THESIS

AN ARCHITECTURAL FRAMEWORK FOR INTEGRATING COTS/GOTS/LEGACY SYSTEMS

by

Karen M. Gee

June 2000

Thesis Advisor: Luqi
Second Reader: Mantak Shing

Approved for public release; distribution is unlimited.
An Architectural Framework for Integrating COTS/GOTS/Legacy Systems

Gee, Karen M.

Naval Postgraduate School
Monterey, CA 93943-5000

US Army Research Office and Institute of Joint Warfare Analysis

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

Approved for public release, distribution is unlimited.

Building distributed systems more effectively and efficiently is an essential goal of Department of Defense (DoD). We are driven by the push toward greater use of COTS, the need to improve access to legacy data and services, and the new business opportunities offered by web-based technologies and electronic commerce. To fully realize the DOD's goal, a new architectural framework is needed.

This thesis proposes an architectural framework suitable for integrating COTS/GOTS/legacy systems in a distributed, heterogeneous environment. The proposed architectural framework uses The Open Group Architectural Framework (TOGAF) as a basis and includes new tools to support the COTS/GOTS/legacy system development and integration. A case study for the Naval Integrated Tactical Environmental Systems (NITES) program where a prototype is built, demonstrates the effective use of the proposed architectural framework.

COTS, GOTS, Legacy Systems, Architectural Framework, Distributed Systems, Software Development

Unclassified

Unclassified

Unclassified

Unclassified

UL

NSN 7540-01-280-5500
AN ARCHITECTURAL FRAMEWORK FOR INTEGRATING COTS/GOTS/LEGACY SYSTEMS

Karen M. Gee
B.S.E.E., University of California, Davis, 1987

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2000

Author: ____________________________
Karen M. Gee

Approved by: ____________________________
Luqi, Thesis Advisor

Mantak Shing, Second Reader

Luqi, Chairman
Software Engineering Curriculum

Dan Boger, Chairman
Department of Computer Science
ABSTRACT

Building distributed systems more effectively and efficiently is an essential goal of the Department of Defense (DoD). We are driven by the push toward greater use of COTS, the need to improve access to legacy data and services, and the new business opportunities offered by web-based technologies and electronic commerce. To fully realize the DoD’s goal, a new architectural framework is needed.

This thesis proposes an architectural framework suitable for integrating COTS/GOTS/legacy systems in a distributed, heterogeneous environment. The proposed architectural framework uses The Open Group Architectural Framework (TOGAF) as a basis and includes new tools to support the COTS/GOTS/legacy system development and integration. A case study for the Naval Integrated Tactical Environmental Systems (NITES) program where a prototype is built, demonstrates the effective use of the proposed architectural framework.
# TABLE OF CONTENTS

## I. INTRODUCTION
- A. PURPOSE ................................................................. 1
- B. MOTIVATION ............................................................. 3
- C. ORGANIZATION .......................................................... 4
  1. Essentials of A Structured Software Development Process .............. 4
  2. Architectural Framework ............................................. 4
  3. Proposed Architectural Framework ..................................... 5
  4. NITES Case Study ...................................................... 5
  5. Recommendations and Conclusions ..................................... 5

## II. ESSENTIALS OF A STRUCTURED SOFTWARE DEVELOPMENT PROCESS
- A. TRADITIONAL SOFTWARE DEVELOPMENT ................................ 7
  1. Requirements .......................................................... 9
  2. Design ................................................................. 14
  3. Implementation ........................................................ 17
  4. Test ................................................................. 18
- B. SOFTWARE DEVELOPMENT USING COTS COMPONENTS .................... 22
  1. Requirements with COTS ............................................. 26
  2. Design with COTS .................................................... 27
  3. Implementation with COTS .......................................... 29
  4. Testing with COTS ................................................... 31

## III. ARCHITECTURAL FRAMEWORK
- A. INTRODUCTION .......................................................... 35
  1. Architecture vs. Architectural Framework ............................ 37
  2. Importance Of An Architecture & An Architectural Framework ......... 39
- B. TOGAF ................................................................. 41
  1. Architectural Development Model ..................................... 42
  2. Foundation Architecture ............................................. 43
  3. Resources .......................................................... 48

## IV. PROPOSED ARCHITECTURAL FRAMEWORK
- A. TOGAF UPDATES ........................................................ 52
- B. TOOLS FOR THE PROPOSED ARCHITECTURAL FRAMEWORK ............. 60
  1. Object Request Brokers .............................................. 61
  2. Wrappers ............................................................ 82
  3. Glue Code .......................................................... 83
  4. XML ............................................................... 86

## V. NITES CASE STUDY
- A. NITES ................................................................. 89
  1. NITES Background ................................................... 89
  2. Existing NITES Architecture ......................................... 90
  3. Data vs. Product .................................................... 92
- B. CASE STUDY ........................................................... 92
LIST OF FIGURES

Figure 1 - Technical Reference Model From Ref. [16]...... 45
Figure 2 - NITES Architecture................................. 91
Figure 3 - Functional View of Existing NITES Environment. 93
Figure 4 - Existing Hardware Topology...................... 95
Figure 5 - NITES Distributed Computing Architecture..... 106
Figure 6 - NITES Security View................................. 109
Figure 7 - NITES Functional Architecture.................... 112
LIST OF TABLES

Table 1 - Software Methods [Ref. 5].......................... 9
Table 2 - Proposed Architecture Development Cycle.......... 59
Table 3 - COTS/GOTS Phase 2 Evaluation Results............... 99
Table 4 - COTS/GOTS Phase 3 Evaluation Results............ 105
Table 5 - Mapping of Services to Existing Architecture... 106
I. INTRODUCTION

A. PURPOSE

The trend towards using Commercial Off-The-Shelf (COTS) components within the Department of Defense (DoD) is the preferred way to build systems [Ref. 1]. Prior to COTS integration into system architectures, all DoD software-intensive systems were built around organically developed source code. With constantly dwindling budgets and tighter schedules, the focus has shifted to building software-intensive systems by integrating COTS software components.

In 1996, DoD created a paradigm shift in software development when it strongly encouraged the use of commercial-off-the-shelf (COTS) components for development of new Automated Information Systems (AIS) if an existing system cannot be modified [Ref. 2].

Initially, the DoD community embraced this mandate believing COTS integration

• is economical, both in initial development and lower life cycle costs;

• is faster to deploy;

• is already developed with proven capability;

• reduces the technical risks;
• is fully supported with documentation and technical support;
• represents state of the art technology.

However, as more experience is gained using COTS for building DoD systems, many have found there are many problems to be addressed with COTS: Tracz [Ref. 3] sites plug and play software does not always work. Software developers had to write code to enable the COTS software packages to intercommunicate.

Building distributed systems more effectively and efficiently is an essential goal of the DoD. We are driven by the push toward greater use of COTS, and the need to improve access to legacy data and services.

This thesis proposes an architectural framework that provides an effective approach to integrate COTS software components into distributed, heterogeneous systems. This thesis focuses on military AIS where multiple, mutually exclusive, standalone COTS products are required to communicate together in a single integrated system along with legacy and GOTS components. A case study with the Naval Integrated Tactical Environmental System (NITES) program demonstrates the effectiveness of the proposed architectural framework.
The primary purpose of this thesis is to build an architectural framework suitable for integrating COTS/GOTS/legacy systems in a distributed, heterogeneous environment. Additionally, the secondary purpose is to discuss and analyze the traditional software development process and how COTS affects it.

B. MOTIVATION

Building systems with COTS/GOTS/legacy components is not as easy as just loading these components onto a platform and assuming all the components will communicate with each other. The conglomerate of software applications on a machine does not comprise a system. A truly integrated system is greater than the sum of its parts; it provides new services and increased value.

Not only must components within a system communicate with each other but when distributed, heterogeneous environments abound, systems must interoperate with each other. Interoperability is the ability of systems to provide services to and accept services from other components within a system, and to use the services so exchanged to enable them to operate effectively together as a cohesive unit.
Architectural frameworks on the market today do not sufficiently address the building of distributed, heterogeneous systems with COTS/GOTS/Legacy components. To effectively build distributed, heterogeneous systems with these components, an architectural framework, which addresses the interoperability issues and provides the tools necessary to build an integrated, interoperable target architecture, is needed.

C. ORGANIZATION

1. Essentials of A Structured Software Development Process

This chapter discusses the essentials of a structured software development process for both a traditional software development and a development using COTS components. It describes how the traditional software development process must be modified to accommodate the use of COTS components.

2. Architectural Framework

This chapter defines the difference between architecture and architectural framework, discusses the importance of an architectural framework in a software development process and introduces The Open Group Architectural Framework (TOGAF).
3. Proposed Architectural Framework

This chapter describes the proposed architectural framework for integrating COTS/GOTS/legacy systems in a distributed, heterogeneous environment. It uses TOGAF as a basis, modifying it to address the integration of COTS/GOTS/legacy systems. Additional tools are also proposed for this new architectural framework to address the interoperability of distributed, heterogeneous systems.

4. NITES Case Study

This chapter presents a case study using the NITES system to demonstrate the effectiveness of the proposed architectural framework to build an integrated NITES architecture and the effectiveness of these tools in building an interoperable, distributed, heterogeneous system.

5. Recommendations and Conclusions

This chapter recommends areas of additional research and concludes with a summary analysis of the proposed architectural framework.
II. ESSENTIALS OF A STRUCTURED SOFTWARE DEVELOPMENT PROCESS

A. TRADITIONAL SOFTWARE DEVELOPMENT

Over the past decade, the field of software engineering has seen many changes to the methods of software development [Ref. 4]. A software methodology is the set of rules and practices used to create computer programs. The purpose of a software method is to bring structure into a software development process. Software engineering has transitioned from the waterfall method to the incremental build method and the spiral method.

The waterfall method requires each phase of the development process to be completed prior to proceeding to the next phase. In conjunction with certain phase completions, a baseline is established that "freezes" the products of the development at that point. If a need is identified to change these products, a formal change process is followed to make the change. The graphic representation of these phases in software development resembles the downward flow of a waterfall.

In using the waterfall method, many limitations are noted. Some of these limitations include a long duration prior to seeing a fielded system, a full budget to see the development to completion, all requirements must be known
upfront which is not always practical. Anyone of these limitations can cause the development to fail. These limitations perpetuated a need for new methods which are less restrictive.

