the Middleware Level is COTS system integration software, which may be purchased or shareware. It provides application-independent objects with service-oriented functions such as interprocess communications, intraprocess communications, encryption algo-

rithms, distributed logging, distributed events, shared memory, and time synchronization.

- **Platform Level.** The Platform Level consists of the vendor-supplied software and hardware. The platforms are Windows 95/98/NT running on PC hardware for the Vehicle domain and Windows NT in the Command Center domain. The command center also supports UNIX running on Silicon Graphics, Data General, or Compaq hardware.

Scope of Work

Application Level

Work in the Application Level includes development of the sample/prototype applications in the Vehicle domain:

- Fleet data communication using the QUALCOMM satellite system.
- Fleet data communication using the ORBCOMM satellite system.
- Fleet data communication using telephonic communication devices.
- Real-time tracking using a GPS unit.
- A graphical user interface which sends and receives messages to/from the command center domain
- Integration with trailer monitoring/communications.

These applications will complement the applications prototyped in the Command Center domain:

- Fleet data communication using the QUALCOMM satellite system.
- Fleet data communication using the ORBCOMM satellite system.
- Fleet data communication using telephonic communication devices.
- A graphical user interface for messages and database queries.
- A mapping/tracking subsystem.

Some programs from the MAP/Tracker project are reused as the basis for the command center application work.

Framework Level

The bulk of CTSS development efforts are in the Framework Level. Transportation-related entities have been defined as software objects. These objects are described later in this document, and are more fully defined in terms of their state and behavior in the CTSS Component Library Reference Manual.

Example objects in vehicle domain are: GPS, Qualcomm, and TeleComm. Example objects in the command center domain are Vehicle, Sensor, LatLong, KeywordMessage, as well as complementary Qualcomm and TeleComm objects.

Some objects from the MAP/Tracker project are reused in the Framework Level.

Middleware Level

COTS software for the following have been identified, procured, and integrated:

- device communication drivers
- encryption algorithms
- communications software to provide a wide spectrum of interprocess communications functions
- database management systems to provide a persistent, client-server architecture for storage of objects
- distributed logging
- a user-interactive agent (e.g. voice capture and message annunciation).

Platform Level

At the Platform Level, initial development will be done on the following platforms:

- Windows 95/98/NT (Vehicle domain)
- Windows NT (Command Center domain)
- UNIX (Command Center domain).

The following compilers and tools are being used:

- C++ (Command Center domain)
- Visual Basic (Vehicle domain)
- runtime debuggers

Conclusions

Development of the CTSS software is complete. In general, CTSS has been a successful endeavor, as evidenced by the number of number of implementations described in Appendix C. As with any project, there have been areas that were very successful, and others that were not.

Successes

The architecture greatly enhanced the ability to develop compatible components. By predefining the fleet management system domains and the framework levels, clear interface demarcation points were a natural outcome.

The goal of "rapid development of highly customized applications" was achieved. The STORC application described in Appendix C.4 gives evidence to this achievement.

The goals of "increased developer productivity" and "reduced costs for software development and maintenance" applied only to the specific applications developed using CTSS. Because of their generalized nature, component development takes 50% longer than software developed to a specific set of customer requirements.

The goal of "improved software maintainability and modifiability" was achieved because it was so easy to extend the components for specific applications. For example, the CTSS messaging capability allowed tremendous flexibility as each application had specific information management requirements. Fixed message types would have required significant rework for each application. Instead, the keyword message used by CTSS allowed for extension and definition of new keywords with virtually no changes to the primary class.

The goal of "better integration with legacy systems" was not verified on the vehicle side. There were no specific applications developed with this goal in mind. The command center software was able to successfully integrate with the existing legacy system in the Sandia Proof of Concept. (See Appendix C.2.)

The goal of "easier extendability to new transportation applications and other applications" was shown when CTSS was integrated with the Cargo Monitoring System. This system provided features in all domains of a fleet management system. Also, as part of the CTSS work, a white paper described how to extend CTSS to cooperate with a fixed-site Material Monitoring System. While the implementation was never done, the design appeared feasible.

Disappointments

The goal of computer platform independence was not achieved. Visual Basic was the only language which provided sufficient robustness and functionality needed for the CTSS vehicle-based components. Visual Basic only runs on Windows-based operating systems.

The goal of "increased software quality" was not verified or refuted during the CTSS development. It will require the test of time to determine if the software is more robust and reliable than that achieved by software developed using other design philosophies.

SECTION 1 Introduction

The Configurable Transportation Security and Information Management System (CTSS) software project is a framework-oriented, component-based development initiative funded by Sandia National Laboratories' Weapon Materials Stewardship (WMS) Program. The theme of the WMS Program is to integrate technologies from the activity areas (e.g., Security Technologies, Monitoring Systems, Advanced Sensors, Containers, Transportation, and Automated Handling) to provide an integrated, comprehensive weapon materials management capability that can be applied to multiple application areas (e.g., Pantex, other DOE, other U.S., Russia, and IAEA).

1.1 Purpose

The purpose of this document is to describe the software component-based development approach and architecture being used by the CTSS project. The intended audience for this document is the CTSS development team, independent architecture reviewers, developers utilizing the CTSS software, and program management.

1.2 Scope

This document focuses on development of the CTSS framework. This document does not provide designs for a specific system.

1.3 Definitions, Acronyms, and Abbreviations

The following definitions, acronyms, and abbreviations are used in this document or may be used in CTSS-related discussions.

abstract class . . . A class whose primary purpose is to define an interface. An abstract class defers some or all of its implementation to subclasses. An abstract class cannot be instantiated.

APL . . . . . Automatic Position Location system

ACE . . . . . Adaptive Communications Environment, shareware that provides communication services such as connection configuration, interprocess communication, and named pipes.

application system An operating system-independent software environment that supports applications and the way they interact with users, other applications, and distributed systems.

big-endian . . . . . Describes a computer architecture in which, within a given multi-byte numeric representation, the most significant byte has the lowest address (the word is stored "big-end-first.") Most processors, including the IBM 370 family, the PDP-10, the Motorola microprocessor families, and most of the various RISC designs are big-endian. See little-endian.

C++ . . . . . One of the most used object-oriented languages, a superset of C developed primarily by Bjarne Stroustrup at AT&T Bell Laboratories in 1986. In C++, a class is a user-defined type, syntactically a struct with member functions. Constructors and destructors are member functions called to create or destroy instances. A friend is a nonmember function that is allowed to access the private portion of a class. C++ allows implicit type conversion, function inlining, overloading of operators and function names, and default function arguments. It has streams for I/0 and references. C++ 2.0 (May 1989) introduced multiple inheritance, type-safe linkage, pointers to members, and abstract classes. C++ 2.1 was introduced in ["Annotated C++ Reference Manual", B. Stroustrup et al., A-W 1990] supporting templates, exception handling, name spaces, and run time type identification.

class . . . . . A template for defining the behavior of a particular type of object. Objects of a given class are identical in form and behavior. In C++, a class has zero or more data members and zero or more member functions referred to as the class interface. The class has levels of program access, public, protected, or private. Non-public level representation enforces encapsulation. The class has a name.

class library . . . . A collection of one or more classes that programmers use to implement an area of functionality. Compare: framework.

client-server architecture A common form of distributed system in which software is split between server tasks and client tasks. A client sends requests to a server, according to some protocol, asking for information or action, and the server responds. There may be either one centralized server or several distributed ones. This model allows clients and servers to be placed independently on nodes in a network, possibly on different hardware and operating systems appropriate to their function, e.g. fast server/cheap client.

COM. . . . . Component Object Model, an object sharing technology developed by Microsoft.

component . . . . . (a) Any standard, reusable, previously implemented unit that is used to enhance the programming language constructs and to develop applications. [Jacobson] (b) A physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. [Booch]

component-based development Development that primarily consists of integrating instances of previously defined classes of objects.

CORBA. . . . . Common Object Request Broker Architecture, which specifies a system that provides interoperability between objects in a heterogeneous, distributed environment in a way that is transparent to the programmer.

COTS . . . . . commercial off-the-shelf

coupling . . . . . The degree to which software components depend on each other.

CTSS . . . . . The Configurable Transportation Security and Information Management System (CTSS) project

daemon . . . . . a program that runs unattended to perform a standard service without overly interfering with the computer system's resources.

DBMS. . . . . Data Base Management System

DCOM . . . . . Distributed Component Object Model, a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. DCOM was developed by Microsoft Corporation with the Microsoft Windows NT 4.0 operating system.

DES . . . . . Data Encryption Standard

DOE. . . . . The Department of Energy

domain task . . . . A task that involves customized applications, such as banking transactions or the Fleet Management System (FMS), or one involving a specific area of functionality, such as multimedia.

extensibility . . . . A property that allows additions and modifications to existing classes. There are several kinds of extensibility: (1) new classes may be defined, commonly based on existing classes; (2) existing classes may be modified to add new operations, attributes, constraints, or implementations, extending the protocol of the class; and (3) existing instances may acquire or lose a type (e.g. as when a "student" becomes an "employee"). The term "schema evolution" is used for operations that modify existing class definitions or the inheritance graph; the term "instance evolution" is used for the process of making existing instances consistent with modified class definitions.

framework . . . . . A set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions. With frameworks, software developers do not have to start from scratch each time they write an application. Frameworks are built from a collection of objects, so both the design and code of a framework may be reused.

encapsulation . . . An object-oriented programming technique that keep each object's data and logic hidden from other objects

so that the only thing an object "knows" about another object is the object's interface.

FMS . . . . . Fleet Management System, a communications system that tracks, communicates with, and monitors vehicles and possibly their cargo in real-time.

GPS . . . . . Global Positioning System

GUI . . . . . Graphical User Interface

IAEA . . . . . International Atomic Energy Agency

inheritance . . . . . A property that allows an object class to have the same behavior as another class and extend or tailor that behavior to provide special action for specific needs.

JAVA . . . . . While primarily considered a programming language, JAVA also provides an architecture and an environment for program development. The architecture provides portable, multi-threaded applications over heterogeneous hardware. The environment includes a wide range of tools all developed for the JAVA Virtual Machine. The language is an evolution of C++ and provides familiar constructs in an object-oriented context.

legacy system . . . A computer system or application program which continues to be used because of the prohibitive cost of replacing or redesigning it and despite its poor competitiveness and compatibility with modem equivalents. The implication is that the system is large, monolithic, and has difficult to modify interface functions.

little-endian . . . . . Describes a computer architecture in which, within a given 16- or 32-bit word, bytes at lower addresses have lower significance (the word is stored "little-end-first"). The PDP-11 and VAX families of computers and Intel microprocessors and a lot of communications and networking hardware are little-endian. See big-endian.

MAP/Tracker . . . . An object-oriented real-time material-tracking and emergency-response system developed by Sandia National Laboratories. MAP/Tracker also provides a data viewer to the object-oriented databases that MAP/Tracker is based on.

MMaC . . . . . Materials Management and Control

multi-tier architecture An architectural style in which each tier performs a pre-defined portion of the work. For example, in a 3-tier architecture, the first, or most visible layer, does the pre-

sentation which manages how users see and interact with the system. The second layer is the functional portion, which contains the application logic. The third layer manages the data. Systems developed with this architecture are able to "wrapper" existing systems and combine them with new functions to produce a unified, new system.

NIST..... National Institute of Standards

object..... A small, self-contained unit of software functionality. It contains both data and the procedures for working with that data. An object is defined by a class, which is a template for an object.

OLE..... Microsoft's Object Linking and Embedding environment for components built on COM/DCOM.

OMG ..... The Object Management Group, an industry consortium whose mission is to define a set of interfaces for interoperable software.

OODBMS ..... Object-Oriented DataBase Management System, a persistent database providing distributed client-server access to stored complex objects. Access is transparent in that there are no explicit reading or writing of objects, but the OODBMS maps the objects into the program directly as if they were program objects. Objects cached on the client system are accessed at the same speed as program virtual memory references.

OOT..... object-oriented technology

OOP..... object-oriented programming

ORB..... Object Request Broker, the OMG's standard that lets clients invoke methods on remote objects.

persistence..... The ability to store data in such a way that its existence extends beyond the process that created it.

pipe and filter architecture An architectural style in which each process in the system has a set of inputs and outputs. Each process reads streams of data on its inputs and produces streams of data on its outputs, delivering a complete instance of the result in a standard order. This is usually accomplished by applying a local transformation to the input streams and computing incrementally so output begins before input is consumed. Hence processes are termed "filters". The connectors of this style serve as conduits

for the data, transmitting outputs of one filter to inputs of another. Hence the processes are termed "pipes".

polymorphism . . . A property that allows two or more objects to respond differently to the same message; i.e., the ability to substitute objects of matching interface for another at runtime.

POSIX . . . . . The IEEE Portable Operating System Interface. Standard P1003.lc(3).

QUALCOMM . . . . A company that develops, manufactures, markets, licenses, and operates advanced communications systems and products based on its proprietary digital wireless technologies. The CTSS project uses QUALCOMM's OmniTRACS system, a geostationary satellite-based, mobile communications system providing two-way data and position reporting services, to communicate with vehicles.

RSA . . . . . RSA Data Security, Inc., a security products company.

STL . . . . . C++ Standard Template Library

thread . . . . . An independent sequence of execution of program control within a process. Threads may be based on the UNIX International (UI) standard or the more portable POSIX pthread standard. Threads within a process are scheduled by the operating system's thread manager and execute independently. On multiprocessors, different threads may execute on different processors. On uniprocessors, threads may interleave their execution arbitrarily and indeterministically.

virtual function . . In C++, derived classes may overload functions from their parent class(es) if the functions have been declared as a virtual function. Every class with virtual functions has a virtual function table, vtbl, that points to the virtual functions. Every object with virtual functions has a pointer to its class vtbl.