With the incremental build method, producing usable functionality earlier is key. The full requirements of the system are known upfront but not implemented all at once. Due to budgetary and/or schedule constraints, the requirements are prioritized and implemented in a build based on priority. The initial build contains some basic functionality and is fielded. Each successive build will contain more of the requirements. The incremental build method is complete when all the requirements are implemented.

The spiral method allows the development to proceed without knowing the full requirements of the system upfront. Through each iteration, the requirements evolve as more information is gathered and learned about the system. The spiral method is complete when no additional requirement is formulated in the last iteration of the spiral.

Table 1 [Ref. 5] summarizes the differences in each of the three software methods.
<table>
<thead>
<tr>
<th>Program Strategy</th>
<th>Define All Requirements First?</th>
<th>Multiple Development Cycles?</th>
<th>Field Interim Software?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Incremental (Preplanned Product Improvement)</td>
<td>Yes</td>
<td>Yes</td>
<td>Maybe</td>
</tr>
<tr>
<td>Spiral</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1 - Software Methods [Ref. 5]

In any of these methods, the following phases of developing a software system remain the same:

- Requirements,
- Design,
- Implementation, and
- Test.

The maintenance phase of the software development cycle is not addressed in this thesis as the focus is purely on the development of software systems. [Ref. 6] provides a detailed discussion on the maintenance of software systems.

1. **Requirements**

The requirements phase is the first phase in the software development process. The requirements phase allows stakeholders to form a consensus as to system requirements. System requirements provide the foundations for all system
development and test. Established requirements reduce the risks of building a system that does not meet the needs of the end user.

The requirements phase forms a basis for defining the software system to be built. It consists of requirements identification/definition, analysis and management. It is concerned with the 'what' of the problem.

a) Requirements Identification

Requirements seldom come ready made from the information process. Normally, when the initial requirements are provided, it is very generic and abstract. It may be nothing more then just an idea. However, a system cannot be built based on just an idea. The requirements phase tries to turn the idea(s) into a concrete set of requirements that a system can be built from.

The requirements include the functions, attributes and constraints of the system to be built. Functions are tasks to be performed by the system. They are the verbs describing the actions the system will perform. To identify these functions involves eliciting them from the customer, users, and other stakeholders. This can be accomplished by interviewing the customer, users and stakeholders, brainstorming, prototyping, questionnaires, etc.
During requirements elicitation, the system requirements are derived from domain experts, people familiar with their domain but not necessarily with building software systems. The system developers must therefore be conversant in the terms and limitations of the domain, since the domain experts are probably not conversant in the terms and limitations of software engineering.

The attributes are characteristics the customer desires. They are the adjectives and adverbs. Many refer to attributes as nonfunctional requirements because the customer wants them but they are not things the system will do.

The constraints are objective statements of the attributes. They place quantitative limits or constraints on the customer's desires. The most common constraints on system requirements are time and money.

When the functions, attributes and constraints are gathered, the initial requirements evolve into a more specific, more detailed, and less ambiguous set of refined requirements. Eventually a set of highly detailed requirements emerges from the initial idea(s). When all the requirements are gathered, they are ready to be analyzed.
b) Requirements Analysis

Requirements analysis serves to make qualitative judgments, i.e., completeness, consistency, feasibility, about the systems requirements and to ensure that everyone involved in developing the system understands the basic objective. The requirements gathered in the identification/definition phase are further broken down and analyzed. This can be performed using use cases and context diagrams to understand the why, what, and how of the system.

A use case is a sequence of events, performed through a system that results in an observable result of value for a particular actor. It describes the functional and dynamic behavior of the system. Each use case describes a particular way the system is to be used. A use case consists of actors, external events and system responses.

A context diagram is used to capture the world of interest to the system, including the actors which the system must interact. It also captures the messages and events flowing between the system and its environment.

Most systems to be built are complex and have many states. In this case, a state-transition diagram can be used to show how the system transitions from one state to the next.
The diagrams produced and the functions, attributes and constraints gathered all comes together into a requirements specification. This forms the functional baseline for the system.

The requirements specification must be communicated and agreed upon by all relevant parties. It serves as a basis for design and for test. Good requirements exhibit the following characteristics [Ref. 7]:

- Lack of ambiguity
- Completeness
- Consistency
- Traces to Origins
- Avoids design
- Requirements are enumerated

Adopting a requirements documentation technology is fairly straightforward. A standard that fits the lifecycle requirements of the system should be chosen and tailored to fit the system's specific requirements, then applied. Many standards exist that can be adopted to suit an organization's needs. A good resource is the compilation by Thayer and Dorfman [Ref. 8]. Twenty-six different requirements specifications are reprinted under one cover.
c) Requirements Management

A requirements traceability matrix provides a level of project control and assured quality that is difficult to achieve by any other means. Analyzing requirements and verifying the design is very cumbersome in a large development effort. For this reason, a requirements traceability matrix normally documents the requirements in a database. The matrix lists information for each testable requirement, its source, title, description, design component to which it is allocated, the software module that implements it, and the test reference that verifies it.

The requirements trace begins at the start of the development when the initial requirements are provided. The software requirements derived in the requirements phase are also added to the requirements trace repository.

A requirements trace is performed to ensure the requirements are satisfied in each phase of the development process, and ultimately, the system built meets the system requirements. During the maintenance phase of a system, the requirements trace provides a methodical and controlled process for managing the changes in a system.

2. Design

The design phase consists of 'how' to solve the problem. This is the period of time in the software life
cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy the system requirements established in the requirements phase.

The design phase can be broken down into two sub-phases, the top-level design and the detailed design. The top-level design is where the system architecture is defined. The detailed design defines the design of the subsystems, modules and components that make up the overall system.

The top-level design is also known as the architecture definition, the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer-based system. The top-level design forms the allocated baseline.

In defining the system architecture, six key areas, common to most development efforts, need to be addressed. This will help to refine the system architecture and establish the details of the software architecture.

- Performance: For real-time applications, the system’s timing aspects must be examined. This includes the real-time computational and operational requirements for the applicable subsystems and modules. The software development platforms, tools, and methodologies used must support the system performance requirements. For example, the effect of a CORBA platform on a distributed application’s...
performance must be determined prior to selecting the platform.

- **Error Handling:** The system must have a consistent way of handling errors. To ensure there is a consistency in error handling, a systemwide interface which provides a standardized error detection and reporting mechanism should be defined as part of the system architecture.

- **Fault Tolerance:** Based on the system requirements, the fault modes of the system should be specified and the affected modules' behavior identified.

- **Concurrency:** In the case of distributed systems, concurrency issues must be addressed and strategies incorporated to avoid deadlocks caused when multiple processes try to access an object.

- **Connectivity:** All external interfaces for the system must be identified and defined. Data bandwidth and related connectivity issues should be examined during high-level analysis and design to avoid architectural design flaws and shortcomings.

- **User Interface:** The user interface design should start in the early stages of system development. This allows customers to work with the interface early enough in the process to provide useful feedback. This also gives the interface time to evolve and mature before deploying the system.

In the detailed design phase, the system architecture is broken down into software items and further refined, and the internal interfaces between each software item are defined. The details of the elements listed in the top-level design are filled in. Enough detail must be included for the programmers to write the software.
3. Implementation

The implementation phase is also known as the coding phase. In this phase, the developer takes the detailed design where the system is broken into small configuration items and begins to implement each software configuration item in code. The code is also broken down further into software units.

Any design with more than one subroutine has a structure. The following four structural issues should be addressed when implementing a design: coupling, cohesion, information hiding, and modularity. [Ref. 9]

- **Coupling**: Coupling is a design property that states how much each unit depends on the others. Loose coupling is desired. In loose coupling, the implementation of one unit does not depend on the implementation of another.

- **Cohesion**: Cohesion describes how well the statements inside a unit relate to one another. High cohesion, which is what is desired, is when one unit performs one function or when the statements are related logically.

- **Information hiding**: Information hiding seeks to hide the details of the software design that the programmer wishes the public not to see. The most common items to hide are those that have a high probability of changing.

- **Modularity**: Modularity keeps design secrets hidden. Information hiding hides important design decisions in modules. The modules provide the software's main structure.
The procedures for breaking down the code into software units and the number of units per software configuration item is dictated by agreed coding standards. Coding standards keep the code consistent and easy for the entire team to read as well as ease life cycle maintenance.

In the implementation phase, peer reviews are conducted to ensure that each configuration item design provides the features and interfaces required to meet the configuration item's needs.

4. Test

Testing refers to the act of detecting the presence of faults in code or supporting documentation, or demonstrating their absence by confirming that requirements are met, and is distinguished from debugging where faults are isolated and corrected. An error is a mistake made by a software developer. Its manifestation may be a textual problem in the code or documentation called a fault or defect. A failure occurs when an encountered fault prevents software from performing a required function within specified limits. The definitions in this paragraph define several important testing terms and were adapted from a paper entitled, "An Examination of Selected Commercial Software Testing Tools" [Ref. 10].
Since so many software test activities are on the critical path of the development schedule, test development should be scheduled and accomplished as early as possible to be prepared for the various phases of testing in the test development lifecycle. This implies that the test effort and software development effort should be started concurrently, and that a software test development lifecycle should be identified and coordinated with the software development lifecycle. Static analysis activities apply to all phases of the software development lifecycle. One published estimate reports that 50 percent or more of the software errors are due to incorrect or modified requirements specifications. It is a well-known fact that software reviews can significantly reduce the number of errors in the later phases of development.

Testing should be considered at both the requirements analysis and design phases. Software requirements and design information provide primary input to define test requirements and prepare the test plans. This activity involves testers early on the project and helps correct requirements and design problems before they are coded when they are more expensive to fix [Ref. 11].

Typically, there are 3 to 10 failures per thousand lines of code (KLOC) for commercial software, and 1 to 3
failures per KLOC are typical for industrial software [Ref. 12]. The cost of correcting defects increases as software development progresses. For example, the cost of fixing a requirements fault during operation can be over 100 times the cost of fixing that same fault during early development stages [Ref. 13]. Consequently, timely defect detection is important.

Testing must be treated and scheduled as an integral part of the development effort. In practice, most projects sandwich the whole testing phase at the end of the development effort and before the installation phase, which has a fixed start date, so when the schedule slips, the allocated time for the testing phase shrinks and ultimately, there is almost never enough time for testing. Without the priority given to the testing phase to minimize the effects of schedule slippage and without scheduling testing throughout the development effort, timely defect detection becomes a very daunting task, if not impossible.

There are many forms of testing that can be scheduled throughout the development effort. Four test execution stages are commonly recognized: unit testing, integration testing, system testing, and acceptance testing. In unit testing, each software unit is tested in isolation, often by the developer. Unit tests provide a safety net of
regression tests and validation tests so that one can refactor and integrate changes effectively. Creating the unit test cases before the code helps even further by solidifying the requirements and improving the developer's focus. In integration testing, these software units are combined so that successively larger groups of integrated software and hardware units can be tested. System testing examines an integrated hardware and software system to verify that the system meets its specified requirements. Acceptance testing is generally a select subset of system test cases that formally demonstrates key functionality for final approval and is usually performed after the system is installed at the user's site.

Functional (black box) tests are derived from system-level, interface, and unit-level specifications. Structural (white box) tests require knowledge of the source code including program structure, variables, or both. With functional strategies, test data is derived from the program's requirements with no regard to program structure. Functional approaches are language-independent. In structural strategies, test data is derived from the program's structure.

Functional strategies can be applied at all testing levels. System tests can be defined at the requirements
analysis phase to test overall software requirements. During the design phase, integration tests can be defined to test design requirements. During the coding phase, unit tests can be defined to test coding requirements.

Prototyping is another form of testing. It evaluates or tests requirements specifications at the conceptualization phase or the requirements analysis phase and can save a considerable amount of development time when properly managed. Prototyping is becoming more widely accepted and implemented as an iterative development activity on many projects. The use of this technique is being accelerated by the availability of more automated tools that enable quicker and easier prototyping of system components.

B. SOFTWARE DEVELOPMENT USING COTS COMPONENTS

With DODD 5000.1's direction to use COTS/GOTS components in 1996 for any newly developed system, the traditional software development process has been undergoing some major changes as more and more developers transition to using COTS components. These changes must be based on facts and lessons learned in using COTS components to build systems.
With the introduction of COTS components as a way of building systems, many myths and misconceptions have to be overcome in order to effectively use COTS components. In the January 2000 issue of Crosstalk [Ref. 3], Tracz addresses some of the myths encountered when using COTS components. Some of these myths include:

- An open system architecture solves the COTS component interoperability problem.

There is no standard definition for 'open system' and 'plug and play' does not always work.

- COTS components come with adequate documentation.

COTS components come with features. Most COTS components are treated as black boxes and the documentation, such as design disclosure, required from a traditional software developer for a component is considered proprietary and never provided by a COTS supplier.

- You do not need to test COTS components.

Because you do not know the details of the design and documentation is not adequate, testing of COTS components becomes very critical to its acceptance.

- You can configure a COTS-based system to meet your requirements.

The cost of modifying COTS, or providing extra functionality is very difficult when there is little control or insight into how the COTS product was designed, developed and tested.

- If you are a large enough customer, you can influence COTS component suppliers.

In general, the marketplace drives the COTS component suppliers. The size of the supplier's
customer base determines his response to user needs. The smaller the customer base, the higher the COTS component cost and the better the service.

- COTS products are selected based on extensive evaluation and analysis.

In the past, COTS products were selected based on slick demos and good marketing. To effectively use COTS products in a software development process, this myth needs to be turned into a reality where COTS components are selected based on best fit to a set of system requirements.

In addition to those myths listed in the Crosstalk article, here are a few more which should be overcome and the truths understood when using COTS components.

The idea that COTS/GOTS components can be used to shorten the system development time is a myth. Most COTS/GOTS products are not just "plug and play". The traditional implementation phase, where software coding takes place, may be reduced when using COTS/GOTS components but when done correctly, the overall software development cycle may still consume the same amount of time. The market survey, the COTS/GOTS selection, evaluation, integration and testing process more than make up for the time reduced when using COTS/GOTS components.

When COTS vendors follow the same standards, their products can interoperate with each other is a myth. The vendors of existing products work to differentiate their product from those of competitors. This leads to a
marketplace characterized by a vast array of products and product claims, extreme quality and capability differences between products, and many product incompatibilities, even when they purport to adhere to the same standards.

Systems can be built by purchasing the right COTS components and loading them onto the target platform is a myth. COTS/GOTS software is not developed to interoperate with other COTS/GOTS/legacy software. When a system is built with these products by just loading these components onto a machine, without an integration effort, interoperability problems occur. The conglomerate of software applications on a machine does not comprise a system. A truly integrated system is greater than the sum of its parts; it provides new services and increased value.

Most phases of the traditional software development process are at least minimally affected when COTS software components are used. With COTS/GOTS components, a structured software development approach should still be followed. However, the actual tasks in each phase of the development will differ from the tasks in a traditional software development.

COTS-based development differs from the traditional software development in that the COTS selection process must occur early in the lifecycle. COTS evaluation and selection
become a critical part of the early analysis process rather than a peripheral activity within the later design process. [Ref. 14]

Unlike COTS software, software developed from scratch is designed to be interoperable. That is, it is assumed when a system is designed by one development team, the system will be internally consistent. However, in a system developed with multiple COTS/GOTS/legacy software packages, there are almost certain to be disparities in the data formats and semantics when one application has to communicate with another application or when multiple applications need to share a common data set. The challenges of COTS incompatibility, inflexibility, complexity, and transience must be addressed in the selection process because the infrastructure will ultimately consist of a suite of COTS components that must operate in harmony.

1. Requirements with COTS

The requirements phase is minimally effected when COTS products are considered in the development. The traditional activities of identification, analysis and documentation of the requirements would still apply with COTS. Unlike in the traditional software development, requirements must be developed with the use of COTS in mind. The requirements
should be flexible enough to allow maximum leverage of COTS components.

Requirements drive the COTS selection process. When using COTS, the requirements derived for the system would be used to conduct a market survey of the available COTS products for the intended system and to evaluate the potential COTS products to be integrated in the system.

A market survey is conducted to see what is available on the commercial market to meet the requirements of the system. In most instances, there will not be a single product in the commercial market that will meet all the system requirements for any system of appreciable complexity. For this reason, the requirements for a system should be tailored. Those requirements intended to be fulfilled by COTS product(s) should form the basis for the market survey. These requirements should be prioritized and each requirement should indicate a threshold and objective range, where the threshold is the absolute minimum requirement and the objective is the ultimate requirement. This range provides a comparison of the candidate COTS products for selection.

2. Design with COTS

In the architecture definition phase, a system architecture is developed for the overall system. With the
use of COTS, the design requirements and guidelines that will minimize dependencies between the COTS products and the components under development must be specified to reduce the effect of COTS product upgrades.

COTS must usually be treated as black boxes, or at best very opaque boxes. As a result, many properties of the COTS product need to be discovered through systematic investigation. This is referred to as qualification.

Once the COTS products are identified in the market survey, the candidate COTS products must be qualified. The qualification process consists of determining 'fitness for use' of previously developed components that are being applied in a new system context. Qualification is also a process of selecting products in a marketplace of competing products.

There are two phases of qualification: discovery and evaluation. In the discovery phase, an evaluation copy of the COTS product is obtained and run to identify and match the properties of a COTS product to the system's requirements. Some of the properties include functionality, data types supported and other aspects of a software component's interface. These properties also include quality aspects that are usually more difficult to isolate, such as reliability, predictability, and usability.
In some circumstances, it is also reasonable to discover 'non-technical' properties, such as the vendor's market share, past business performance, and process maturity of the developer's organization. The non-technical properties help to identify the maintenance risks of the product.

Once the relevant properties of a COTS product have been discovered, it is possible to identify which properties exhibited by a component are in conflict with other components, or with a system design. This is the evaluation phase.

Using the properties discovered in the earlier phase, each of these properties are prioritized, evaluated and compared amongst the COTS product candidates. The selected COTS products for an intended system should be a 'best fit' as it will be rare, if not impossible, for a COTS product to match the intended system requirements exactly. The conflicts, or mismatches, must be repaired through component adaptation in the implementation phase.

3. Implementation with COTS

When using COTS, the biggest impact is to the implementation phase. The traditional coding that occurs during the implementation phase may be minimal when COTS software is used. In contrast to traditional development,
where system integration is often the tail end of an implementation effort, COTS software integration is the centerpiece of the approach. Thus, implementation has given way to integration as the focus of system construction.

Because individual COTS software components are written to different requirements, and are based on differing assumptions about their context, the COTS software must often be adapted when used in a new or integrated system. Normally, this adaptation is performed by developing the middleware (glue and wrapper code) to enable the COTS/GOTS/legacy software to intercommunicate.

The middleware software must be adapted based on rules that ensure conflicts among the different software components are minimized. The degree to which a software component's internal structure is accessible suggests different approaches to adaptation:

- **white box**, where access to source code allows a software component to be significantly rewritten to operate with other software components.

- **opaque box**, where source code of a software component is not available for modification but the software component provides its own extension language or application programming interface (API).

- **black box**, where only a binary executable form of the software component is available and there is no extension language or API.
Each of these adaptation approaches has its own advantages and disadvantages; however white box approaches can result in serious maintenance and evolution concerns in the long term. Wrappers and glue code are specific middleware programming techniques used to adapt opaque and black box components.

Once the mismatches between components have been removed, it is possible to assemble them into systems, and to evolve the system through re-assembly with different components. This is often referred to as COTS integration. This process may be performed repeatedly until the best solution is attained.

4. **Testing with COTS**

With COTS, testing at both the requirements analysis and design phases becomes very critical to the success of using COTS in a system. When testing is performed early on the COTS component, risks can be mitigated in a timely manner. Early testing can make the difference between a successful project using COTS and a failed project.

Similar to unit testing when software is developed from scratch, prior to integrating COTS/GOTS components into a system, each of the COTS/GOTS components should undergo its own independent testing. Testing should be performed on the COTS/GOTS products to ensure that it fulfills the specified
system requirements, the COTS/GOTS components function as advertised and catastrophic faults are detected as early in the testing phase as possible so that timely corrective action can be taken. Timely corrective action may include reporting the problem back to the vendor to get the problem(s) fixed, deciding to disqualify the COTS/GOTS component, creating a workaround, or building a wrapper to hide the function causing the fault.

Testing becomes more nebulous with COTS because we do not know what the product really does i.e. cannot view the source code. Because of this, COTS must be treated as a black box and the inputs/outputs to/from the black box must be thoroughly tested to ensure correctness. All the dependencies between COTS/GOTS/legacy components must be tested and verified. Testing of the interfaces between the components is critical to ensure system interoperability. The output from one component must be consistently interpreted when provided as input and manipulated in the next component.