WMS . . . . . Weapons Material Stewardship

WWW . . . . . World Wide Web

XDR . . . . . Sun Microsystems' eXternal Data Representation library, which supports a communications protocol standard for portable object representation, which allows a platform-independent mechanism for transparent access to distributed objects.

1.4 References

Additional information on Object-Oriented Framework technology and the CTSS project can be found in the following references:

- Carroll, Martin D., and Margaret A. Ellis, *Designing and Coding Reusable C++*, Addison-Wesley Publishing Co., 1995.
- Firesmith, Donald G., and Edward M. Eykholt, *Dictionary of Object Technology*, SIGS Books, Inc., 1995.
- Gamma, Helm, Johnson, and Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Co., 1995.
- Garlan, David, and Mary Shaw, *An Introduction to Software Architecture*, Technical Report CMU/SEI-94-TR-21.
- Johnson, R.E., "Designing Reusable Classes," *The Journal of Object-Oriented Programming*, Vol. 1, No. 2, 1988, pp 22-35.
- Kara, Dan, "Build vs. Buy: Maximizing the Potential of Components," *Component Strategies*, July 1998, pp 22-35.
- Kelly, Suzanne M., et. al, *CTSS-Configurable Transportation Security and Information Management System Component Library Reference Manual*, October 30, 1998, Internal Department Document.
- Schmidt, Douglas C., *The Adaptive Communication Environment, An Object-Oriented Network Programming Toolkit for Developing Communication Software*, Department of Computer Science, University of Washington.
- Singer, Alan, "Integrating Heritage Back-Office & Open Client/Server Systems," *Object Magazine*, November 1997, pp 28-31.
- Taligent, Inc., "Building Object-Oriented Frameworks," Taligent White Paper.
- Taligent, Inc. "Leveraging Object-Oriented Frameworks, A Technology Primer from Taligent," Taligent White Paper.

1.5 Personnel

The CTSS development personnel and program management are working together to ensure that this project delivers a hardware and software environment that will demonstrate and support object-oriented development commonly needed in a Fleet Management System.

Team members are listed below:

- Design and Development

- Suzanne M. Kelly, 6321, Command Center Domain, project lead
- Robert N. Cook, III, 6321, Vehicle Domain
- Karl L. Green, 6321, Encryption
- John W. Myre, 6321, Command Center Domain, Encryption
- Mark Price, 6321, Vehicle Domain

• Management
- Richard E. Thompson, 6321, Department Manager
- Henry J. Abeyta, 6301, WMS Transportation Surety Area Manager
- Dennis Mangan, 5314, WMS Program Manager;

This page left intentionally blank.

SECTION 2 Project Goals and Rationale

2.1 Goals

The major objectives of the CTSS Software Architecture project include:

- Rapid development of highly customized applications.
- Increased developer productivity.
- Improved containment of costs associated with hardware purchases, software development and maintenance.
- More straightforward development of multiple applications running on a variety of platforms throughout the development cycle (from initial single-host proof-of-concept systems to final multiple-host fielded systems)
- Enhanced software reliability and robustness through reuse of software from application to application.
- Platform compatibility resolution through consolidated access to existing software.
- Better integration with legacy systems.
- The capability to build new applications from a growing framework of current applications.
- Support of new transportation functions such as resource scheduling and leveling.

Component-based development using an object-oriented framework has been chosen as the most appropriate strategy for attaining these goals, for the reasons detailed in this section.

2.2 Rationale

As with nearly all software development efforts over the last two decades, software development for Weapon Materials Stewardship (WMS) program by Sandia National Laboratories has continued to evolve. These changes have been motivated by the need to produce software more quickly and to deliver more value to end-users. Despite gains, we still face long development cycles that produce software that runs the risk of not addressing customer problems adequately. These limitations have moved us to adopt object-oriented technology (OOT) because of its potential to significantly increase developer productivity and encourage accurate representation of the real-world environment.

2.2.1 Object-Oriented Technology and the Object-Oriented Framework

OOT has the potential to dramatically improve the software development process for the WMS program. However, we are not focusing only on OOT, but on how this technology is delivered (see Section 2.2.2 below). We are developing an object-oriented framework approach for WMS software development. This approach uses extensible sets of object-oriented classes that are integrated to execute well-defined sets of computing behavior, thus providing a sound foundation for fully exploiting OOT.

An object-oriented framework approach will

- Empower developers to fully leverage OOT with a framework that spans entire systems, delivering rich built-in functionality at all levels and providing more computing value than when this functionality is added on as an option.
- Provide well-defined mechanisms that allow software and hardware developers to reuse, extend, and leverage this functionality for increased productivity and integration.
- Provide an integrated development environment designed for object-oriented programming (OOP) that includes a wide variety of development tools all designed for rapid application development and customization.

2.2.2 Component-Based Software Development

From the perspective of component-based software development, the framework-based approach is just one approach in a spectrum of component-development approaches that the CTSS project is now making use

of or planning to make use of (Figure 2). While the object-oriented framework approach has been suitable for initial CTSS development, the CTSS project is now adopting a wider variety of component families and using them in combination with each other to build and integrate more substantive systems.

A description of the spectrum of component development approaches (as described by Kara, 1998 and illustrated in Figure 2, also from Kara, 1998), follows:

- **Integrative Approach**: pieces of existing applications are encapsulated and then integrated with other components to form a complete solution.

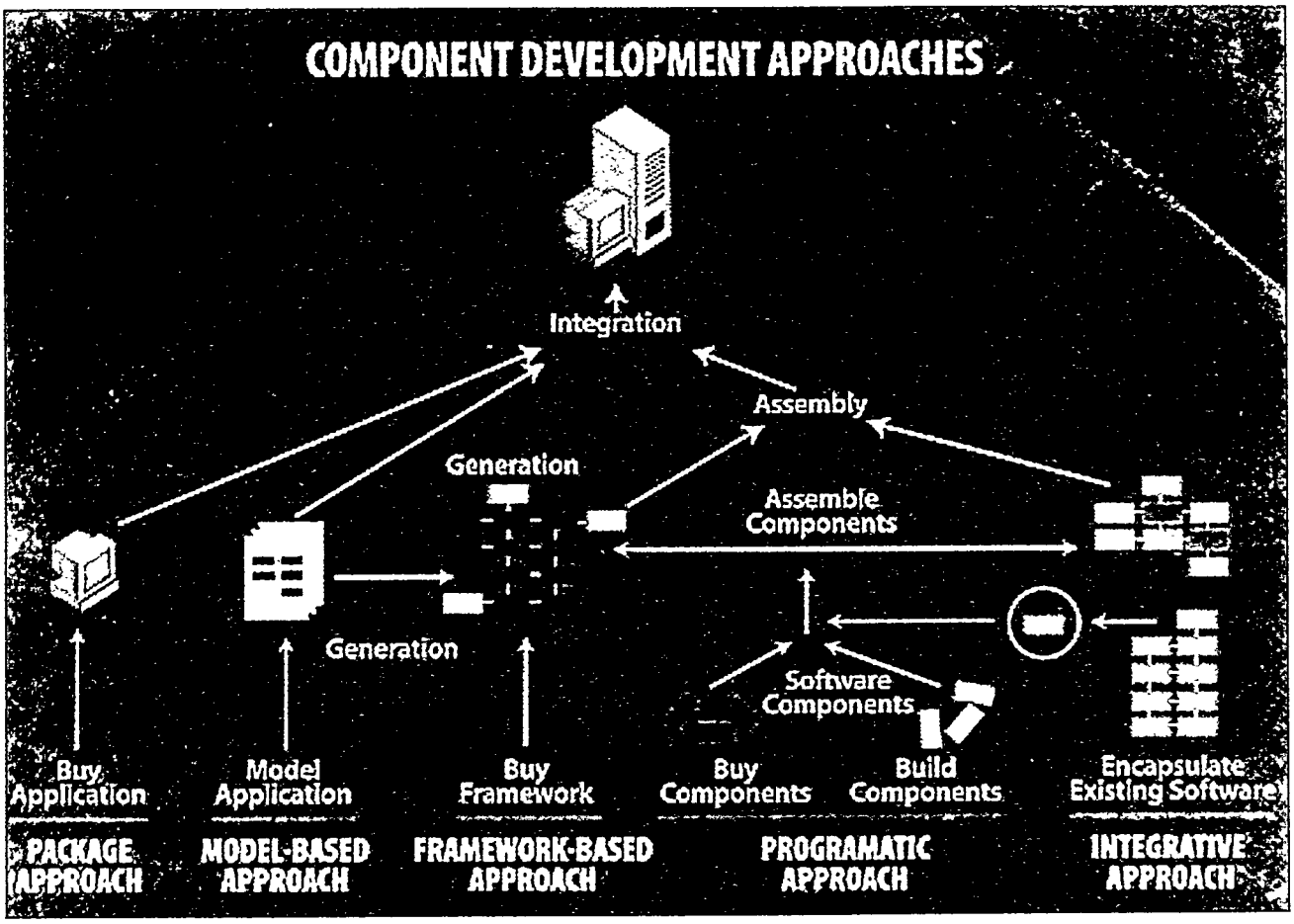

Figure 2 An overview of component development approaches (from Kara, 1998).

- **Programmatic Approach:** applications built using class libraries and encapsulated software objects. The same holds true in those instances where pieces of existing applications are encapsulated and then integrated with other components to form a complete solution.
- **Framework-Based Approach:** generic application programs or subprograms that developers tailor to the own particular needs by adding highly specialized functionality, often in the form of other components, that is called by the framework. The frameworks themselves typically handle control flow, database access, and other low-level functions throughout an application or subsystem. (While Figure 2 indicates that frameworks are purchased, the CTSS framework has been developed in-house.)
- **Model-Based Approach:** modeling tools capable of generating software modules and frameworks for combining components. The model-based approach represents component development taken to a higher abstraction, which increases developer productivity to a greater degree than will using low-level reusable objects.
- **Package Approach:** complete applications that provide a generalized interface that can be used with other components to build the system. One class of applications, the enterprise application packages, now have published interfaces to their suites, so that it is possible to modify and extend these systems using component technology and techniques.

2.2.3 Software Development Approaches

2.2.3.1 Procedural Programming

In a procedural environment, the developer writes an application by making a series of calls to library routines provided by the system (as well as to routines written by the developer as shown in Figure 3). The developer's code sits on top of the system code. The developer's code can access all of the system's services, but the system knows nothing about the developer's code. The developer is responsible for providing the overall behavior and flow of control of the application, with the system providing the functionality.

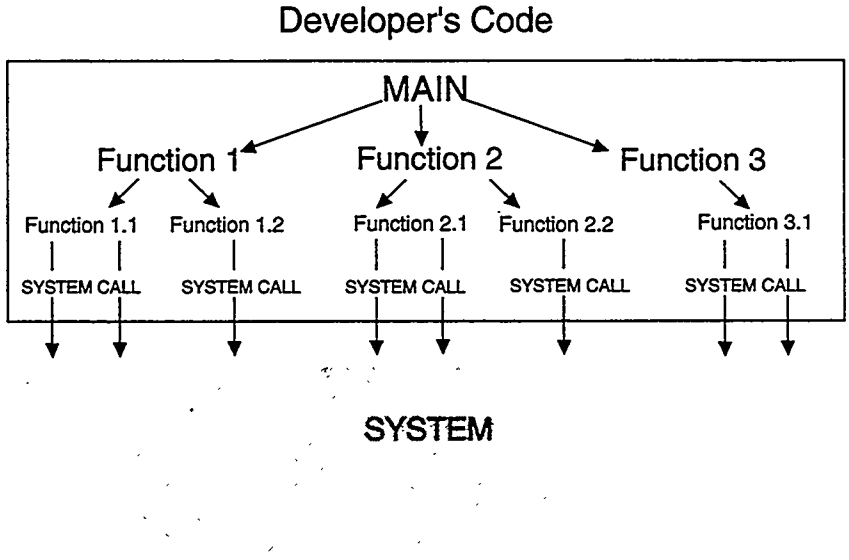

Figure 3 Procedural programming approach.

The procedural approach (also called structured programming) has improved software quality over the last 20 years, but its limitations are painfully apparent, due to the following:

- **Difficulty in extending and specializing functionality.** Because procedural systems do not provide flexible interfaces, developers cannot selectively change or extend the structure or behavior.
- **Difficulty in factoring out common functionality for reuse.** It is difficult to consolidate common system functions for straightforward reuse by other applications. Since it is difficult to factor out the common pieces in a procedural system, functionality has to be duplicated every time a new feature is introduced, resulting in a longer development time.
- **Barriers to interoperability.** Even when extensions and modifications are made in a procedural application, it is hard to ensure that the changes will interoperate correctly with other systems that depend on the modifications. The result is that the solutions from one developer might not have anything in common with any other developer's solutions. *So, instead of a small team of experts solving a particular prob-*

lem once, there are numerous teams repeatedly addressing the same problem.

- **Maintenance overhead.** Since there is minimal reuse of code in a procedural system, maintenance requirements increase due to the greater amount of coding involved and the subsequent increased potential for introducing new bugs.

Lack of extensibility, factorability, interoperability, and maintainability inherent in the procedural approach adds up to lower code and design reuse. The result is that developer productivity is severely hindered since more time and resources are spent writing code instead of solving new problems.

2.2.3.2 Object-Oriented Programming

Although the principles of procedural programming have improved the clarity and reliability of programs, large-scale programming such as that required for the WMS program still remains a challenge. Object-oriented programming (OOP) brings a new approach to that challenge.

Unlike procedural programming, which emphasizes algorithms and procedures, OOP emphasizes the binding of data structures with the methods to operate on the data. The idea is to design object classes that correspond to the essential features of a problem. Rather than trying to fit a problem to the procedural approach of a computer language, OOP allows the programmer to use the language to effectively model and solve real-world problems. A vehicle tracking program, for example, might define classes that represent vehicles that communicate over a QUALCOMM network and vehicles that communicate over an HF network. The class definitions would include common functionality that is the same for each class, such as location reporting and velocity reporting. Then a developer would proceed to design a program by deriving subclasses and overriding existing methods or implementing new methods within each class. Thus, once a developer had taught the software how to display vehicle location and velocity, the procedure would be the same regardless of whether it was a QUALCOMM or HF vehicle. Also, new vehicles can be added by deriving new subclasses and overloading new behaviors but keeping useful old behaviors.