Prototyping is used to determine the overall feasibility of the system and also to demonstrate the user interface to be provided with the system. Prototyping is especially useful when using COTS to allow early user
feedback on the COTS components, especially when the COTS components contain GUIs.
III. ARCHITECTURAL FRAMEWORK

A. INTRODUCTION

In the design phase of the COTS software development process, a target architecture needs to be formulated where multiple COTS/GOTS components and legacy software can operate seamlessly together to form a system. In order to successfully create a target architecture for an interoperable COTS/GOTS/legacy system, an architectural framework must exist to draw from.

Many architectural frameworks are available on the market today to develop a target architecture for an Information Technology (IT) system - Technical Architecture Framework for Information Management (TAFIM), The Open Group Architectural Framework (TOGAF), the Open Group's Distributed Computing Environment (DCE), IEEE 1003.0, the NCR Enterprise Architecture Framework, and the Defense Information Infrastructure Common Operating Environment (DII COE), to name a few. Each architectural framework has its own specific focus.

DoD's TAFIM provides guidance for the evolution of the DoD technical infrastructure. It is currently replaced by the Joint Technical Architecture (JTA). TAFIM provides the services, standards, design concepts, components, and
configurations that can be used to guide the development of technical architectures that meet specific mission requirements [Ref. 15].

TOGAF is a tool for defining an IT architecture. TOGAF was developed by members of The Open Group, working within the TOGAF Program. The original development of TOGAF in 1995 was based on the TAFIM, developed by the US Department of Defense.

The essentials of TOGAF consists of three parts, the architectural development model (ADM), the foundation architecture and the resources [Ref. 16].

Another architectural framework developed by The Open Group includes the Distributed Computing Environment (DCE), which provides a set of services that can be used as the basis of a DCE-Centric Architecture related to TOGAF [Ref. 17].

IEEE 1003.0, also known as POSIX 1003.0, is an architectural framework built on open systems standards.

The NCR Enterprise Architecture Framework is based on NCR's architecture practice Global Information Technology Planning (GITP), and NCR's architecture model Open Cooperative Computing Architecture (OCCA 6). The NCR Enterprise Architecture Framework is created to guide the
development of systems, industry, and customer specific architectures.

The DII COE architectural framework seeks to build systems around a common operating environment using reusable software components called segments to function in a joint arena, promoting data sharing within systems and between systems. It implements a 'plug and play' open architecture designed around a client/server model [Ref. 18].

Many of these architectural frameworks were developed prior to the demand for using COTS in systems integration and during a time when stovepipe systems were built. Because of this, the existing architectural frameworks available today do not sufficiently address the integration of an interoperable, distributed, heterogeneous system architecture composed of COTS/GOTS/legacy components.

1. Architecture vs. Architectural Framework

The terms, architecture and architectural framework are often used interchangeably when in fact, there is a clear distinction between the two terms. Before deeply progressing into this chapter, the definitions of architecture vs. architectural framework needs to be defined to clarify the distinction.
The Open Group defines Architecture as follows:

Specifically, an architecture is a formal description of an information technology (IT) system, organized in a way that supports reasoning about the structural properties of the system. It defines the components or building blocks that make up the overall information system, and provides a plan from which products can be procured, and systems developed, that will work together to implement the overall system.

The Open Group provides an analogy to improve understanding of the architecture definition:

One of the most common approaches to answering the question of what is an architecture is to draw an analogy between the architecture of buildings and the architecture of computing systems. While there are a number of things wrong with this kind of analogy, it can serve to illustrate basic concepts.

Actually, in today's highly distributed computing environments, a better analogy is with the discipline of Planning or Urban Design rather than with Architecture as applied to individual buildings.

On this basis, then, the difference between a computer system with an architecture and one without, is the difference between an urban sprawl that has just grown willy-nilly, and a township where thought has been given to the ability to link buildings and districts together to form whole communities that function well and serve the needs of its inhabitants.

Continuing The Open Group’s architecture analogy, like the urban plans used for a township to build whole functioning districts and communities, an architectural
framework is like the building standards and zoning laws that exist to guide the urban development architecture in city planning.

The Open Group defines architectural framework as:

An architectural framework is a tool which can be used for developing a broad range of different IT architectures. It should describe a method for designing an information system in terms of a set of building blocks, and for showing how the building blocks fit together. It should contain a set of tools and provide a common vocabulary. It should also include a list of recommended standards and compliant products which can be used to implement the building blocks.

www.opengroup.org/public/arch

2. Importance Of An Architecture & An Architectural Framework

As systems become larger and increase in complexity as in C4ISR systems, it becomes even more important to have an architecture. A software architecture improves our ability to effectively construct large-scale software systems [Refs. 19, 20]. An architecture allows us to communicate effectively to the various participants in a development activity what to build, what it must do, and how to build it [Ref. 21].

Not only does the architecture allow us to effectively construct software systems but it also allows us to effectively maintain the system once it is built. The
absence of a thoughtful architectural act assures that there is no initial accommodation to the changes that will propel the product from version to version. This is especially important in an evolutionary software development process and in today's time of COTS usage in software development. With an evolutionary development process, each cycle of an evolution requires additional capabilities. Without an architecture to support this, each cycle would entail significant changes to the existing system structure in order to tack on the additional capabilities. With COTS usage in software development, the architecture also must minimize dependencies between the COTS products and the components under development to reduce the effects of COTS product upgrades/replacements throughout the evolutionary cycle.

Other motivations for a software architecture include [Ref. 21]:

- To support analysis and prediction prior to creating concrete solutions.

- To move from expensive point solutions to solution families (analogous to product lines from the software vendor community) that emphasize common parts and reuse.

- To give guidance and controlling information analogous to commercial building codes which benefit the community such as standards to be followed,
preplanned interoperability requirements, use of COTS software products, etc.

Using an architectural framework will speed up and simplify architecture development, ensure more complete coverage of the designed solution, and make certain that the architecture selected allows for future growth in response to the changing needs of the business. [Ref. 16]

Neither adhering to a structured COTS software development process nor the design of an architecture by themselves can guarantee an interoperable system without implementing the tools from an architectural framework that addresses the system interoperability issues.

B. TOGAF

TOGAF is a tool for defining an IT architecture. TOGAF was developed by members of The Open Group, working within the TOGAF Program. The original development of TOGAF in 1995 was based on the TAFIM, developed by the US Department of Defense.

TOGAF consists of three parts, the architectural development model (ADM), the foundation architecture and the resources [Ref. 16].
1. **Architectural Development Model**

The Architecture Development Method (ADM) is the core of TOGAF. It describes the steps to follow in developing an IT architecture. The ADM is a cyclic process consisting of seven phases.

- **Initiation and Framework** - Identify requirements, initiate architecture development cycle.

- **Baseline Description** - Capture relevant existing environment to build technical architecture, including a description of the current system and its functions, statements of the constraints imposed by the internal organization and external environments, assumptions, and current architectural principles embodied in the current system.

- **Target Architecture** - Define target architecture. In defining the target architecture, eight steps are involved.
  
  - Represent the baseline using the Open Group framework to give a common starting point.
  
  - Consider a number of architectural views to ensure all aspects of system are considered.
  
  - Select a high-level model for the architecture.
  
  - Select the services required.
  
  - Identify and validate business goals are met.
  
  - Establish a set of criteria for service selection.
  
  - Define the architecture in detail.
  
  - Conduct a gap analysis.
• Opportunities and Solutions - Evaluate and select major work packages. This forms the basis for the implementation.

• Migration Planning - Prioritize work, cost/benefit analysis, develop outline plan.

• Implementation - Develop full plan and execute.

• Architecture Maintenance - Establish procedure for maintenance of new baseline. The maintenance procedure for the new baseline will typically provide for continual monitoring of new developments in technology and changes in business environment, and for determining whether to formally initiate a new architecture evolution cycle.

2. Foundation Architecture

The TOGAF Foundation Architecture comprises the TOGAF Technical Reference Model (TRM) and The Open Group's Standards Information Base (SIB). The Foundation Architecture is an architecture of generic services and functions that provides a foundation on which more specific architectures and architectural components can be built.

TOGAF's TRM is derived from the TAFIM TRM. The TRM provides a model (figure 1) and taxonomy of generic platform services. It focuses on the services and structure of the underlying platform necessary to support the use of applications, by centering on the interfaces between the platform and supported applications, and between the
platform and the external environment. TOGAF’s TRM contains three entities:

- Application Software.
- Application Platform.
- Communications Infrastructure.

and two interfaces:

- Application Platform Interface.
- Communications Infrastructure Interface.
Figure 1 - Technical Reference Model From Ref. [16]
There are two categories of application software in the TRM: the infrastructure applications and the business applications. The infrastructure applications provide general purpose functionality such as word processing, spreadsheet, electronic mail applications, and presentation software. Business applications are tailored applications that meet a particular enterprise's needs, i.e. in the C4I world, an application to track the routes of a deployed unit.

The Application Platform in the TOGAF Technical Reference Model is a single, generic, conceptual entity. From the viewpoint of the TOGAF TRM, the Application Platform contains all possible services. These services include:

- Data Interchange Services.
- Data Management Services.
- Distributed Computing Services.
- Graphics and Imaging Services.
- International Operation Services.
- Network Services.
- Operating System Services.
- Security Services.
- Software Engineering Services.
- System and Network Management Services.
- Transaction Processing Services.
- User Interface Services.

In a specific target architecture, the Application Platform will contain only those services needed to support the required functions and many will include additional services to support application software specific to the organization.

The Application Platform for a specific target architecture will typically not be a single entity, but rather a combination of different entities for different, commonly required functions, such as Desk Top Client, File Server, Print Server, Application Server, Internet Server, Data Base Server, etc., each of which will comprise a specific, defined set of services necessary to support the specific function concerned.

The Communications Infrastructure provides the basic services to interconnect systems and provide the basic mechanisms for opaque transfer of data. It contains the hardware and software elements which make up the networking and physical communications links used by a system, and of course all the other systems connected to the network. It deals with the complex world of networks and the physical
communications infrastructure, including switches, service providers and the physical transmission media.

The Application Platform Interface specifies a complete interface between the Application Software and the underlying Application Platform across which all services are provided. A rigorous definition of the interface results in application portability, provided that both platform and application conform to it. For this to work, the API definition must include the syntax and semantics of not just the programmatic interface but also all necessary protocol and data structure definitions.