This development approach allows developers to break problems into small, manageable modules of code, where the principle of encapsulation insulates developers from having to know the implementation details. Due to the principle of inheritance, developers can subclass to derive new

classes from existing ones and be provided with the "hooks" to add extensions.

In addition, polymorphism gives the developer flexibility to create multiple definitions for functions. This allows classes to be more general and hence more reusable. It also allows new components and functions to be added easily and without disturbing the existing system. Implementing OOP makes it possible to design software that is more extensible, reusable, and maintainable.

By helping developers design and produce code more productively, the advantages of OOP have proven to be a significant revolution over traditional programming techniques. However, even though the programming job is made easier, since the developer works at a higher level of abstraction with objects and class libraries, the developer still has to "put the pieces together." Simply changing from procedural techniques to OOP does not fix the problem that developers still are responsible for providing infrastructure and are not provided with a clean mechanism for extending functionality. Even with OOP, developers write a lot of code since they are still responsible for providing the flow of control of the application. The object-oriented framework approach and other component development approaches carry the OOP paradigm further by providing infrastructure and flexibility for deploying OOT.

2.2.3.3 Framework-Oriented Programming

We define framework oriented programming as the exploitation of object-oriented frameworks to maximize the benefits of OOT. A widely accepted definition comes from Ralph E. Johnson of the University of Illinois:

"A framework is a set of classes that embodies an abstract design for solutions to a family of related problems." (Johnson, 1988)

An object-oriented framework can be thought of as a prefabricated structure, or template, of a working program.

Because the framework approach provides infrastructure and flexible interfaces, it avoids the problems and overhead that traditional programming imposes on developers. With a well-designed framework, it is much easier to add extensions, to factor out common functionality, to enable interoperability, and to improve software maintenance and reliability.

The way the Object-Oriented Framework approach achieves these benefits over other development approaches are based on two fundamental principles:

- **The framework approach provides infrastructure and design.** An object-oriented framework is not simply a collection of classes. Rather, it comes with rich functionality and strong "wired-in" interconnections between the object classes that provide an infrastructure for the developer. It is these inter-connections that provide the architectural model and design for developers and frees them to apply their expertise on the problem domain. By providing an infrastructure, the framework dramatically decreases the amount of standard code that the developer has to program, test, and debug. The developer writes only the code that extends or specifies the framework behavior to suit the program's requirements. This code can be represented visually as a "puzzle piece" since this is the creative and undefined part the developer provides (see Figure 4).
- **The framework calls you, you don't call the framework.** Framework-oriented programming requires a new way of thinking. In procedural systems, the developer's own program provides all of the structure and flow of execution and makes calls to function libraries as necessary (see Figure 2). However, in framework-oriented programming, the roles are turned around. The role of the framework is to provide the flow of control, while the developer's code waits for the call from the framework. This is a significant benefit since developers do not have to be concerned with the details, but can focus their attention on their particular problem domain. However, this flip-flop in control can be a significant change for developers experienced only in procedural programming. The developer must learn to think in terms of the responsibilities of the objects—what are the objects required to do—and let the framework determine when the objects should do it. Once the investment has been made to understand frameworks, developers will begin to realize the enormous advantages that framework-oriented programming can deliver over other development approaches.

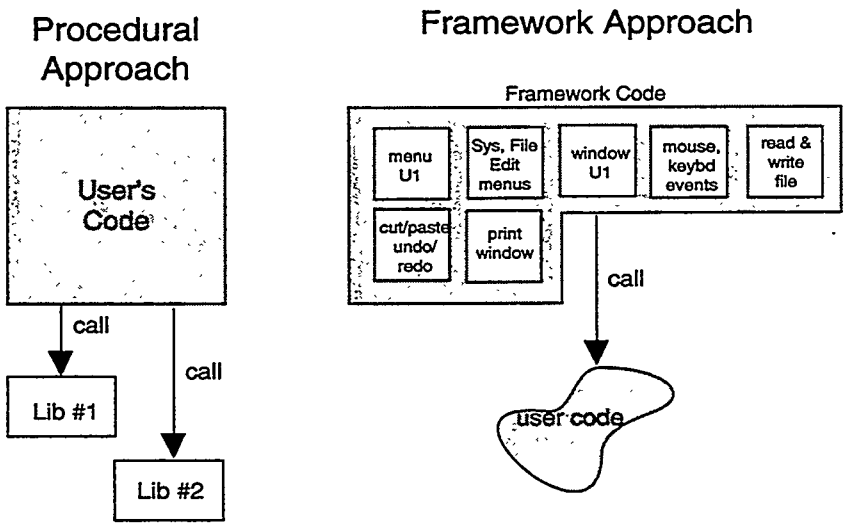

Figure 4 Procedural versus framework approach.

The overall benefit of the object-oriented framework approach is that it enables a higher level of code and design reuse than what is practical with other design approaches. In addition to the framework approach, there are certainly many other reuse technologies such as 4GLs, code generators, and class libraries. However, 4GLs and code generators are based on procedural programming techniques and cannot easily provide the infrastructure and design guidance that are possible from frameworks. While class libraries do improve code reuse, they provide functionality at a very low level and force the developer to provide the interconnections between the libraries. These advantages of the framework approach over procedural approaches and class libraries are much more difficult to create or recreate and constitute the real value of the object-oriented framework approach.

Also, the developer should realize that the benefits from the framework approach and reuse are gained over time, since the productivity gains do not come just from the first or second use, but from multiple uses of the technology. The following summarizes the major advantages:

- **Infrastructure and architectural guidance.** By virtue of the interconnections among the class libraries, much of the needed functionality already exists in the framework, thus reducing coding, testing, and

debugging efforts. In addition, frameworks encourage better design in the code that developers do write by providing an "example" to guide them to more effectively utilize object technology. Applications developed with frameworks tend to be smaller, as well as more maintainable and reusable.

- **Mechanism for reliably extending functionality.** While objects and object classes provide interfaces for extending functionality at a fine-grained level, the framework approach provides this flexibility at a higher level. In this way, applications can be developed by using the framework as a starting point and writing smaller amounts of code to modify or extend the framework's behavior. These extensions can be added without sacrificing compatibility and interoperability because the interfaces are well defined.
- **Reduced maintenance.** Due to inheritance, when a framework bug is fixed or a new feature is added, the benefits of those changes become available more quickly to the derived classes. Also, changes are made only in one place, thus, the chance of introducing additional errors in the code is minimized.

2.2.3.4 Other Component-Development Approaches

As was explained in Section 2.2.2 and illustrated in Figure 2, framework-based programming is just one of a variety of object-oriented component-development approaches. The CTSS project is now assimilating these other component-development approaches to build and integrate more substantive systems.

As Dan Kara explains in his article on component development (Kara, 1998), the real challenge will lie in knowing which components to select and how to integrate them:

"The various component types and approaches have their own advantages and disadvantages, together with their realm of applicability. The real windfall for increasing developer productivity, however, happens when larger-grained components, such as application frameworks, application models, and complete pre-built applications are used and reused for building systems. Unfortunately, the more large-grained the software component, the less flexible and malleable it is, reducing its reuse potential. The best approach to component development and integration, therefore, is to employ higher-order tools that support the manipulation and modification of large-grained objects, and use these in conjunction with small-grained components offering specialized functionality."

SECTION 3 Architecture

3.1 Implementation

The Configurable Transportation Security and Information Management System, which is built using an object-oriented framework and other component development approaches described in Section 2, has the following key features:

- **High level of abstraction.** The larger-grained component-development approaches used in the CTSS project (e.g., the CTSS framework and prebuilt application packages) raise the level of abstraction, allowing less-technical developers to assemble components into sophisticates applications while more knowledgeable developers can focus on creating components. A higher level of abstraction leverages developer investment and accelerates development of new applications.
- **Reuseability.** The CTSS framework is designed to be reusable and grow in complexity. As objects are added to the framework to fulfill the needs of one application, other applications may make use of them. However, modifications made to objects for one application do not necessarily have to be used by other applications (see next bullet).
- **Decoupled design.** The framework is designed to be decoupled. Objects are as independent as possible by design, following standards and conventions. The framework does not require that features, services, or objects required by certain applications be built into each application that may access the framework. This approach allows simple or complex applications to be built using the same framework, and

it simplifies code maintenance and modification. By keeping objects, services, and applications decoupled, enhancements may be made without the common problem of "breaking" (i.e., revising and recompiling) other code.

- **Standard, compatible objects.** Objects follow common conventions, standards, and interfaces to enhance maintainability, reuse, and extensibility.
- **Database management.** Commercial off-the-shelf (COTS) object oriented database management systems (OODBMSes) and relational database management systems (RDBMSes) will provide much of the cross-platform data management and data translation. Currently, CTSS supports the OBJECTSTORE and VERSANT OODBMSes and the Oracle RDBMS.
- **Multiple platforms supported.** The framework is intended to be supported on the Windows/NT, Windows 95/98, and UNIX platforms. These environments support fairly complete compilers and a full list of development tools.
- **Designed-in configurability.** The framework is designed to be configurable. Multiple applications can be built using the same framework.

3.2 Domains

The Fleet Management System is broken into the following domains:

- **Command Center domain.** The command center domain includes the central site and the activities that take place there: database maintenance, report generation, resource allocation, communication, incident management, mapping, information management, vehicle interface, and operator interface.
- **Vehicle domain.** The vehicle domain includes the mobile application, location determination, resource and sensor monitoring, emergency notification, driver interface, and trailer interface.
- **Trailer domain.** The trailer domain includes sensor monitoring, countermeasures, vehicle interface, and the container interface.
- **Container domain.** The container domain includes the trailer interface and materials monitoring.

The value of architecture is in decomposing a system into logical units. Taken as a whole, a fleet management system is a complex problem. By breaking it down, each piece is more comprehensible and doable task to implement. Figure 5 graphically depicts the domains.

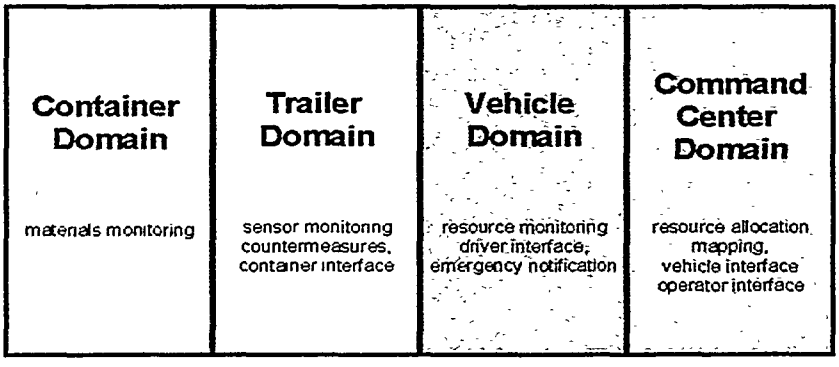

Figure 5 Domains of a Fleet Management System

3.3 Functional Levels

Once divided into domains, each domain can be further divided. CTSS selected a layered architectural model for the command center and vehicle domains. This model builds in an isolation between non-touching layers.

The CTSS Framework consists of the following four functional levels:

- Application Level
- Framework Level
- Middleware Level
- Platform Level.

The functions of each of these levels are introduced in Figure 6.

The lower three levels, which comprise the software base, are designed for re-use by multiple transportation applications. The framework is

object oriented and has been implemented in C++ (command center domain) and Visual Basic (vehicle domain). The framework is making use of commercial off-the-shelf software as much as possible. The framework has been implemented initially on both a popular UNIX platform (command center domain) and the Windows/NT/95/98 platform (vehicle domain).

The architecture for the framework and middleware levels is the same in the vehicle domain as it is in the command center domain. However, in these two levels, some objects and services may not be necessary for a given domain, and vehicle domain objects and services may complement those in the command center domain. Therefore, in keeping with the reusability goal of this project, the framework and middleware layers of this architecture in the vehicle domain is a scaled subset of the same layers within the command center domain. Whereas the command center domain can be implemented across multiple hosts, the initial instantiation of the vehicle domain is scaled to a single host. However, no impediments to a multiple-host vehicle system should exist given the scalability of this architecture. If a multiple-CPU platform were chosen for the vehicle domain, this architecture should be easily scaled to such a platform.

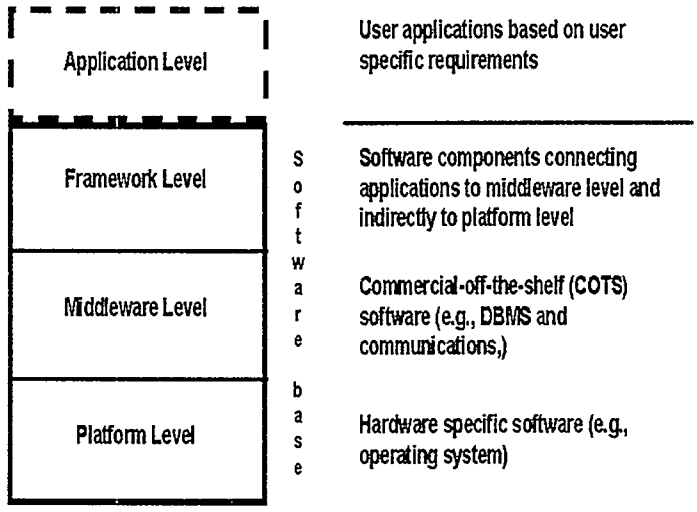

Figure 6 CTSS Framework functional levels.