The Communications Infrastructure Interface is the interface between the application platform and the communications infrastructure.

The SIB provides a database of standards that can be used to define the particular services and other components of an organization specific architecture that is derived from the TOGAF Foundation Architecture.

3. Resources

The resources comprise the TOGAF Resource Base - a set of tools and techniques available for use in applying TOGAF and the TOGAF ADM. It contains the Architecture Description Markup Language (ADML) and the Architectural Views. ADML is used to define architectural building blocks in a way that
allows their interactions with other building blocks to be captured. Architectural Views provides different views for consideration when developing an architecture. These views include function, security, management, builder’s, data management, user, computing, and communications.
IV. PROPOSED ARCHITECTURAL FRAMEWORK

Now that DoD has moved to building distributed, heterogeneous, C4ISR systems with COTS software, the need for a new architectural framework to integrate COTS/GOTS/legacy software is quite evident. The existing architectural frameworks on the market today do a very good job at addressing the building of traditional stovepipe software systems where the interoperability of distributed, COTS/GOTS/legacy systems is not an issue. Most lightly talk about COTS but do not sufficiently address the details required or the tools necessary for developing interoperable systems and architectures with COTS/GOTS/legacy components. TOGAF addresses interoperability but limits it to the communications infrastructure interface and does not address the more difficult issue of data interoperability, the correct interpretation from one system to another of the semantics of the data.

Of the three architectural frameworks mentioned, TOGAF is the most manageable and comes closest in addressing the needs of distributed, heterogeneous systems but it also lacks the details and the tools necessary for a COTS development in this environment. For this thesis, TOGAF is
chosen as the architectural framework to enhance to address distributed, heterogeneous systems.

The proposed architectural framework uses TOGAF as a basis and seeks to address the interoperability issues encountered when building target architectures where integrating COTS/GOTS/legacy components to form a distributed, heterogeneous system is essential. It builds upon the TOGAF foundation and includes additional tools like the Object Request Brokers (ORBs), middleware (glue and wrapper software) and Extensible Markup Language (XML) to assist in the development of interoperable COTS/GOTS/legacy systems.

A. TOGAF UPDATES

In TOGAF, the Architectural Development Model is prescriptive, where detailed guidelines and procedures for building a target architecture are provided. The guidelines and procedures do not address the building of an architecture using COTS/GOTS/Legacy components and superficially address interoperability by just stating interoperability issues should be listed in the second phase of the architectural development cycle, baseline description, but does not specifically address how to identify or resolve these interoperability problems.
This thesis will use the existing TOGAF and modify it to incorporate the specific guidelines and procedures necessary to build and maintain an architecture using COTS/GOTS/Legacy components. The bulk of the procedural changes are concentrated in TOGAF's Architectural Development Model.

In the first phase of the architectural development model, Initiation and Framework, the inputs include the business goals, financial constraints, time limits, mission statement, strategic plans and current business system description. In addition to these inputs, when COTS/GOTS products are a consideration, the architectural framework also should include a process for conducting a market survey to evaluate what is available on the market to meet the needs of the system.

DoD Regulation 5000.2-R, which implements DoD Directive 5000.1 and provides policies and procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information Systems (MAIS) Acquisition Programs, encourages market research [Ref. 22].

Market research and analysis shall be conducted to determine availability and suitability of existing commercial and non-developmental items prior to the commencement of a development effort, during the development effort, and prior to the preparation of any product description.
The market survey is conducted based on high level requirements for the intended system. A market survey should be conducted to gather the list of available COTS/GOTS candidates for the system. The output of the Initiation and Framework phase should include the list of candidate COTS/GOTS components.

The second phase, baseline description, consists of the high level description of the characteristics of the current system including its architecture. In this phase the current system is evaluated to determine if there are any existing interoperability issues needed to be addressed, lessons to be learned, items to be reused, and serves as a starting point for the system requirements and architecture. The only change to this phase is the order. This phase should precede the Initiation and Framework phase as the functions listed for the baseline description phase should be performed prior to the start of looking at the architectural framework for the target system. When a current system does not exist, this step is skipped.

The third phase, Target Architecture, is where the target architecture for the system is identified. In this phase, a number of architectural views are reviewed to ensure all aspects of the system are considered. Unlike
newly developed software, the upgrade cycle of COTS/GOTS components are not within the system developer's control. Thus, when using COTS/GOTS components, an architecture which minimizes dependencies between the COTS/GOTS products and the components under development to reduce the effect of COTS/GOTS product upgrades should be considered.

Based on the services required of the COTS/GOTS components, a tradeoff analysis should be conducted amongst the candidate COTS/GOTS components. To aid the tradeoff analysis, a COTS/GOTS selection and evaluation plan is created in this phase. For specific details on what should be addressed in the evaluation/selection process, see appendix A.

Because component-based architecture development is a relatively new field, systems integrators still struggle with methods to keep abreast of technology advances and ways to determine which products best suit their needs. In the rush to make a decision, the choice of COTS products is not made based on a strong business case or the total ownership cost. [Ref. 3]

By following a formal COTS/GOTS selection process, the mistake of blindly rushing to make a decision of which COTS/GOTS product to use can be avoided. A formal selection process based on system requirements is used to mitigate the
risks associated with the acquisition of COTS/GOTS products. The COTS/GOTS selection process and the evaluation plan are both created during the design phase of a COTS software development process once the architecture is first articulated, identifying the use of COTS/GOTS components. The system requirements, derived during the requirements phase, feed the COTS/GOTS market survey, selection process and evaluation plan. When using COTS/GOTS components in a software development process, the requirements for a COTS software development process should be flexible enough to allow maximum leverage of COTS/GOTS components.

Once the candidate COTS/GOTS components are evaluated and a set selected, a model of prospective building blocks, including COTS, GOTS, legacy and newly developed components, a detailed architecture can be created to show the interrelationships of the components.

Opportunities and Solutions is the fourth phase of the Architectural Development Model. In this phase, work packages are developed to implement the proposed architecture. With COTS/GOTS components in the architecture, the work packages must be developed with the COTS/GOTS components in mind to ensure that the work packages include work to integrate and test the COTS/GOTS components. Traditionally, work packages consist of software coding but
with a COTS/GOTS development, integration and testing should also be included in work packages. Once the individual work packages are completed, the individual components will then integrate and interoperate to form a system.

Migration planning is the fifth phase of the architecture development model. In this phase, the individual work packages are prioritized. Outline plans should include how the individual work packages will integrate with COTS/GOTS components including standards to be used. In the prioritization of the individual work packages, testing of each integration piece should be performed prior to the integration of the whole system. When using COTS and GOTS, integration testing with newly developed code and legacy components needs to be incorporated early in the development process to identify the potential individual interoperability problem.

Architecture maintenance is the last phase of the architecture development cycle. The purpose of this phase is to establish a procedure for the ongoing maintenance of the architecture. To maintain an architecture consisting of COTS, GOTS, and legacy components requires a continuous monitoring and analysis of potential candidates available in the marketplace to replace the existing COTS, GOTS, and legacy components. Some GOTS and legacy components may
never be replaced as their functionality may be very specialized, but at a minimal, an evaluation of COTS components should be regularly conducted. The output from this phase should include an up-to-date list of COTS/GOTS upgrades.

Over time and in evaluating these components, a change in the existing target architecture may need to be considered. If the target architecture requires a change, a new architectural evolution cycle is initiated.

Table 2 summarizes the proposed modifications (in bold, italics) required during each phase of the architecture development cycle to address the interoperability of heterogeneous systems.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline Description</td>
<td>• Existing system Description</td>
<td>• Clear description of the current system and its functions</td>
</tr>
<tr>
<td></td>
<td>• Existing Architecture</td>
<td>• Assumptions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Lessons Learned</td>
</tr>
<tr>
<td>Initiation and Framework</td>
<td>• Business goals</td>
<td>• Results of Market Survey</td>
</tr>
<tr>
<td></td>
<td>• Strategic plans</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Mission statement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Financial constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Schedule constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Current business system description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Plan for remaining phases</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Market survey</td>
<td></td>
</tr>
<tr>
<td>Target Architecture</td>
<td>• Baseline description from Phase 1</td>
<td>• A baseline systems requirements in</td>
</tr>
<tr>
<td>Opportunities and Solutions</td>
<td>TOGAF terms</td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A full description of the baseline system and the proposed architecture from all relevant views.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A description of the services selected and a detailed architecture definition of the standards used to implement these services</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Candidate building blocks selected (including COTS, GOTS, Legacy and newly developed components)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interoperability issues derived from different architectural views</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gap analysis</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Migration Planning</th>
<th>Work packages (includes integration and test, glue and wrapper code to address interoperability issues)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prioritized work packages</td>
</tr>
<tr>
<td>Current architecture</td>
<td>Prioritized work packages</td>
</tr>
<tr>
<td>Analysis of how proposed architecture meets business goals and objectives</td>
<td>Prioritized work packages</td>
</tr>
<tr>
<td>Organizational information</td>
<td>Prioritized work packages</td>
</tr>
<tr>
<td>COTS/GOTS product information</td>
<td>Prioritized work packages</td>
</tr>
<tr>
<td>Standards information</td>
<td>Prioritized work packages</td>
</tr>
<tr>
<td>Other specification information</td>
<td>Prioritized work packages</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architecture Maintenance</th>
<th>COTS/GOTS Product Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Updated list of COTS/GOTS upgrades</td>
</tr>
</tbody>
</table>

Table 2 - Proposed Architecture Development Cycle
B. TOOLS FOR THE PROPOSED ARCHITECTURAL FRAMEWORK

Using the TOGAF Resource Base as the basic tool set for the proposed architectural framework, additional tools are required to address the specific integration issues with COTS/GOTS/legacy software components and the interoperability of systems composed of these software components.

Interoperability is the ability of systems to provide services to and accept services from other systems, and to use the services so exchanged to enable them to operate effectively together. Many COTS/GOTS systems and products are not developed for Joint Operations. Each of these systems usually performs only the specific task that drove its design.

The additional tools that this thesis will address include object request brokers, wrappers, glue code and the Extensible Mark Up Language (XML). Wrapper, and glue code are tools to address two different aspects of system integration with COTS/GOTS/legacy software components while an object request broker is a tool to address system interoperability and XML is a tool to address data interoperability.
1. Object Request Brokers

The foundation for the changes in TOGAF is the basis from which COTS/GOTS/legacy software is integrated and interoperable. It consists of a hybrid of ORBs.

ORBs encapsulate the provided integration/interoperation services to make them independent of implementation details such as computing hardware, operating system, programming language, and data representation. ORBs must rely on the surveyed building blocks to realize the services they provide. The difference in using an ORB and using the building blocks directly is that the application programs (such as those in C4ISR systems) interact with standardized interfaces rather than directly with the building blocks underneath those interfaces. This decouples the applications from the mechanisms used to realize the needed services, and makes it much easier to take advantage of new approaches to realize the building blocks as technology improves [Ref. 23].

A hybrid is recommended for this architectural framework to enable the broadest application amongst the various IT target architectures and systems. Any one target architecture may choose to use a single ORB for their application. However, for heterogeneous systems, the ORBs currently available in the commercial market today are not
yet mature enough to carry a wide product line to encompass multiple platforms although they each are striving to run on multiple platforms. The available ORBs on the marketplace consist of Microsoft’s COM/DCOM, Sun Microsystems’s Enterprise Java Beans and the Object Management Group’s (OMG) CORBA.

a) COM/DCOM

COM has its roots in OLE version 1, which was created in 1991. It is a proprietary document integration and management framework for the Microsoft Office suite. Microsoft later realized that document integration is just a special case of component integration. OLE version 2, was later released in 1995 with a major enhancement over its predecessor. The foundation of OLE version 2, now called COM, provides a general-purpose mechanism for component integration on Windows platforms. While this early version of COM include some notions of distributed components, more complete support for distribution became available with the DCOM specifications and implementations for Windows95 and Windows NT released in 1996. Beta versions of DCOM for Mac, Solaris and other operating systems followed shortly after.

Microsoft’s Distributed Component Object Model (DCOM) is an extension of COM to support communication among
objects on different computers - on a LAN, a WAN, or the Internet. It is based on The Open Group's DCE RPC specification and will work with both Java applets and ActiveX components through its use of COM. With DCOM, an application can be distributed at locations that make the most sense to the customer and to the application.

The DCOM protocol is an application-level protocol for object-oriented remote procedure calls useful for distributed, component-based systems of all types. It is a generic protocol layered on the distributed computing environment (DCE) RPC specification and facilitates the construction of task-specific communication protocols through features such as: a platform neutral argument/parameter marshaling format (NDR), the ability for objects to support multiple interfaces with a safe, interface-level versioning scheme suited to independent evolution by multiple parties, the ability to make authenticated connections and to choose levels of channel security, and a transport-neutral data representation for references (including by-value) to objects.

With DCOM, the following issues are addressed [Ref. 24]:

- Location independence
• Connection Management
• Scalability
• Performance
• Bandwidth and latency
• Security
• Load Balancing
• Fault tolerance
• Protocol neutrality
• Platform neutrality
• Seamless integration with other Internet protocols

(1) Location Independence. When a distributed application is implemented on a real network, several conflicting design constraints must be considered:

• Components that interact more should be ‘closer’ together.

• Some components can only be run on specific machines or at specific locations.

• Smaller components increase flexibility of deployment, but they also increase network traffic.

• Larger components reduce network traffic, but they also reduce flexibility of deployment.

These critical design constraints can be overcome with DCOM relatively easily because the details of
deployment are not specified in the source code. DCOM completely hides the location of a component, whether it is in the same process as the client or on a machine on the other end of a network. In all cases, the way the client connects to a component and calls the component’s method is identical. A simple reconfiguration changes the way components connect to each other.

(2) Language Neutrality. As an extension of COM, DCOM is completely language independent. Virtually any language can be used to create COM components. Language independence also enables rapid prototyping: components can be first developed in a higher-level language to show proof of concept and later reimplemented in a different language that can better take advantage of advanced features such as DCOM’s free threading, free multithreading and thread pooling.

(3) Connection Management. Network connections are inherently more fragile than connections internal to a machine. Components in a distributed application need to be notified when a client is no longer active. DCOM manages connections to components by maintaining a reference count on each component. Each
connection to a component increments that component's reference count. This is part of DCOM's distributed garbage collection mechanism, which functions transparently to the application.

DCOM uses a pinging protocol to detect if clients are still active. Client machines send a periodic message. DCOM considers the connection broken when three consecutive ping periods pass without a response at which point DCOM decrements the reference count and releases the component when the count has reached zero. Whether the connection breaks due to a catastrophic network or hardware failure, or a client disconnecting, the same reference counting mechanism is used.

(4) Scalability. It is important that distributed applications have the ability to grow with the number of users, the required functionality, and the amount of data. DCOM provides a couple of features to enhance an application's scalability:

- Symmetric Multiprocessing (SMP)
- Flexible Deployment

DCOM takes advantage of Windows NT support for multiprocessing. For applications that use a
free-threading model, DCOM manages a thread pool for incoming requests. On multiprocessor machines, this thread pool is optimized to the number of available processors. DCOM shields the developer from the details of thread management and delivers the optimal performance that only costly hand coding of a thread pool manager could provide.

Flexible deployment allows an application to move to another machine as the load on one machine increases. DCOM’s location independence makes it easy to redistribute components over other computers, offering an easier and less expensive route to scalability.

DCOM’s location independent programming model makes it easy to change deployment schemes as the application grows. Initially a single server machine can host all the components. As the demand on the server grows, other machines can be added, and the components can be distributed among the machines without any code changes.

Applications also need to scale as new features are added and other features modified. DCOM provides a versioning scheme (described in COM section), which allows clients to dynamically query the functionality of a component. Instead of exposing its functionality as a single, monolithic group of method and properties, a COM component can appear differently to different clients. A
client that uses a certain feature needs access only to the methods and properties it uses. Clients can also use more than one feature of a component simultaneously. If other features are added to the component, they do not affect an older client that is not aware of them.

The initial component exposes a core set of features as COM interfaces, on which every client can rely on. As the component acquires new features, most of the core features will still be necessary; new functions and properties appear in additional interfaces without changing the original interfaces at all. Old clients can still access the core set of interfaces while new clients can query for the presence of new interfaces and use them when available, or they can gracefully degrade to the old interfaces.

(5) Performance. With any new technology, there are tradeoffs. Microsoft's DCOM provides a standardized wire-protocol and programming model for the developer so that application-specific custom protocols need not be developed to interface with local and remote applications. However, there is a performance degradation when using DCOM (as high as 35% overhead for remote calls).
(6) Bandwidth and Latency. A common problem in designing distributed applications is an excessive number of network round trips between components on different machines. On the Internet, each of these round trips incurs a delay of approximately 1 second, often significantly more.

DCOM reduces network roundtrips to avoid the impact of network latency. A common technique for reducing the number of network round trips is to bundle multiple method calls into a single method invocation. DCOM uses this technique extensively for tasks such as connecting to an object or creating a new object and querying its functionality. The disadvantage of this technique for general components is that the programming model changes significantly between the local and the remote case.

To reduce the number of network round trips, DCOM also uses the connectionless UDP subset of the TCP/IP protocol suite. This protocol allows DCOM to perform several optimizations by merging many low-level acknowledge packages with actual data and pinging messages. Advantages with DCOM are also obtained when running over connection-oriented protocols.
(7) Security. Using the network raises new issues related to security between and among clients and components. Since many operations are now physically accessible by anyone with access to the network, access to these operations has to be restricted at a higher level.

DCOM can make distributed applications secure without any specific security-specific coding or design in either the client or the component. Just as the DCOM programming model hides a component’s location, it also hides the security requirements of a component. The same binary code that works in a single-machine environment, where security may not be a concern, can be used in a distributed environment in a secure fashion.

DCOM achieves this security transparency by letting developers and administrators configure the security settings for each component. Just like the Windows NT File System lets administrators set access control lists (ACL) for files and directories, DCOM stores ACL for components and the ACL can be configured using the DCOM configuration tool.

The Windows 2000 will have a Kerberos-based security provider allowing even more advanced security controls like regulating what components can do while impersonating clients. This security provider also requires
fewer resources for performing authentication than the original Windows NT security provider-NTLM.

Windows 2000 will also include public-key based security. The public-key security makes it possible to decentralize the management of security credentials with any Windows NT application, including DCOM applications. Authentication with public keys is less efficient than with private keys but it allows authentication without storing the client's private credentials.

(8) Load Balancing. As a distributed application becomes more successful, the user demand increases for all the components of the application. Sometimes, even the fastest hardware is not enough to keep up with the user demand. It is at this point that the load must be redistributed among multiple machines.

One method of load balancing is to permanently assign certain users to certain servers running the same application. This method is called static load balance because the load does not change with conditions on the network or other factors. DCOM applications can use static load balancing by simply changing a registry entry.
An alternative flexible approach is to use a dedicated referral component, residing on a publicly-known server machine. Client components connect first to this component, requesting a reference to the service required. The referral component can use DCOM's security mechanisms to identify the requesting user and choose the server depending on who is making the request. The referral component can actually establish a connection to this server and return it directly to the client. DCOM then transparently connects the client directly to the server. This mechanism can also be completely hidden from the client by implementing a custom class factory in the referral component.

As user demand grows, administrators can change the components to transparently choose different servers for different users. Client components remain entirely unchanged, and the application can migrate from a model whose administration is decentralized to a centrally administered approach.

Static load balancing is a good technique for dealing with growing user demand, but it requires the intervention of an administrator and works well only for predictable loads.
The referral component can be used to provide more intelligent load balancing. Instead of just basing the choice of server on the user ID, the referral component can use information about server load, network topology between client and available servers, and statistics about past demands of a given user. With this information, every time a client connects to a component, the referral component can assign it to the most appropriate server available at the moment. This method is called dynamic load balancing.

DCOM does not provide support for dynamic reconnection and distribution of method invocations, since doing so requires an intimate knowledge of the interaction between client and component. The component typically retains some client-specific status information (i.e. the state of the client) between method invocations. If DCOM dynamically reconnected the client to a different component, this information would be lost. However, DCOM makes it easy for application developers to introduce this logic explicitly into the protocol between client and component.

(9) Fault Tolerance. Graceful fail-over and fault tolerance are vital for mission critical
applications that require high availability. DCOM provides basic support for fault tolerance at the protocol level through the pinging mechanism described in the section, Connection Management.

Also, with the referral component described above, clients are able to reconnect to the same referral component that established the first connection when it detects a failure. The referral component retains information about which servers are no longer available and automatically provides the client with a new instance of the component running on another machine.