3.3.1 Application Level

The Application Level consists of high-level applications written by the user. This level makes use of common objects, application standards, and communication between applications provided by the Framework Level services to access the encapsulated transportation capabilities provided by the Framework level and the links to the Middleware and Platform Levels.

3.3.2 Framework Level

3.3.2.1 Encapsulated Transportation Services

The framework level has objects that provide encapsulated transportation services. These objects are used from the Framework Class Library such as Vehicle, Container, and so on.

3.3.2.1.1 Vehicle Domain Encapsulated Transportation Services

The following encapsulated transportation services are being implemented as the framework architecture in the vehicle domain:

- **Fleet data communications.** Two-way data communications are being implemented in the vehicle domain using the QUALCOMM and ORBCOMM satellite systems and a satellite phone system. Voice communications are not currently supported.
- **Real-time tracking.** An automatic position-location (APL) system is being implemented in the vehicle domain using a GPS unit. The vehicle domain stores the latest vehicle position, speed, and heading, but does not keep a location history. Status of the GPS unit is stored as well. All of this data is transmitted to the command center domain via the fleet data communications application and is displayed to the user in both the command center and vehicle domains upon request.
- **Trailer monitoring/communications.** An interface to a transportation data unit (TDU) that monitors containers as they are transported in trailers and that provides the destination material management system with an inventory and status of assets that were transported is currently being developed.

These applications complement the applications to be developed in the command center domain. Objects and daemons are being reused across domains wherever possible, but specific needs within the vehicle domain do require some differences.

3.3.2.1.2 Command Center Domain Encapsulated Transportation Services

The following applications are being implemented as a proof of concept of the framework architecture in the command center domain:

- **Fleet data communications.** Two-way data is being implemented in the command center domain using the QUALCOMM and ORB-COMM satellite systems and a satellite phone system to store and forward messages.
- **Mapping/tracking.** A subsystem is integrating two-dimensional mapping to allow tracking of multiple types of vehicles of any number (or quantity).
- **Messaging.** The following functionality is supported:
  - transmission and reception of canned messages
  - composition of messages using a modern text editor
  - display of vehicle and trailer objects and their data.

3.3.2.2 Framework Services

The Framework Level also provides application-independent objects (See Figure 7). These patterns provide applications with service-oriented functions such as interprocess communications, distributed logging, diagnostics, and system status.

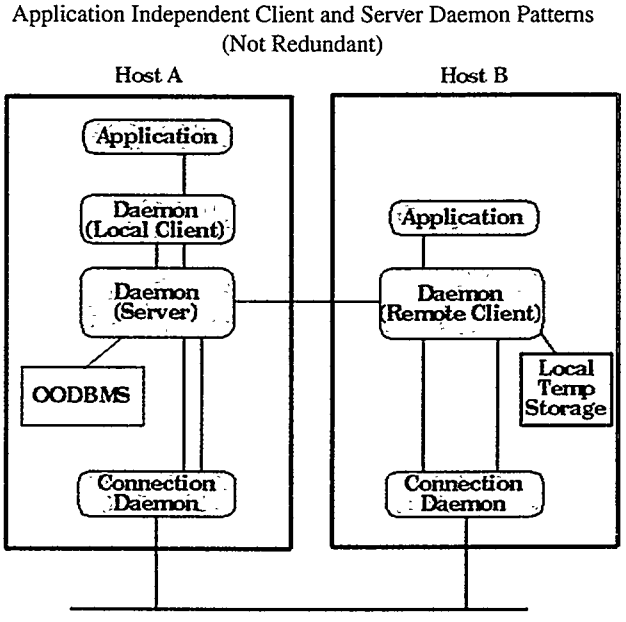

Figure 7 Multiple host non-redundant client/server daemon pattern.

The Framework Level provides the interface to Middleware software as necessary. The Framework Level provides objects to encapsulate data such as the CTSSMessage object. Some applications, or servers, may be reused entirely from system to system.

The architecture of this layer in the vehicle domain is complementary to that of the same level in the command center domain.

3.3.3 Middleware Level

The Middleware Level is COTS system integration software, which may be purchased or shareware. Middleware generally is broken into distributed processing software (lower) and database and user interface software (upper). The middleware chosen is object-oriented.

The architecture of this layer in the vehicle domain is comparable to that of the same level in the command center domain.

3.3.4 Platform Level

The Platform Level consists of the vendor-supplied software and hardware. The initial platforms will be Windows/NT/95/98 and UNIX. These platforms are POSIX compliant to a degree. The Platform Level also consists of the development software, editors, compilers, debuggers with memory leak detection, source control software, and GUI testers. Third-party software is required to run on a variety of platforms.

This page left intentionally blank.

SECTION 4 Design Detail

4.1 Command Center

4.1.1 Framework Level

The transportation-related functions that were selected for development in the object-oriented framework were

- communication
- container status
- information management
- mapping
- message handling
- tracking
- trailer monitoring
- incident management
- system diagnostics
- system configuration
- operator interface

These functions span the Vehicle and Command Center domains. The framework is scalable from one vehicle and command center to multiple command centers and vehicles.

These services require wide area communication with actual vehicles in the vehicle domain, communication link handling, interprocess communication, an expandable real-time mapping function, and distributed persistent information management.

The Framework Level encapsulated transportation objects are not directly aware of what platform they are running on. The platform is transparent to the application because the Platform Level will be shielded by the Middleware Level (Section 3.3).

Some objects from the MAP/Tracker project have been reused as the basis for the encapsulated transportation objects (i.e., objects from the Framework Class Library).

4.1.1.1 Framework Layer Objects

4.1.1.1.1 Communication Objects

CTSS supports several objects for communication mechanisms. These objects can be used for cross-domain or intra-domain communication.

Table 1 CTSS Communication Objects

Object Name	Object Description
Fifo	UNIX POSIX-named pipe
HayesModem	Hayes Communication protocol implementation
Orbcomm	ORBCOMM message
Qualcomm	Qualcomm message
SerialPort	UNIX and Windows NT implementations of a serial port
TeleComm	Telephonic communications

4.1.1.1.2 Container Status Objects

CTSS supports the ability to interface to a FMS container domain.

Table 2 CTSS Container Status Objects

Object Name	Object Description
Sensor	Sensor id, location, type, state

4.1.1.1.3 Information Management Objects

CTSS supports the ability to persistently maintain information on transportation objects that can be used by decision makers.

Table 3 CTSS Information Management Objects

Object Name	Object Description
Contact	Several classes of who to contact
Convoy	Id, Member vehicles, Commander
Shipment	Physical instance

4.1.1.1.4 Mapping Objects

CTSS supports several objects for mapping. These objects can provide data on instances that can be mapped or can provide the messages which support mapping actions.

Table 4 CTSS Mapping Objects

Object Name	Object Description
Airport	Complete DOT airport information
Hospital	Location, contacts, beds, ER wards
MapTrackerMessage	Message formatted for MAP/Tracker map display
TrackingAnalystMessage	Message formatted for the Tracking Analyst extension of ESRI's ArcView product
TLPInterface	Interface to TracerlinkPro map display software

4.1.1.1.5 Message Handling Objects

CTSS supports several objects to support messaging by the application. It provides a generic keyword-based message. A cipher object can encrypt or decrypt a message. The CTSSMessage object can package and send the message for interprocess communication.

Table 5 CTSS Message Handling Objects

Object Name	Object Description
Cipher	Encryption/Decryption capability
CTSSMessage	Interprocess message delivery mechanism
KeywordMessage	Generic message format

4.1.1.1.6 Tracking Objects

CTSS supports two tracking objects. CTSS is designed to track vehicles based on a latitude/longitude position.

Table 6 CTSS Tracking Objects

Object Name	Object Description
LatLong	Latitude and Longitude position in several representations
Vehicle	Id, location history, events, state

4.1.1.1.7 Trailer Monitoring Objects

CTSS supports an interface to trailer domain.

Table 7 CTSS Trailer Monitoring Objects

Object Name	Object Description
Container	Vehicle container

4.1.1.2 Framework Level Services

The Framework Level provides application-independent service patterns for

- interhost and intrahost connection management
- distributed logging

The generic daemons may be on one host or multiple hosts. The Framework Level services may or may not use an Object Oriented Database Management Systems (OODBMS) or an Object Request Broker (ORB), depending on the number of hosts and the complexity of the system that must be generated.

A system may use all or some of the services. Framework Class Library objects are available to build generic applications.

Interprocess communication is a service that is both interprocess and interplatform. This requires a connection daemon or gateway to authorize connection requests and a message transmission object to encapsulate data.

COTS software (Section 4.1.2) provides the actual multi-platform communication middleware. This also provides the multiple platform support required.

Applications that desire interprocess communications, for instance, instantiate an object that requests a service from a server. Message objects are then sent/received on the socket object from the server. Servers build on the ACE (Active Communications Environment) middleware described below to provide multi-host servers that authorize socket communication requests, provide message transmission of the CTSSMessage class using sockets, and encapsulate shared memory objects.

4.1.1.3 Framework Daemons

4.1.1.3.1 Gateway Daemon

A required gateway communications daemon, or connection daemon, provides a single host-to-host communication path. The daemon also provides intrahost process-to-process connections. This daemon is built using the ACE package. It provides authorized connections between hosts. Message encryption is optional. The gateway provides a single point of connection between hosts for event messages and object updates. Host connections are used to transmit events and object updates. Processes then look up complete objects in the object databases. The gateway daemon processes CTSSMessage objects. The gateway daemon also has an optional message filtering capability.

Cross-platform connection objects could be instantiated in every program, rather than the gateway, but we choose to use *one* connection daemon. The advantages to having one connection daemon is that it improves program control, security control, and extensibility. It is easier to demonstrate that the one connection daemon is configured correctly than having to monitor all socket connections from every process.

Typically, most objects reside in a database and are accessed in a client/server fashion. However, socket transmissions are used to transmit notification of events that may be applied to an object, such as database object changes, events, errors, and state changes. Examples include **VehicleUpdate**, **ErrorText**, and **Text** messages.

The gateway daemon connects to local processing using objects from the ACE package (Section 4.1.2). These objects are either local authenticated sockets or named pipes. These are streamed objects to provide some buffering.

The gateway daemon has a configurable host-to-host buffering capability by storing messages to disk or memory. When a connection is then made, the pending messages, if desired, are delivered. The buffering may be zero if thus configured.

Connections to the gateway are made using connection objects provided by the Framework Level. Each connection is maintained by a separate POSIX thread so that multiple connections can be handled.

To make expansion and configuration changes easier, all application messages on a host go through the one gateway. To add a new application that needs to connect to other applications, only the gateway's configuration file is updated, and the gateway can be reset to read the data. This approach keeps applications from having to know about each other. By default, all messages go to all applications on a host. Since this may be costly, the filter file identifies what message types go to what applications for the gateway's routing data.

A required configuration file or database contains the remote connection information, local process connections, port addresses, types of connections, and retry and store information. Types of CTSS**Message** objects to pass on each connection are optional.

4.1.1.3.2 Logger Daemon

An optional distributed logger daemon managed the application log database. If there are multiple loggers in the generated system, one logger daemon is the global master and is responsible for keeping its database complete and up to date. Applications connect to a local logger client. The local applications find a logger server that actually stores the message in persistent storage.

4.1.1.4 Framework Level Platform Independence

The Framework Level, which is intended to be platform independent, provides support to common UNIX systems based on System V UNIX and to Windows 95/98/NT. The platforms should be POSIX compliant so that POSIX threads and numerous other functions such as time standards can be supported more easily.

Hardware differences are a bigger problem. Big-endian versus little-endian problems (i.e., differences in where most- and least-significant bytes are stored in multi-byte numeric representations; see Glossary in Section 1.3) can be handled initially in several ways. Most data transfers will be done by sharing databases using the DBMS or an ORB. These packages handle the platform differences already. Data or event transfer

objects that do not go through middleware objects (i.e., transferred directly through pipes, serial ports, ftp, and so on) can use the XDR standard to convert themselves to the standard format. Ultimately, it is hoped that an automatic object decomposer middleware can be found to read header files and produce the object layouts for all platforms. Some coding standards have to be followed to eliminate coding unportable code such as C's bit fields.

4.1.2 Middleware Level

Middleware is COTS software that provides system integration software for object-oriented communication and object-oriented databases on multiple platforms.

4.1.2.1 Communication Software

The communication software provides a wide spectrum of functions to provide the required communication between processes. The Adaptive Communications Environment (ACE) is shareware written in C++ and JAVA and is supported on multiple platforms. ACE provides the following communication services:

- Connection configuration (Each host has a server that verifies connection validity.)
- Interprocess communication (shared memory, mapped shared file objects)
- Named pipes (same host)
- Socket communication (This type of communication occurs across hosts, and allows transmission of limited data (i.e., transient messages, authentication requests, and so on) that are not supported by an OODBMS. Data transmitted is encapsulated in the `CTSS`**Message** class. This code performs the actual connection maintenance, reconnection, error logging, and so on.)
- Time services (multiple host/platform time synchronization)
- Event notification (multi-threading, dynamic reconfiguration of services, and so on).

4.1.2.2 DBMS

Object Oriented Database Management Systems (OODBMSes) provide a persistent, client-server architecture for the storage of objects. Much expertise has been acquired with the **ObjectStore** OODBMS in the MAP/Tracker project. Persistent storage of objects such as **Vehicle**, **Convoy**, **Events**, and **Shipments** are accomplished with **ObjectStore**. **ObjectStore** provides the cross-platform mapping of C++ objects to deal with the incompatibilities of various C++ compilers and hardware data formats. One or

more OODBMS servers now exist; typically one process updates a database, and many clients read the data. These clients may be on multiple hosts and access the data correctly. Objects are stored directly into an OODB using whatever data structures desired.

To validate the flexibility of the CTSS middleware/framework layer interface, two additional DBMS systems were incorporated. The OODBMS, Versant, and the object-relational DBMS, Oracle were added. Versant was chosen since it had a large percentage of the OODBMS market and Oracle is the leader in the DBMS market.