(10) Protocol Neutrality. Many distributed applications are integrated into a customer's existing network infrastructure. If these applications required a specific network protocol, all the client applications would need to be upgraded. This would be completely unacceptable. Developers of distributed applications need to keep the application independent of the underlying network infrastructure.

DCOM provides this abstraction transparently. DCOM can use any transport protocol, including TCP/IP, UDP, NetBIOS, and IPX/SPX. This feature is especially attractive in the Defense community where
firewalls prevent certain protocols access. DCOM also provides a security framework on all these protocols. Developers are able to use these features of DCOM to remain network protocol neutral.

(11) Platform Neutrality. The DCOM architecture allows the integration of platform-neutral development frameworks and virtual machine environments (Java), as well as high-performance, platform-optimized custom components into a single distributed application.

(12) Seamless Integration with Other Internet Protocols. Distributed applications can take advantage of the Internet in many different ways. Virtual private networks such as the Window NT 4.0 Point-to-Point Tunneling Protocol (PPTP) are one way of using the network to securely tunnel private information over the Internet. DCOM applications will transparently leverage this technology.

Since DCOM is an inherently secure protocol, it can be used without being encapsulated in a virtual private network. DCOM applications can use the cheap, global TCP/IP network. Most dedicated server machines are hidden behind a firewall that typically
consists of protocol level and application level filters. DCOM is able to work with both classes of firewalls.

- DCOM uses a single port for initiating connections and assigns a configurable range of ports to the actual components running on a machine.

- Application level proxies can easily be built. They can be either generic (forwarding configurable DCOM activation and method calls) or application specific.

- Server administrators can also choose to tunnel DCOM through HTTP, effectively bypassing most of today’s firewalls.

With this range of options, DCOM applications can use the Internet for private connectivity locally or with external clients anywhere in the world.

b) CORBA

CORBA is the first to bring ORBs to the marketplace. CORBA is a set of specifications (not an implementation) for the development of ORBS.

When the CORBA specification was first developed, its focus was on interoperability. At this writing, the CORBA 3 specification is getting ready to enter the marketplace and it will take a while before we see ORBs that are CORBA 3 compliant widely available in the marketplace.
The specifications follow Object Oriented Design (OOD) principles of encapsulation, inheritance, polymorphism and instantiation.

Currently there are about 70 CORBA 2.0/2.1 ORBs available in the marketplace.

The CORBA specification defines seven major ORB components:

- Interface Definition Language (IDL).
- Interface Repository (IR).
- Implementation Repository.
- Dynamic Invocation Interface (DII).
- Static Invocation Interface (SII).
- Object adapters.
- ORB core.

IDL is a special language used by a developer to describe object interfaces (operations names and parameter names and types). It provides a programming language-independent mechanism to define interfaces to objects.

The Interface Repository stores and manages object interface information (a collection of object definitions specified in IDL). The Implementation Repository stores and manages object implementation information. Object
implementations may be hardware and ORB implementation dependent.

The DII allows a client to dynamically invoke operations on objects so that a client need not necessarily be modified or recompiled as new objects or new operations on existing objects are added to the system.

The SII allows clients to invoke operations on objects using a subroutine call interface and stubs generated by the IDL compiler.

Object adapters are used to invoke object implementations and to generate and interpret object references (unique identifiers for objects). The Basic Object Adapter (BOA) is specified by OMG, but vendors may provide additional object adapters to support different object management mechanisms.

The ORB core is responsible for delivering a request from a client to the appropriate object adapter for the target object.

c) Enterprise JAVA Beans

The Enterprise JavaBeans (EJB) Specification defines a standard model for a Java application server that supports complete portability.
The Enterprise JavaBeans component model logically extends the JavaBeans component model to support server components. Server components are reusable, prepackaged pieces of application functionality designed to run on an application server and can be combined with other components to create customized application systems. Server components are similar to development components but they are generally larger grained and more complete than development components.

The EJB architecture provides an integrated application framework that simplifies the process of developing enterprise-class application systems. An EJB server automatically manages a number of tricky middleware services on behalf of the application components. Because EJB component builders can concentrate on writing business logic rather than complex middleware, the applications get developed more rapidly and the code is of better quality.

The Enterprise JavaBeans architecture is completely independent from any specific platform, protocol, or middleware infrastructure. Applications that are developed for one platform can be redeployed to another platform. EJB applications can scale from a small single-processor, Intel-based Novell environment to a large multiprocessor, SUN UltraSPARC environment to an IBM
mainframe environment without any modification to the applications.

JavaBeans component model is the platform-neutral architecture for the Java application environment. It's advantage lies in developing or assembling network-aware solutions for heterogeneous hardware and operating system environments—within the enterprise or across the Internet.

JavaBeans component architecture extends "Write Once, Run Anywhere" capability to reusable component development. In fact, the JavaBeans architecture takes interoperability a major step forward—Java code runs on every OS and also within any application environment. JavaBeans architecture connects via bridges into other component models such as ActiveX. Software components that use JavaBeans APIs are thus portable to containers including Internet Explorer, Visual Basic, Microsoft Word, Lotus Notes, and others.

The JavaBeans API makes it possible to write component software in the Java programming language. Components are self-contained, reusable software units that can be visually composed into composite components, applets, applications, and servlets using visual application builder tools.
JavaBean components are known as Beans. Components expose their features (for example, public methods and events) to builder tools for visual manipulation. A Bean's features are exposed because feature names adhere to specific design patterns. A "JavaBeans-enabled" builder tool can then examine the Bean's patterns, discern its features, and expose those features for visual manipulation. A builder tool maintains Beans in a palette or toolbox. A bean can be selected from the toolbox, dropped into a form, it's appearance and behavior modified, define its interaction with other Beans, and compose it and other Beans into an applet, application, or new Bean. All this can be done without writing a line of code.

The following list briefly describes key Bean concepts.

- Builder tools discover a Bean's features (that is, its properties, methods, and events) by a process known as introspection. Beans support introspection in two ways:

  - By adhering to specific rules, known as design patterns, when naming Bean features. The java.beans.Introspector class examines Beans for these design patterns to discover Bean features. The Introspector class relies on the core reflection API. The Reflection API trail is an excellent place to learn about reflection.
• By explicity providing property, method, and event information with a related Bean Information class. A Bean information class implements the BeanInfo interface. A BeanInfo class explicitly lists those Bean features that are to be exposed to application builder tools.

• Properties are a Bean’s appearance and behavior characteristics that can be changed at design time. Builder tools introspect on a Bean to discover its properties, and expose those properties for manipulation.

• Beans expose properties so they can be customized at design time. Customization is supported in two ways: By using property editors, or by using more sophisticated Bean customizers.

• Beans use events to communicate with other Beans. A Bean that wants to receive events (a listener Bean) registers its interest with the Bean that fires the event (a source Bean). Builder tools can examine a Bean and determine which events that Bean can fire (send) and which it can handle (receive).

• Persistence enables Beans to save and restore their state. Once you’ve changed a Bean’s properties, you can save the state of the Bean and restore that Bean at a later time, property changes intact. JavaBeans uses Java Object Serialization to support persistence.

• A Bean’s methods are no different than Java methods, and can be called from other Beans or a scripting environment. By default all public methods are exported.

2. Wrappers

Wrappers are software code developed to add, modify, and hide functionality from a COTS, GOTS or legacy software
components to align them with the overall system requirements and architecture. With this technique, an interface is created around an existing piece of software, providing a new view of the software to external systems, objects, or users.

Wrapping can be accomplished at multiple levels: around data, individual modules, software components, subsystems, or entire systems. When access to code is provided, wrappers are much more integrated and tightly coupled so that one cannot even tell it's a wrapper whereas when wrapping software components, subsystems or entire systems, the wrapper functions as an interface. This interface can be crude depending on accessibility of the system or components (i.e. how open the system is and whether public APIs are available.)

For legacy software code, object-oriented technology (OOT) can be used to wrap and encapsulate the legacy software code. The narrow concept of a wrapped object is an object with its methods surrounding the legacy software representing its encapsulation as a single object.

OOT is one of the better practices for software development by virtue of its efficiencies in development and maintenance and its inherent support for reuse. OOT consists of a set of methodologies and tools for developing
and maintaining software systems using software objects composed of encapsulated data and operations as the central paradigm. The legacy software code is accessible only through the object-defined methods (or operations). Any user access to the legacy software would be mediated through some of these methods, whether the user interface is a complex set of objects constituting a graphical user interface (GUI) or simple terminal line command input/output (I/O).

The broader concept of an OO wrapper is an object model consisting of multiple classes and objects. This object model is created as part of the wrapper to provide a natural OO interface to the principal conceptual entities implicit in the original system. The new objects and classes of such a wrapper can interface with the legacy programs and data in different ways. An application programming interface (API) may mediate communication between the wrapper object model and the legacy program. When the legacy software is a database, a database server might provide the functionality of an API, with objects accessing the database through SQL calls to the server.

When multiple COTS/GOTS software components are used in a system, two of the components may provide duplicate functionality. The functionality in both components would
have to be evaluated and it may be decided that one is superior to the other. In this case, the weaker component’s functionality may be hidden with the use of a wrapper. In another instance, a COTS software component may have been selected to fulfill some specific functionality of the overall system but one small feature may also be needed. In this situation, a wrapper can be written to add this feature. This is especially useful when the COTS software component supports a set of application program interfaces (APIs).

3. Glue Code

Glue code, as applied in this architectural framework, is a tool to aid system developers to integrate software components by creating a bridge between two or more software components, which otherwise would not communicate with each other. The term, glue code, was coined with the onset of the use of COTS in software development but can be used with GOTS and legacy software that needs to communicate together in a system. The bridge can be as simple as a data translation between two applications where the two applications are expecting the data in two different formats.

In the situation where the target architecture supports a set of APIs for a given system, glue code provides the
capability for a COTS/GOTS software component to interface with other system components using these APIs. Using the APIs provided by the system helps to ensure the system architecture is adhered to.

Sometimes two COTS components may accept the same data in the same format but there is no means of passing this data from one application to another since they are not designed to communicate with each other. In this case, the glue code provides the communication path between the two applications.

4. XML

To attain interoperability in a distributed, heterogeneous system, consistent interpretation of data between the various applications on the different platforms is key. Sharing different formatted data requires a common representation of data to interpret, send, and receive any data, any format, anywhere.