4.1.3 Platform Level

COTS products do provide many functions, but they generally do restrict the choices of other software and hardware that can be integrated. Because it generally drives what other software is compatible and supported, COTS software must therefore be carefully selected. POSIX compliance implies support of sockets, named pipes, and threads and many other features. The degree of compliance varies from system to system so some platform-dependent conditional code may be necessary. Most Middleware shields this level from the Framework Level. Java is rapidly becoming a viable choice given its cross platform nature and built-in user interface capability. However, at this time C++ will be used as the primary language due to its maturity, COTS APIs, third party packages, and libraries.

Initial development is being done on the following platforms:

- **Windows/NT.** This is the most "industrial strength" of the multiple Microsoft operating systems. It claims POSIX compliance to support necessary threads and **WinSock** libraries. The selected Middleware is supported.
- **UNIX.** UNIX is the still the system of choice for applications requiring an around the clock operation. Its maturity and breadth of developer services make it a viable platform for the command center.

The following compilers and tools will be used:

- C++
- runtime debugger

The C++ Standard Template Library (STL) is being used rather than writing much code from scratch as is often the case. STL is supported on many platforms including Microsoft and most UNIX platforms.

4.1.4 Software Configuration Management

A single software repository is available on host SAIX2965 using the CVS version control package.

4.2 Vehicle

4.2.1 Framework Level

The transportation-related functions that were selected for development in the object-oriented framework were

- communication
- information management
- message handling
- tracking
- trailer/cargo monitoring
- incident management
- system diagnostics
- system configuration
- operator interface

4.2.1.1 Framework Layer Objects

These objects were constructed using Microsoft Visual Basic classes and collections. While application independent, they are specific to a PC platform running Microsoft Windows 95 or 98. All of the objects are also compatible to Windows NT 4.0, with the exception of the clsParallelPort object.

4.2.1.1.1 Communication Objects

The CTSS vehicle domain supports three primary communication mediums.

Table 8 CTSS Communication Objects

Object Name	Object Description
clsQCTransceiver	Qualcomm satellite communications
clsTeleComm	Telephonic communications (Hayes-modem compatible)
clsOCTransceiver	Orbcomm satellite communications
clsSlipPort	Serial Line Interface Protocol (SLIP) packet-based port
clsOCTPort	Orbcomm packet-based port

4.2.1.1.2 Information Management Objects

CTSS supports the ability to persistently maintain information on transportation objects that can be used by decision makers.

Table 9 CTSS Information Management Objects

Object Name	Object Description
clsVidEntry	Allows user entry and storage of the vehicle ID
clsRealTimeClock	Provides timestamping for all messages and logs.

4.2.1.1.3 Message Handling Objects

CTSS supports several objects to support messaging by the application. It provides a generic keyword-based message. A crypto-object can encrypt or decrypt a message.

Table 10 CTSS Message Handling Objects

Object Name	Object Description
clsMessage	Generic message object
clsTextMessageEntry	Generic text-message entry object
clsDesCrypto	DES Encryption/Decryption capability
clsMessageProcessor	Keyword-based message construction/deconstruction

4.2.1.1.4 Tracking Objects

The CTSS vehicle supports four tracking objects. CTSS is designed to track vehicles based on a latitude/longitude position. These objects also provide real-time-clock information for the vehicle computer system.

Table 11 CTSS Tracking Objects

Object Name	Object Description
clsTrimbleGps	Implementation of a Trimble GPS receiver interface
clsNMEAGps	Implementation of a NMEA/Delorme GPS receiver interface
clsSimulateGps	Implementation of a GPS simulator for laboratory demonstrations and testing.
clsOCTransceiver	Implementation of an Orbcomm internal GPS interface
clsSerialPort	Implementation of simple serial communications for GPS.

4.2.1.1.5 Trailer/Cargo Monitoring Objects

The CTSS vehicle supports an interface to trailer/cargo domain, with the capability to auto-configure to different trailer types.

Table 12 CTSS Trailer/Cargo Monitoring Objects

Object Name	Object Description
clsTrUhfMgr	Master interface object to all trailers/cargo
clsSgtTrailer	Implementation of interface to the Transport
clsTrailerEvent	Generic trailer event object
clsContainer	Generic object for cargo container (supports INUMM and T1 tag types)
clsContainerEvent	Generic container event object

4.2.1.1.6 Incident Management

The CTSS vehicle supports several objects to provide the capability to handle incidents in the field.

Table 13 CTSS Incident Management Objects

Object Name	Object Description
colPriorityQueue	Priority-based message queue
clsPriorityContainer	Priority-based message holder for colPriorityQueue
clsDigitalInput	Generic digital input for emergency switches, etc.
clsDigitalOutput	Generic digital output for warning lights, etc.
clsParallelPort	Generic interface to parallel port for digital I/O.

4.2.1.1.7 System Diagnostics

The CTSS vehicle supports two objects to provide system diagnostics for easy maintenance of the vehicle hardware and software.

Table 14 CTSS System Diagnostics Objects

Object Name	Object Description
clsLog	Object for logging all vehicle events in a chronological fashion to an ASCII file
clsInterfaceMonitor	Object for monitoring hardware interfaces at a binary level.

4.2.1.1.8 System Configuration

The CTSS vehicle supports two objects to provide easy system configuration and reconfiguration from an initialization file.

Table 15 CTSS System Configuration Objects

Object Name	Object Description
colInitKeys	Collection of keyword/value pairs representing an initialization file
clsInitKey	Object containing a single keyword/value pair (e.g. TRAILER_PORT=3)

4.2.1.1.9 Operator Interface

The CTSS vehicle supports several objects to provide easy building of operator interfaces to vehicle computer applications.

Table 16 CTSS Operator Interface Objects

Object Name	Object Description
clsVidEntry	Allows user entry and storage of the vehicle ID
clsTextMessageEntry	Allows user entry of text messages
clsAgent	Provides speech synthesis and recognition
clsSound	Provides audible alarms using simple WAV files
clsPwrControl	Provides safe-shutdown of the vehicle computer
clsMsgBox	Provides simple information displays to the operator
clsMsgBoxYesNo	Provides simple yes/no prompts to the operator
clsAlarm	Provides alarm displays to the operator

4.2.1.2 Framework Level Components

The Framework Level provides application-independent components for

- application configuration and initialization
- communications to/from the command center domain
- communications to/from the trailer/cargo domains
- tracking/real-time-clock data from a variety of sources
- event logging

- digital I/O (e.g. emergency switches, warning lights)
- speech recognition/synthesis

A system may use some or all of the components. Communication and data transfer between different components is handled in the application layer and varies from application to application.

4.2.2 Middleware Level

Middleware is COTS software that provides commercial implementations of serial ports, telephonic communications, hardware interfaces, etc.Several COTS components were encapsulated within the Framework objects.

4.2.2.1 Process Control

Microsoft Visual Basic timers provided the multi-tasking used by each component in the framework. One timer was allocated per component, with a maximum of 6 timers per any single application.

4.2.2.2 Communication Software

The Microsoft Comm Control included with Microsoft Visual Basic provided an interface to standard PC serial ports and was used by most of the interfaces requiring serial communications.

Crescent PDQComm provided modem control and file transfer for telephonic communications and was an extension of the Microsoft Comm Control. It was purchased from Crescent separately from Microsoft Visual Basic.

4.2.2.3 Operator Interface

Microsoft Agent provided speech recognition and synthesis for operator interactions with the application. This component was distributed with Microsoft Windows 98 and was also freely available from Microsoft for use with Windows 95 and NT.

4.2.2.4 Encryption Package

The Bokler Software Corporation's encryption package, DESCipher/OCX was used for cryptography rather than developing custom cryptographic code. This package meets the FIPS Publication 46-2 requirement for the DES algorithm and has been NIST-certified.

4.2.3 Platform Level

Development was done initially on PCs running Microsoft Windows 95 and NT 4.0 and using Microsoft Visual Basic 5.0. The operating system was eventually migrated to Windows 98 due to NT restrictions on access

to hardware interfaces, particularly the PC parallel port. During the course of the CTSS project, the compiler was migrated to Visual Basic 6.0.

4.2.4 Software Configuration Management

The archive for the vehicle domain resides on sass2753 at this time. Initially, Microsoft Visual SourceSafe was used in conjunction with Microsoft Visual Basic to provide version control and configuration mangament. Microsoft Visual Basic projects are structured in such a way that general version control systems cannot be used with it. Visual SourceSafe was found, after some use, to be unreliable as a control package (it corrupted several projects and failed to archive the latest copies of the source code correctly). Another package is being sought at this time that will provide reliable archival and also work within the Visual Basic project structure.

APPENDIX A

Vehicle Framework: Development History and Lessons Learned

A.1 Initial Studies

At the beginning of the CTSS project, initial studies were conducted to determine the best platform, operating system, and programming language to use for the vehicle framework. Original plans for the framework were to make it portable to several different hardware platforms (including PCs, workstations, and embedded computers) and operating systems (including Unix, WindRiver VxWorks, and Microsoft Windows) using individual processes for each component.

The processes in each component were to communicate using the Adaptive Communications Environment (ACE) to provide a portable interprocess communications mechanism. Initially, the project team was able to make ACE work properly only on Unix platforms, and not on Microsoft Windows NT. (It was not advertised to run at all on Microsoft Windows 95 or 98.) Several computers were considered for the vehicle computer, but none ran Unix or Microsoft Windows NT, and political concerns prohibited the use of Unix in the vehicle environment. Therefore, ACE was not a viable option for the vehicle computer. Because of these concerns, the project team decided to limit the vehicle platform and operating system to PCs running Microsoft Windows 95 or 98, using a single-threaded application.

Some time was spent attempting to perform efficient serial communications using Microsoft C++ and a third-party product, GreenLeaf Com-

mLib. (This was attempted because Microsoft Visual C++ contained no built-in components for serial communications other than the same one used in Visual Basic.) After repeated failures, Microsoft Visual Basic was chosen as the programming language, primarily because development of serial communications and GUIs were extremely easy under this programming language, and because, with the advent of Visual Basic 5.0, classes and objects were introduced into the language.

A.2 Vehicle Framework, FY97

A.2.1 Development

The first year of framework development for the vehicle focussed on communications between the vehicle and the command center and navigational tracking of the vehicle from the command center.

A.2.1.1 Communications: The QualComm Component

For communications, the first medium chosen was the Qualcomm satellite messaging system. This system uses a store-and-forward packet network, and had already been used successfully in the production system for several years. Implementation of the interface to this system followed lines similar to that used within the Vehicle Interface Controller, but the software was written as an encapsulated, reusable object within Visual Basic. The interface between the QualComm Mobile Communications Terminal (MCT) and the vehicle computer used RS232C protocol and was in a Serial Line Interface Protocol (SLIP)-based binary format. The specification of the interface itself was proprietary and was obtained from QualComm under a non-disclosure agreement.

A.2.1.1.1 Middleware

Microsoft Visual Basic proved to be particularly easy to use for development of the serial interface, but bit manipulation under the Visual Basic environment was somewhat limited. (Visual Basic did not contain any bit-shifting functions, so these had to be written to process the binary data to/from the QualComm MCT.)

Two middleware objects were used from Visual Basic to implement the platform interface: the Microsoft Comm Control and Visual Basic timers. Several problems were encountered with these middleware objects. "Wrappers" were used to encapsulate the objects wherever possible to mitigate these problems.

The Microsoft Comm Control is a component included within Visual Basic which allows easy reading and writing to a standard RS232C serial port. A serious defect was found in the Comm Control which caused all null bytes (0) received from the serial port to be thrown away, even if the Comm Control's "null-discard" property was set to false. However, the Comm Control would still indicate the null bytes in the count of bytes received on the serial port. This defect was corrected by the wrapper placed around the Comm Control.

Visual Basic timers are components included within Visual Basic that can be set to periodically execute code on an adjustable interval. Initially, several Visual Basic timers were used to perform periodic tasks, including polling the MCT for status and incoming messages and transmitting outgoing messages on 3 to 5 second intervals. As more interfaces were implemented, it was found, contrary to assertions in Visual Basic documentation, that no more than 6 Visual Basic timers can be included in any single application. If more than 6 timers were used, the timer system within the application began to malfunction (e.g. lock-ups, unreliable timing). When this was discovered, the development team decided to limit the application to one Visual Basic timer per interface, plus one or two for the top-level application. This limits the entire application to no more than 5 interfaces which use timers. An alternative that was considered but never implemented was to write each framework component as a separate single-threaded application, which would allow each component to use up to 6 timers. However, it was recognized that this implementation would require the application layer to communicate with the framework layer via slower interprocess communications. The development team decided not to implement this approach; instead, it was held in reserve should circumstances require its use.

A.2.1.1.2 Framework

At this time, the first attempt was made at "standardizing" the application interface to be used by this and all future communication framework components. These interface properties/functions included 1) sending a message, 2) receiving a message, and 3) communications status. Other interface functions were provided to the QualComm framework object, but these were specific to QualComm, and their use in the application layer would interfere with the interchangeability of the application with other future communication framework components.

A.2.1.2 Tracking: The Trimble Component

For navigational tracking, the first unit chosen was the Trimble Placer-400 GPS unit. Again, this unit had already been used successfully in the production system for several years, and implementation of the software interface to this unit was also done as an encapsulated, reusable object. The interface between the Trimble unit and the vehicle computer was an RS232C packet-based ASCII protocol. The specification of this interface was provided freely by Trimble. Since the Trimble GPS interface was packet-based ASCII characters, no bit manipulation was required, making implementation fairly easy and straightforward in Visual Basic.

A.2.1.2.1 Middleware

The problems encountered with the Microsoft Comm Control while developing the MCT-vehicle computer interface were not a factor in development of the Trimble-vehicle computer interface, since the interface was entirely in ASCII and no null (0) bytes were used. A single Visual Basic timer was used to poll the serial interface for data, just as was finally used in the QualComm component.

A.2.1.2.2 Framework

As with the QualComm component, a first attempt was made at "standardizing" the application interface to be used by this and all future tracking framework components. These interface properties/functions 1) latitude, 2) longitude, 3) speed, 4) heading, and 5) time.

The only problem encountered during the development of the Trimble component was determining how to parse the Trimble packet data received from the serial interface. GPS devices, including the Trimble, usually broadcasted various packets at 4800 baud and tended to overrun the ability of a receiving computer to process them. Finally, it was decided to grab blocks of bytes from the serial interface and search them only for the desired packets. Thus, not all the packets were or needed to be individually parsed. This corrected the performance problems seen within the component, while still providing timely updates of the tracking information (once every 2-3 seconds).

A.2.1.3 Other Components

Some smaller components were developed to support the primary communication and tracking components. For example, to allow text-message entry from the application layer, an already existing Visual Basic

form for entry of text messages was encapsulated into a small "text-message-entry" component.

A.2.1.4 Integration: The Vehicle Computer

The Qualcomm component and the Trimble component were implemented in parallel by two different developers, each of whom wrote a simple application-layer to test each component. The two components were integrated using a single, simple application layer which would serve as the first vehicle computer.

A.2.2 Evaluation

One of the primary goals of CTSS with respect to the vehicle domain was to make development of vehicle computer applications quick and easy. The integration and implementation of the first vehicle computer took less than 1 hour.

Following its initial implementation, the vehicle computer application was tested on several PCs running Microsoft Windows 95. Before running the vehicle computer application, the system CPU was, on average, less than 1% loaded. With the vehicle computer application running (using two serial interfaces, the system CPU was, on average, approximately 65% loaded. (Measurements were taken using the performance monitor included within Microsoft Windows, and no other applications were running while the vehicle computer application was measured.) As an experiment, the application was also measured running under Windows NT on a 200MHz Pentium I system. The results were similar. However, given the PC/Windows architecture, which is designed for general office use and not for large amounts of serial I/O, this was to be expected. This system was demonstrated to management during the 4th quarter of FY97.

A.3 Vehicle Framework, FY98

A.3.1 Development

The second year of framework development for the vehicle focussed on alternate and encrypted communications between the vehicle and the command center, alternate navigational tracking, and interfaces between the vehicle computer and the trailer and cargo domains.

A.3.1.1 Communications: The TeleComm Component

Another communications component (referred to as the TeleComm component) was developed specifically for telephonic communications using Hayes-compatible modems. In a mobile environment, this was expected to be used with cellular modems or satellite telephones containing modems. With telephony, the communications link was a point-to-point, session-based methodology, which differs greatly from the store-and-forward packet network within the QualComm system.

A.3.1.1.1 Middleware

The TeleComm component encapsulated a third-party Visual Basic add-on called PDQComm, developed by Crescent. This was an expanded version of the Microsoft Comm Control included with Visual Basic. (Crescent wrote the Comm Control for Microsoft, who included in the Visual Basic software.) PDQComm contained all the features of the Microsoft Comm Control, plus the ability to perform file transfers using various protocols, such as Xmodem, and the ability to easily process serial/modem events (such as loss of carrier).

As development of the TeleComm component proceeded, another serious limitation of Visual Basic timers was encountered. Use of Visual Basic "MsgBox" forms caused all Visual Basic timers within the vehicle computer application to "freeze" while the form was displayed. This "feature" is noted in the Visual Basic documentation, and the same documentation suggested that concerned developers construct their own "MsgBox" forms to prevent this from happening. The project team did, and discovered that any "modal" Visual Basic form caused an intermittent failure of all the timers within the same application. (A "modal" form is one that stays on top of all other forms and prevents the operator from using any other form in the application while it is displayed.) Also, it was discovered that any kind of "wait" loop within the application layer could cause the same failure. At this point, both the vehicle computer application developed from the previous year and the SATCAM/FIRST application were modified to prevent this failure from occurring.

A.3.1.1.2 Framework

The TeleComm component was designed using the "standard" interface between the application and the communication component that was set down during development of the QualComm component. The TeleComm component implemented the standard functions: 1) sending a message, 2) receiving a message, and 3) communications status. Since telephonic

connections are by nature point-to-point sessions, a feature specific to the TeleComm component included: the ability to automatically dial up a remote computer on a specified interval. In addition to the messaging capabilities of the Telecomm component.

At this point, fixed-length FIFO message queues were added to both the TeleComm and QualComm components to allow the application layer to send messages without having to wait for previous ones to be transmitted, and to allow incoming messages to stack up until the application layer was ready for them. The queues were designed to handle a maximum of 200 incoming and 200 outgoing messages. A fixed length was used because, at the time, the developers did not know of a variable-length, linked-list equivalent in Visual Basic.

A.3.1.2 Tracking: The NMEA GPS Component

Another tracking object was developed for any NMEA-compliant GPS unit, with special allowances made for the inexpensive Delorme Tripmate GPS. This object maintained the same interface as the Trimble GPS object to allow for drop-and-swap replacement within the existing vehicle application.

A.3.1.3 Trailer/Cargo Monitoring Components

Toward the end of this year, trailer and container objects were developed. The trailer object was based on the existing trailer interface in the VIC software and, at the protocol level, is very similar to the QualComm interface. The container objects which represented the cargo were based on work being done by Bill Pregent and Dave Skogmo with the Cargo Monitoring system. For each of these objects, objects were also constructed to represent various events in this domain, such as connection of a trailer or detection of a container status change.

A.3.1.4 Other Components

A standard key-word message format was agreed to by the developers on the vehicle and command center domains of CTSS, and a "message processor" object was developed in the vehicle framework to allow easy construction/deconstruction of messages sent to/from the vehicle.

Following development of the message format, the Bokler DESCypher component was encapsulated into a DES cryptographic component and integrated with the standard key-word message format to provide crypto-

graphic capabilities for both QualComm and TeleComm communications.

A.3.1.5 Integration: The Vehicle Computer

The simple vehicle computer application which was developed at the end of FY97 was modified to allow the operator to configure for different GPS units and to indicate that a trailer and cargo were to be monitored. A similar vehicle application was also developed to test the TeleComm object. To date, no vehicle application has been developed which allows for switching between communication objects, although this should be feasible.

A.3.2 Evaluation

Integration of the TeleComm component into a new vehicle computer went smoothly, but options had to be added to support telephone number entry and call originator settings at the application layer. Integration of the NMEA GPS component were equally smooth. Two vehicle applications were built: one with the QualComm component and one with the TeleComm component. However, both applications contained selectable interfaces for either Trimble or NMEA GPS components at run-time.

Several problems were encountered during the development and testing of the TeleComm component: 1) the unreliability of general telephonic communications, 2) the unreliability of cellular data links, and 3) the incompatibility of so-called "Hayes-compatible" modems.

Telephonic communications were unreliable primarily due to the variability of telephone line conditions from call to call and site to site. This was mitigated through the development of an extremely robust handshaking protocol and retry procedure between the command center and the vehicle.

Data connections through cellular modems/telephones proved highly unreliable during testing. Two different modems were tested: the Motorola "Montana" modem and the USRobotics Megahertz 33.6 PCMCIA modem. These two modems were designed to be used over both land-line-based and cellular-based data links. Both modems auto-configured themselves depending on whether a regular telephone or a cellular telephone was used. During testing with cellular telephones, both were found to have serious problems losing this auto-configuration information and attempting to use the cellular data link as a regular land-line based link, which caused the link to fail completely. After much research, the devel-

opment team also discovered that cellular data links were only considered to be reliable in stationary use (i.e. the vehicle cannot be moving while the data link is established and operating). (See article on The cellular telephone network automatically adjusts the bandwidth of any link based on the traffic in a particular cell, which causes unreliable data throughput. Also, movement of the vehicle causes frequent "hand-offs" from one cellular tower to another, causing severe noise bursts which subsequently cause the data link to fail. It was finally determined that use of cellular telephones for data communications in a mobile environment was not viable at this time.

Data connections through an AMSC satellite telephone proved more reliable, but the "Hayes-compatible" modem within the telephones were not very "Hayes-compatible". Two AMSC satellite telephones were tested: a Westinghouse AMSC telephone and a Mitsubishi AMSC telephone. The Westinghouse telephone, with its omnidirectional antenna, was the only one usable in a mobile vehicle. The Mitsubishi telephone comes with a directional "aim-and-shoot" antenna which fails in a mobile environment. Certain commands which should work on any Hayes-compatible modem caused errors within the Westinghouse telephone, so the TeleComm component was limited to sending only a subset of the Hayes command protocol. Also, the modem within the Westinghouse telephone did not implement a certain RS232C line correctly. The "CDHolding" line, which provides a constant indication as to the state of the data link, appeared to not be implemented at all.

This component was immediately used in another project (SATCAM/FIRST), which proved out the framework architecture using AMSC satellite telephones to transfer digital pictures from remote vehicles to a command center. The SATCAM/FIRST project used only the file-transfer features of the TeleComm object; the CTSS vehicle computer application used only the messaging features.

An interesting problem with data formats, the Visual Basic compiler, and the Bokler DESCypher middleware component occurred during testing. Visual Basic stores strings in Unicode format (two bytes per string character), but only sends the lower-order byte over the serial port using its Comm Control object. The Bokler component encrypts and decrypts using both the higher-order and lower-order bytes of the Unicode strings. Modifications were made to the encapsulation of the Bokler component to compensate for this "feature". The result was that 16-byte blocks were required for encryption/decryption rather than the 8-byte blocks that DES

normally requires. Because of this, the command center was also required to send encrypted messages in multiples of 16-bytes.

A.4 Vehicle Framework, FY99

A.4.1 Development

The third year of framework development for the vehicle focussed on more alternate communications between the vehicle and the command center, more alternate navigational tracking, modifications to the interfaces between the vehicle computer and the trailer and cargo domains, incident management, system diagnostics, system configuration, and system power control.

A.4.1.1 Communications: The Orbcomm Component

At the time of this report, a framework component is being developed to provide communications to the command center and other vehicles via the Orbcomm satellite communication system. The operation of this component is very similar to the QualComm component, except that this component must retry messages if the Orbcomm transceiver indicates that a message is lost.

The Orbcomm transceiver that is being used in this development also contains an internal GPS unit. The interface to this unit has been implemented through the Orbcomm transceiver interface.

A.4.1.2 Tracking: The Orbcomm Internal GPS

As mentioned above, the Orbcomm transceiver being used in the development of the Orbcomm communication component contains an internal GPS receiver. The Orbcomm component therefore acts like two different standard components: a communications component and a tracking component.

A.4.1.3 Tracking: The GPS Simulator

A component which reads GPS data from a file and simulates the tracking component interface to the application layer was developed for demonstrations and for laboratory testing without requiring the use of actual vehicles. The component does not utilize any hardware interface on the platform.

A.4.1.4 Incident Management

To perform efficient incident management in the vehicle, priority-based queuing was added to all the communication components at this time. The application may specify any of priority levels from 0 to 255 for any outgoing message to the command center. The higher the number, the higher the priority. Setting all messages to the same priority causes the queues to act as FIFOs. The "URGENCY" keyword in the keyword-based messaging format was finally used int messages between the vehicle and command center domains to indicate the priority level of certain messages.

To allow the generation of events within the vehicle, three components were developed: a wrapper which encapsulates a standard PC parallel port, a digital-input class, and a digital-output class. A hardware interface box was developed to allow easy connections of switches and LEDs to the port, and to allow buffering of port inputs. Switches, such as an emergency switch, could be connected to the PC parallel port via this hardware interface box. The digital-input class could then be used by the application to determine if a switch-flip or button-press had been made and generate an appropriate incident report to the command center. Also, LEDs could be connected to the PC parallel port via this hardware interface box to indicate system status.

A.4.1.5 System Diagnostics

A component for time-stamped component-stamped logging was developed. This log was ASCII-based, using a keyword-format similar to the one used in messaging. Also, each log event was timestamped to allow easy compilation/sorting of different logs from different vehicles and domains. Most of the communication, tracking, and trailer/cargo components were modified to use this logging component if so specified by the application.

Another component, one for monitoring the raw data streaming across any data interface, was developed. This component was linked into the communications, tracking, and trailer/cargo components and made accessible via the status forms for each component.

A.4.1.6 System Configuration

Due to the need to easily reconfigure applications to different ports, users, etc., a component to read keyword/value pairs from an initialization file was developed to allow data-driven configuration of the applica-

tion. This file was put in an ASCII format. The application decides how to use these keyword/value pairs, and whether to default to some "safe" settings if these values are erroneous or nonexistent.

A.4.1.7 Other Components

Changes were made to the trailer, container, and container event objects to support a new type of container tag being used in the Cargo Monitoring system. This change did not effect the component interfaces to the application.

A power control/system shutdown component was added to the framework to allow the application to shut itself and the computer down when necessary. This component could activated by the application as needed.

A.4.1.8 Integration: The Vehicle Computer

The vehicle computer applications which were developed at the ends of FY97 and FY98 were modified to allow the operator to configure for an emergency switch and for automatic shutdown of the vehicle computer based on an external input.

A.4.2 Evaluation

Evaluation of the Orbcomm communications and tracking component is ongoing at the time of this report.

Testing of the incident management features added to the vehicle framework showed that Windows NT would not allow manipulation of the parallel port at the level required by the parallel port object. At this point, the vehicle framework was limited to use under Windows 95 and 98. The priority-based queues for messaging worked as expected, allowing emergency messages to jump ahead of lower priority messages outbound to the command center.

The diagnostic components for logging and interface monitoring were very successful, and the vehicle development team agreed that they should have been developed sooner in the project.

The system configuration component was used by the STORC and the SATCAM/FIRST projects to allow easy reconfiguration of those applications. It was very easy to integrate and to utilize.

APPENDIX B Command Center Framework: Lessons Learned

B.1 Development of the Encryption Classes for the Command Center

In implementing encryption for CTSS, two basic lessons stood out. The first relates to the software design process, and is an instance of a well known fact: sometimes new requirements conflict with prior design decisions - and cannot in fact be implemented without going back and modifying the design. In this case, the CTSS communication design was not sufficiently versatile to allow the encryption layer to include a key-exchange component. The second lesson is well known to any security practitioner: encryption is easy, security is hard. Beyond encryption algorithms and protocols, issues of software quality, key management, and platform weaknesses present difficult (read "expensive") problems to solve. It may be of some use, however, to set out a few details of how that fact presented itself during the CTSS project.

B.1.1 CTSS Encryption and the Software Design Process

The idea of encrypting CTSS communications was added late in the design process. At least, this is true formally: although it was known for the entire life of the project that encrypting messages would be a useful capability, the actual design process did not take encryption into account until the end. The main result of this is that encryption did not fit naturally into the software structure, and some capabilities could not be integrated at all.

There are actually two aspects to consider. The first aspect is the communications protocol. That is, CTSS defines the message format (both syntax and semantics) for messages between the control center and the vehicles. Any changes to this interface affect both sides, and are therefore expensive. CTSS followed a model that will be common in any project dealing with two different environments (in this case, the control center and the mobile components) - that different development personnel work in each environment. Therefore the communications protocol is in fact an interface at the highest level of the project: between the two halves of the development team.

It is of course well known that great care must be taken in defining interfaces at this level. Although the message format was chosen with the concurrence of all involved, it is clear in hindsight that it was not enough: the encryption requirement was not considered. The particulars of how the message format specification made certain capabilities difficult are set out later. At this point it suffices to note a couple of lessons. Generally, one must recognize which interfaces are important in order to give them sufficient attention. An interface that affects developers from two different teams must be treated as though it affects the entire teams. And communication protocols are almost inevitably interfaces at the highest level.

In addition to the communication protocol definition, the actual implementation of communications was not constructed so as to allow new layers to be added. To begin with, the class TeleComm implemented message exchange, as supported by HayesModem and SerialPort. Then actual encryption (and key exchange) was implemented with Cipher and related classes. Finally, ProtTeleComm inherits from TeleComm and adds encryption capability using Cipher. This is a moderately successful use of object-oriented techniques: it was not necessary to implement the original class again, but it was necessary to modify it. The modification made certain behavior overridable (so that ProtTeleComm could override it!). The result was also limited because TeleComm did not have any concept of different levels of messages, which is necessary to properly implement key exchange. ProtTeleComm supports some key exchange in spite of this, but is not robust: it fails if one side to a communication has to restart (because the other side won't recognize the restart). To reiterate, supporting robust key exchange requires both sides to be able to recognize which messages are key exchange messages and which messages are (encrypted) application messages.

In fact it is the message format and meaning, i.e., the protocol, that is the root cause of TeleComm's inflexibility. The comments in the file header lay out the protocol, including some limitations. In addition to the fact that the protocol only defines one kind of (application) message, there is another limitation. To quote: "This protocol implements a non-dialog approach to vehicle/command center communication. There is no requirement or envisioned usage that would necessitate a dialog. As implemented, this protocol can be used to send one request and receive one reply. Or, it can be used to send lots of messages, but not receive a reply after each one." This is a fatal limitation for key exchange.

Another way to interpret these results is a failure to abide by the rule "don't reinvent the wheel". There is in fact a great deal of literature on communication protocols. For example, the ISO defined a Reference Model for networking comprising seven layers before 1980. Also, the Internet provides a rich source of protocol definitions, starting with TCP/IP; the SSL (secure sockets layer) could have been applicable. The Unix "streamio" package is another example; it defines a way to add "modules" to a bi-directional data stream. Jon Bentley describes the usefulness of "self-describing data" in recognizing what to do with a set of information. What happened in CTSS is that the communication step itself was regarded as a simple problem, and there was no need to do anything more than construct a simple solution - a new wheel. In the end, the simple solution was too simple.

From the point of view of software design, there are two ways in which the CTSS project could have done better in regards to encryption. The first way would be to consider encryption early: even relatively simple design changes, done early enough, would leave adequate "hooks" to add encryption in later. One could include robust key exchange, compression, and other facilities. The other way would be to recognize that the communication protocol is an interface at a very high level, and needs to be designed with a lot of flexibility to begin with, using ideas from other systems.

B.1.2 CTSS Encryption and Real Security

The algorithms used for encrypting messages for CTSS are more than adequate in terms of cryptographic strength. But other security concerns are addressed in only the simplest ways.

Consider, first of all, the need to generate good keys. The CTSS implementation simply allows the users to specify passwords (or passphrases),

with no particular check on their quality. We do not even provide guidance on what constitutes a good password. A secure system requires good quality keys, and CTSS does not prevent that, but neither does it provide any help. If the user chooses passwords of sufficient complexity and length, then the enemy will not be able to guess them and the encrypted messages will remain safe. A better system, however, does not depend on user skill for security.

Another requirement on a secure system is to store keys securely - that is, to hide them. CTSS simply stores keys (at the control center) in an ordinary file, with no extra protection added. Either the control center computers must be physically protected from the enemy (including insiders!), or we presume that the enemy doesn't realize how the key file is used. It is possible to operate in this way, but it is easy to see the potential for failure here. A simple improvement would be to encrypt the key file with a master key - but make sure that the master key itself is good - and protected! And there is more to come, below.

Another aspect of secure key storage is to ensure that the cryptographic module (software, in the case of CTSS) protects the keys from accidental exposure. One usual requirement is an extremely careful inspection of the code. One would hate for a bug to result in sending the key out in a message, instead of the user data. While every reasonable care was taken in implementing the CTSS encryption, it could not be fairly said, for example, that the development procedures would pass the stringent NSA requirements. Another problem for CTSS is the platform on which the control center operates - an ordinary workstation computer. The operating system (e.g., Unix or Windows NT) can be counted on to swap application data to disk, including encryption key values. While it seems unlikely that a key can be found by examining swap area (and of course the existence of the key file mentioned above obviates this technique anyway), a truly secure system needs to protect against even this kind of accident. There are techniques for avoiding swap problems, but CTSS did not go to the trouble (expense) of doing this.

Finally, I will mention the key distribution problem. Both sides of an encrypted channel must know the key. CTSS provides no facilities at all for distributing keys (passwords). The user is responsible for communicating passwords through some other channel - which should, of course, be secure. Public key methods for key exchange are perfect for this, but (as described earlier) CTSS does not have that capability.

The above difficulties will not surprise anyone who has attempted to design a very secure system. For those who supervise such work, or pay for it, this information might provide a better feel for the difficulty inherent in the security problem.

One more note. In recent years many standards for encryption have appeared. It would be beneficial for any future system, whether based on CTSS or not, to be cognizant of these standards. Like software in general, it pays to avoid reinventing the wheel in encryption. RSA Labs maintains a set of numbered standards known as PKCS (Public Key Cryptography Standards) #1, #2, and so forth. At present they can be found on the Internet at http://www.rsa.com/rsalabs/pubs/PKCS/. There is also an effort by IEEE to create a set of standards, known as P1363. That group's web site is http://grouper.ieee.org/groups/1363/. The U.S. government standards agency, NIST, is at http://www.nist.gov/, and has a lot of useful information, including Federal Information Processing Standards at http://www.nist.gov/fips/.

B.1.3 Summary

In the end, the CTSS encryption capability works, and is capable of keeping communication private, but is not particularly sophisticated. Future systems may benefit from the lessons learned in implementing CTSS. In software design, it pays to be careful with high level interfaces, particularly communication protocols, and this includes considering all requirements to begin with. For security, the least of the problems is the encryption algorithm; key management is much harder and may influence many decisions, including the choice of platform and overall system architecture.

APPENDIX C Example Implementations

The CTSS software has been deployed with several applications. Descriptions follow of implementations in the vehicle and command center domains. These implementations can be intermixed to support specific customer requirements. Figure 8 shows how one system can support various types of vehicles and command centers

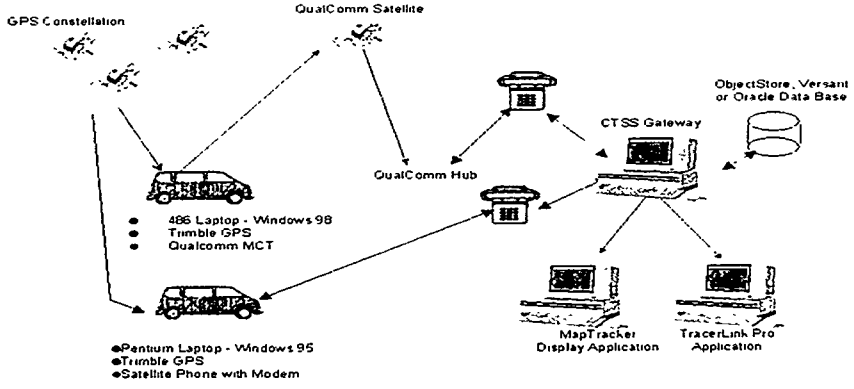

Figure 8 Application using multiple CTSS transportation objects in the vehicle fleet and command center.

C.1 Vehicle Domain

The CTSS software for the vehicle domain has been developed for Windows environments. Several commercial vendors are beginning to produce ruggedized PCs designed for mobile environments. While these products still need to mature, it appears that the PC hardware platform may be adequate for some mobile applications. Problems areas are:

- The standard, overt, windows shutdown mechanism is incompatible and non-intuitive with an automobile shutdown.
- Mobile applications require multiple inputs. In particular, multiple serial ports are needed for monitoring and data acquisition. The new USB (universal serial bus) interface appearing on some computers may address this problem.

Notwithstanding these problems, vehicles applications have been developed using the CTSS framework. Figure 9 diagrams the CTSS components used in building two vehicles. One vehicle supports a QUALCOMM communications unit. and the other supports a satellite telephone.

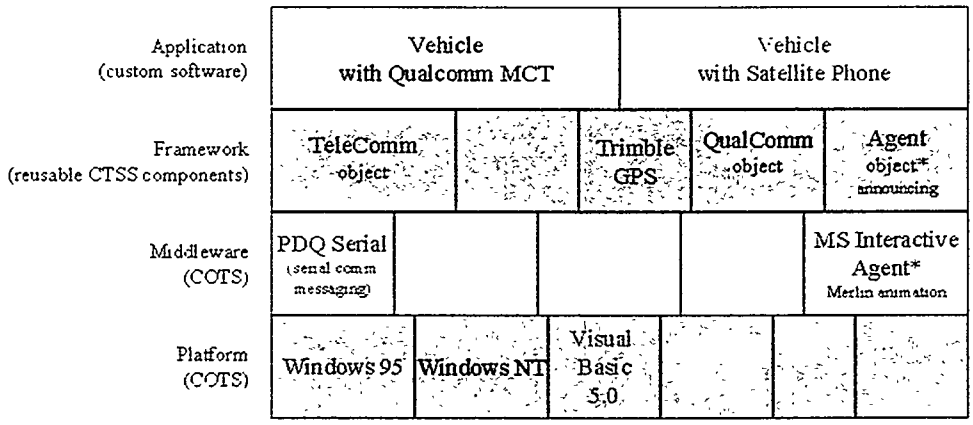

Figure 9 Example of two vehicles using CTSS.

A rudimentary application interface has been developed and is shown in Figure 10. The interface has been designed to support either keyboard, mouse, or touch screen input.

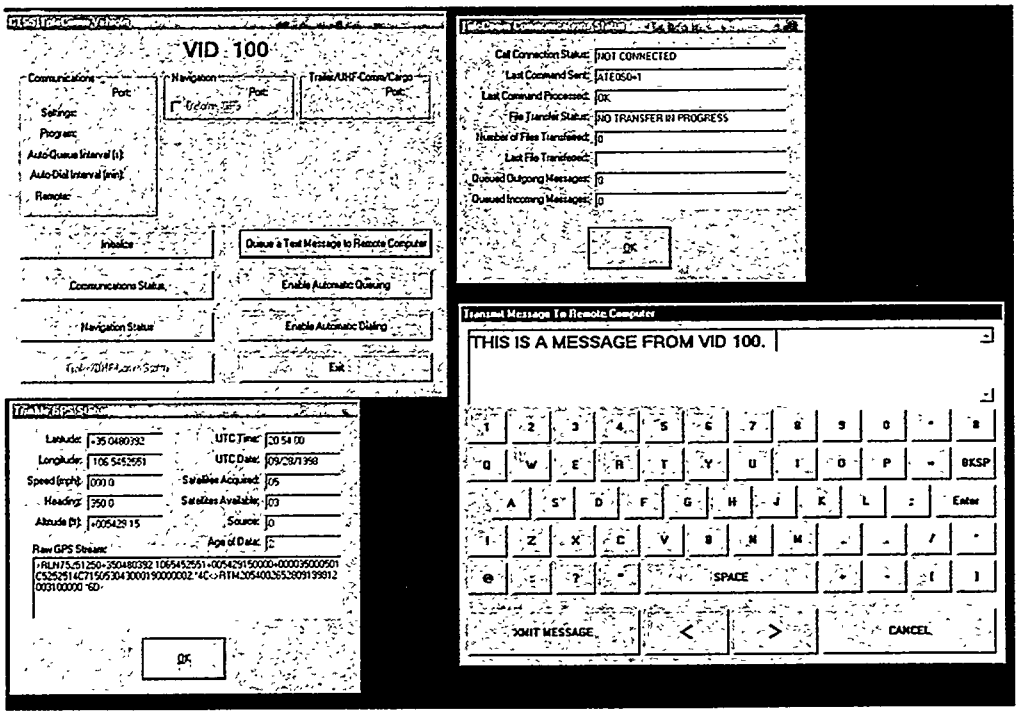

Figure 10 Application interface for CTSS-based vehicle.

C.2 Command Center Domain

The CTSS software for the command center was originally developed on a Silicon Graphics Computer, running IRIX, which is a UNIX derivative. During fiscal year 1998, the software was ported to two other UNIX-based operating systems: Digital UNIX and Data General DG/UX. The port of the command center software to Windows NT was not completed in fiscal year 1998, as planned, but was completed in fiscal year 1999.

The CTSS components have been used in two command center applications. Figure 11 diagrams the CTSS components used in building these two applications.

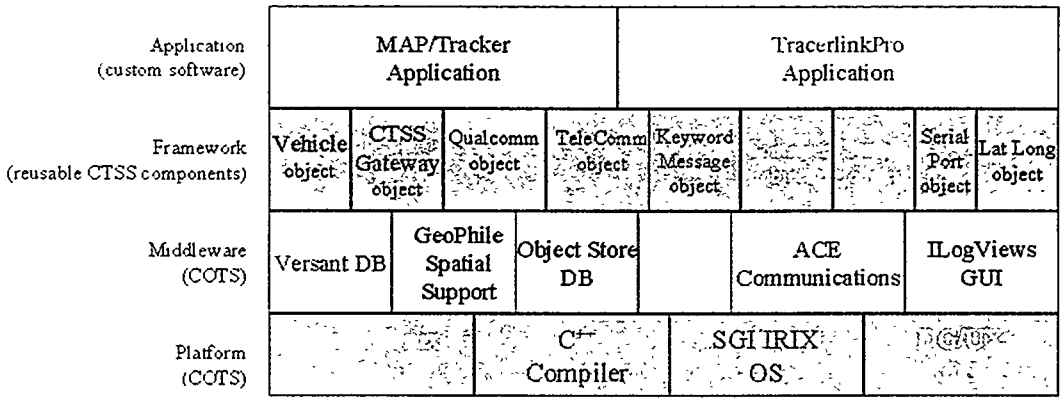

Figure 11 Example of two command centers using CTSS.

The MAP/Tracker application has a relatively rich user interface. It support a large number of the CTSS objects. Figure 12 shows the start-up screen for MAP/Tracker using CTSS components.

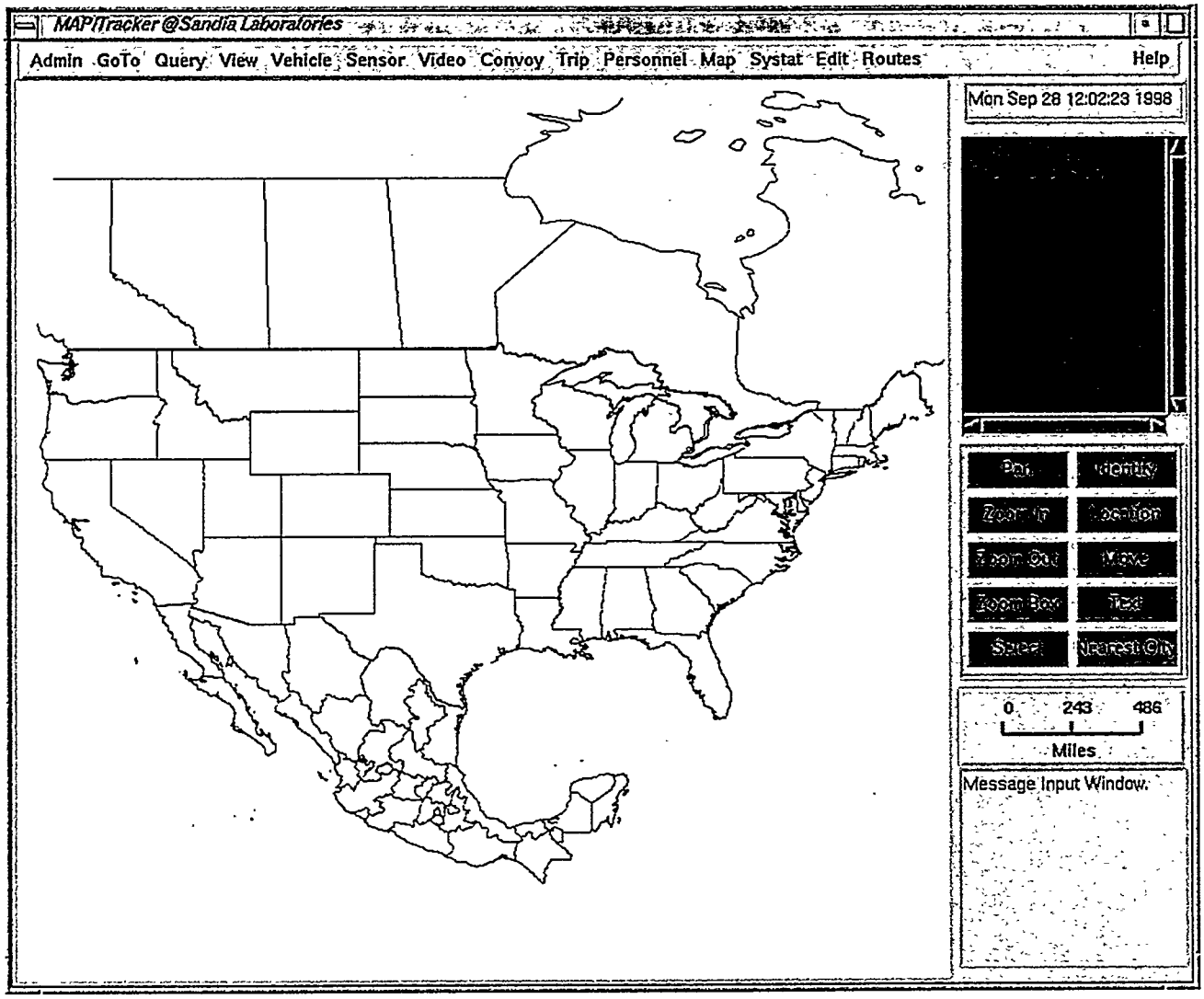

Figure 12 MAP/Tracker application interface for CTSS command center.

The CTSS component library was also used in the development of the Sandia Proof of Concept (SPOC) during fiscal year 1998. SPOC uses a database-centric architecture. The CTSS layered approach adapted very well to this architecture. By using one database as the focal point, the issue of platform-specific software became a moot point. The CTSS components used in SPOC are diagramed in Figure 12.

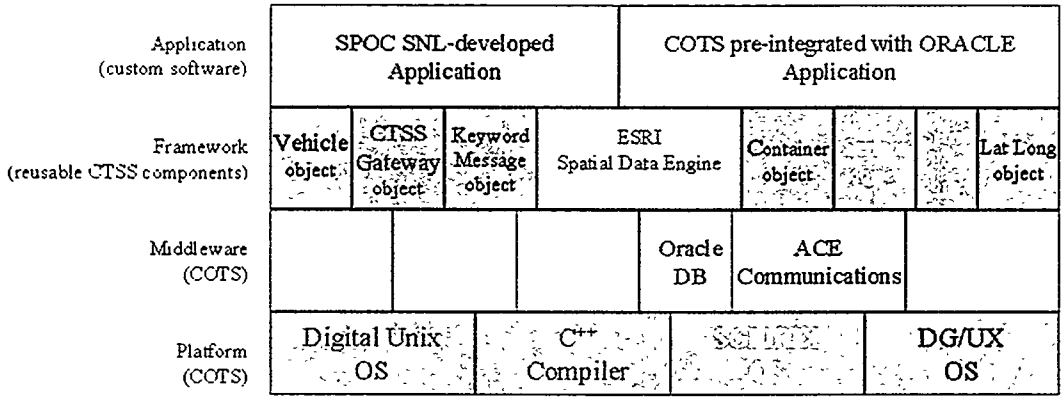

Figure 13 CTSS usage in SECOM Proof of Concept

C.3 Satellite Camera

The CTSS vehicle components were used to develop the Satellite Camera product. The unique feature was that the vehicle components were also used to develop the command center portion of the application. Figure 12 diagrams CTSS usage in the satellite camera application.

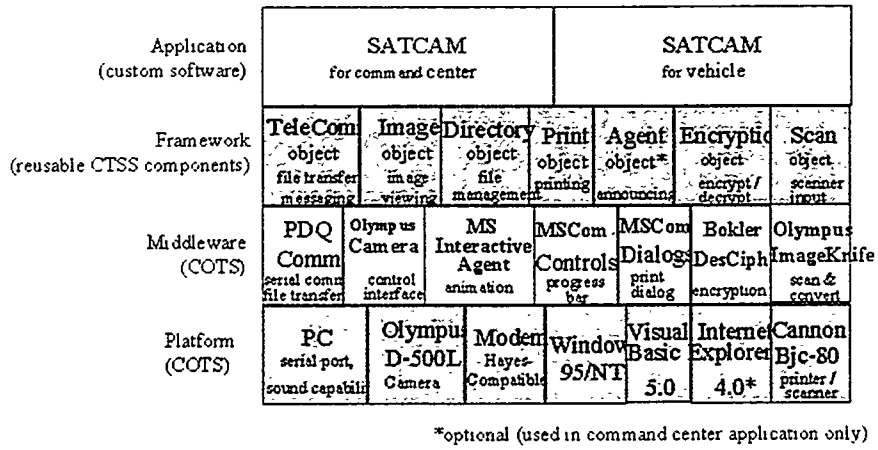

Figure 14 CTSS usage in Satellite Camera Application.

C.4 STORC

The CTSS software and architecture was the basis for developing a tracking demonstration system for the Air Force Space BattleLab. The Satellite Tracking OF Re-entry vehicle Convoys is referred to by the initial ism STORC. This system used both the command center and vehicle implementations.

This particular application gave specific evidence to the successfully having achieved the goal of "rapid development". The application was designed, developed, and tested in 3.5 staff months. This is a notable achievement for a very customized application.

Here are the block diagrams showing which components of the CTSS library were used to develop STORC.

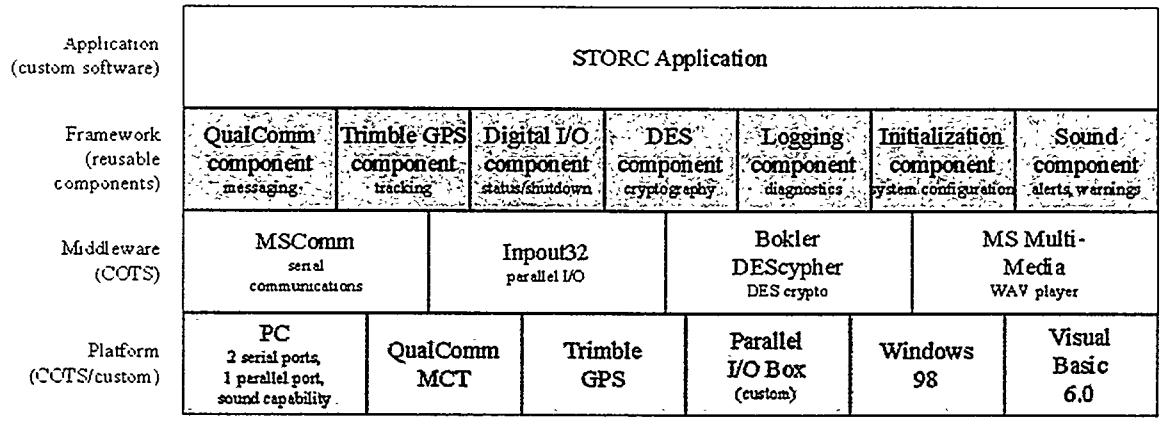

Figure 15 CTSS usage in STORC Vehicle Application.

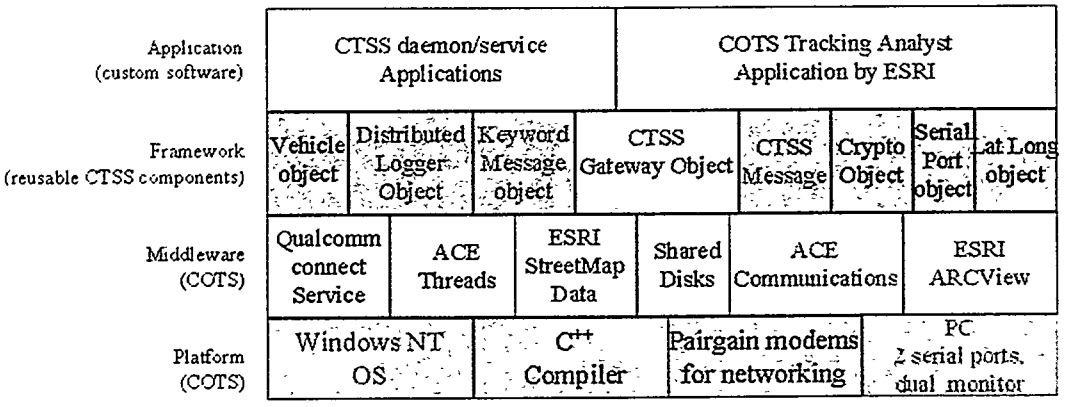

Figure 16 CTSS usage in STORC Command Center Application.

Figure 17

DISTRIBUTION:

1	MS 0775	R. C. Beckman, 5852
1	MS 0775	G. M. Corbett, 5852
1	MS 0775	K. L. Green, 5852
5	MS 0775	S. M. Kelly, 5853
5	MS 0775	J. W. Myre, 5853
1	MS 0775	S. W. Ratheal, 5853
5	MS 0775	D. W. Scott, 5853
1	MS 0775	C. A. Ulibarri, 5853
5	MS 1110	E. D. Russell, 9223
1	MS 9018	Central Technical Files, 8940-2
1	MS 0899	Technical Library, 9616
1	MS 0612	Review and Approval Desk, 9612 For DOE/OSTI
1	MS 0161	Patent and Licensing Office 11500