XML is the new data interchange format approved by the World Wide Web Consortium (W3C) [Ref. 25], an independent organization at http://www.w3.org that develops protocols for interoperability on the web. Platform and vendor independent, XML provides lightweight, flexible, self-describing text in the form of tags that may be used in concert with JAVA, in any system, document, or database.
XML format can handle data, data structures and the description of data (metadata).

XML solves the data interoperability problem by providing self-describing tags along with the data so that the receiving applications can consistently interpret the data correctly. These self-describing tags are detailed in the Document Type Definition (DTD).

Another source of XML's unifying strength is its reliance on a new standard called Unicode, a character-encoding system that supports intermingling of text in all the world's major languages. In HTML, as in most word processors, a document is generally in one particular language, whether that be English or Japanese or Arabic. If your software cannot read the characters of that language, then you cannot use the document. The situation can be even worse: software made for use in Taiwan often cannot read mainland-Chinese texts because of incompatible encodings. But software that reads XML properly can deal with any combination of any of these character sets. Thus, XML enables exchange of information not only between different computer systems but also across national and cultural boundaries.

It lays down ground rules that clear away a layer of programming details so that people with similar interests
can concentrate on the hard part—agreeing on how they want to represent the information they commonly exchange. This is not an easy problem to solve, but it is not a new one, either.

Such agreements will be made, because the proliferation of incompatible computer systems has imposed delays, costs and confusion on nearly every area of human activity. People want to share ideas and do business without all having to use the same computers; activity-specific interchange languages go a long way toward making that possible.
V. NITES CASE STUDY

A. NITES

The Naval Integrated Tactical Environmental System (NITES) is a DoD system, representative of systems in the Automated Information System (AIS) arena, which must interoperate with other Command and Control, Communications and Computers, and Intelligence, Surveillance, and Reconnaissance (C4ISR) systems.

1. NITES Background

NITES is the 4th generation evolutionary upgrade to the original Tactical Environmental Support System (TESS). NITES provides tactical Meteorological and Oceanographic (METOC) support to Navy and Marine Corp forces engaged in worldwide operations, ashore and afloat. Though NITES is able to operate independently, the prime mode of operations is through interoperability with C4ISR systems. Sharing of information between METOC and C4ISR systems is critical to achieving total situational awareness by METOC personnel and tactical operators.

NITES provides a METOC database containing climatological data, in-situ environmental data, regional observations, forecasts and warnings, and numerical METOC forecast models. NITES provides the operator with the
capability to produce METOC assessments and forecasts, and METOC product generation applications to support weather briefings that display METOC information to the tactical decision maker; and interfaces to co-located METOC sensor systems. NITES integrates METOC data and products with combatant sensor data, weapon system and platform parameters, and available intelligence to provide tailored tactical products to C4ISR systems.

The basic NITES missions are:

- to store observed and forecast METOC information relevant to ongoing operations.
- to assess the impacts of present and forecast METOC conditions on operations.
- to provide METOC data to planning and decision support systems.

2. **Existing NITES Architecture**

The NITES architecture (figure 2) decouples the application from the data enabling true modularity where one application can be removed/replaced without requiring a total redesign of the system. This is especially crucial when using COTS products where there is no control of future COTS updates.

COTS products are designed to run standalone thus they have a tendency to maintain their own data and their own
database. In systems using multiple COTS products, the nature of COTS hinders true system integration. Unless a system architecture uses a single database and decouples the application from the data, redundant databases must be maintained, data files are duplicated, and no value-added products are shared between applications.

![Figure 2 - NITES Architecture](image)

This architecture also facilitates the requirement for having a central repository for all METOC data. This central repository enables internal and external users to only go to one place for its data. By having a central repository, this architecture also enables the forecaster to control what data entered and left the system. Because the system receives an immense amount of data from a variety of sources and at times may receive the same data from
different sources with conflicting information, the forecaster is responsible for ensuring the data distributed to external users and/or systems is accurate. Maintaining a single database greatly eases the forecaster’s job.

3. Data vs. Product

In the NITES program, the term, data, is distinguished from the term, product. Data refers to the raw information received. Products refer to finished products the aerographer may create. These are usually image products. For example, the observation and gridded data can be either contoured or plotted on a map background, creating a product.

B. CASE STUDY

The NITES system, in its current implementation, drastically deviates from the envisioned system architecture. This case study uses the NITES system to demonstrate the use of the recommended features of the proposed architectural framework to build systems using COTS/GOTS/legacy components. With the proposed architectural framework, this case study will demonstrate the building of the existing NITES architecture using TOGAF terms. In addition, a portion of NITES will be modified to
demonstrate the feasibility of using the proposed tools in the new architectural framework.

1. **Definition of Existing Environment in Existing Terms**

NITES can be viewed as several subsystems: Communications, METOC Application, Tactical Environmental Data Server (TEDS), RAID Storage, NITES Workstations, and Briefing Display. In the existing system, none of the products created on the NITES Workstations are stored back to the TEDS database. The functional view of the NITES architecture is shown in figure 3.

![Diagram showing the functional view of the existing NITES environment](image)

**Figure 3 - Functional View of Existing NITES Environment**

The Communications Subsystem is the primary entry point for all METOC data coming into NITES.
The RAID Storage Subsystem provides for the storage of METOC data on multiple RAID storage devices. These storage devices are mirrored to provide redundancy.

The TEDS Subsystem is the METOC database for the NITES system.

The METOC application subsystem is the application server consisting of the forecaster applications and tools to manipulate the METOC data.

The NITES Workstations run the client application, providing the ability to view and manipulate the METOC data, and create METOC products for briefing purposes.

The Briefing Display consists of an interface to the ship’s CCTV via an Appian Graphics video card. Through the video card, the brief products are transmitted throughout the ship’s CCTV system for display.

The topology of the existing NITES is shown in figure 4.
2. Lessons Learned From The Existing System

The main lesson learned from the existing NITES system is that the forecaster application needs to be more user friendly, requiring less operator intervention and more automation capability for routine tasks. Also, the implemented system did not adhere to the original design architecture which prevented the data/products from being
optimally shared. These lessons learned will be addressed further in this case study.

3. **Market Survey**

The NITES project was tasked to transition from a monolithic system with organically generated source code on a Unix platform to a COTS/GOTS system on a Windows NT platform. The key functionally for the new system would be a Forecaster tool that displays and manipulates METOC data, and enables the operator to annotate and enhance displays for effective presentations.

A market survey of Forecaster applications was conducted as the first step in determining the system architecture for the new system. The survey was based on the following high-level system requirements:

- The Forecaster Application must run on a PC.
- The Forecaster Application must run on the Windows NT operating system.
- The Forecaster Application must have the capability to receive and process standard METOC data including:
  - Model Grids
  - WMO encoded observations (ship, surface, synoptic, upper air, bathythermograph, salinity profile,)
- Forecasts
- Warnings
- Satellite imagery
The initial phase consisted of a preliminary engineering analysis to identify the qualifying applications for further evaluation. This phase consisted of an evaluation of the high level qualifications. Of the over 20 responses to the request for information submitted on the Commerce Business Daily (CBD), five applications (3 COTS and 2 GOTS) met all the high-level requirements and were selected for further analysis. Due to the proprietary nature of this process, neither the vendor nor the applications are identified here.

4. COTS/GOTS Evaluation and Selection

The five Forecaster applications selected proceeded to the second phase, which is the evaluation phase. The evaluation process consisted of three major categories 1) user evaluation 2) engineering analysis and 3) administrative analysis.

The user evaluation consisted of a vendor demonstration of each of the five qualifying applications. This phase consisted of an in-depth demonstration of each qualified application at the vendor’s facility by a team of ‘user experts’. The expert team consisted of Navy personnel from both East and West Coast ships and shore facilities that have had extensive experience as Navy Forecasters. In this phase, the evaluation consisted of a demonstration of
features and capabilities by the vendor using a set of canned data. The primary objective of these demonstrations was to allow the vendor to demonstrate their application and show the expert team how well their application meets the high-level requirements as well as demonstrate the detailed capabilities of the system. It also allowed the user experts to see the application run for the first time.

The preliminary engineering analysis consisted of a technical survey of the vendor's product. The analysis was performed by having the 5 vendors fill out sections A through C, and G of the COTS Evaluation/Selection form enclosed in appendix A. Based on the vendors' responses, a preliminary rating of each of the five vendors was generated. The vendor responses are proprietary and cannot be shown here. However, a brief summary of the results are shown in table 3.
<table>
<thead>
<tr>
<th>Vendor</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. System Req.</td>
<td>Sat</td>
<td>Sat</td>
<td>Sat</td>
<td>Sat</td>
<td>Sat</td>
</tr>
<tr>
<td>B. Documentation</td>
<td>3.5</td>
<td>3.5</td>
<td>4</td>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>Operator</td>
<td>4-Good</td>
<td>3-Adequate</td>
<td>4-Good</td>
<td>3-Adequate</td>
<td>4-Good</td>
</tr>
<tr>
<td>Installation</td>
<td>3-Adequate</td>
<td>4-Good</td>
<td>4-Good</td>
<td>3-Adequate</td>
<td>3-Good</td>
</tr>
<tr>
<td>Maintenance</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>C. Integration</td>
<td>1.3</td>
<td>2.6</td>
<td>2</td>
<td>1.3</td>
<td>1</td>
</tr>
<tr>
<td>Data Ingest</td>
<td>0-Proprietary ingest</td>
<td>5-Full compatible w/ Navy data</td>
<td>3-Takes all Navy data but requires specific dir structure</td>
<td>2-Does not handle ocean data</td>
<td>1-Very limited data types</td>
</tr>
<tr>
<td>Data/Product Export</td>
<td>4-Good Product export</td>
<td>3-Meets all major export requirements</td>
<td>3-Meets all major export requirements</td>
<td>2-limited due to lack of ocean data</td>
<td>2-limited due to lack of data</td>
</tr>
<tr>
<td>APIs</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G. Supportability</td>
<td>4</td>
<td>3.5</td>
<td>3.5</td>
<td>3</td>
<td>2.5</td>
</tr>
<tr>
<td>Vendor Maturity</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Product Maturity</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Customer Support</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Maintenance Cost</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3 - COTS/GOTS Phase 2 Evaluation Results

The Administrative analysis consisted of programmatic considerations including overall costs of the product, and contract availability. This analysis contributed to the rating of each of the five vendors and did not specifically eliminate any vendor. Preference was given to those vendors
who are on the General Schedule Administration and Small Business Administration. All COTS vendors had an existing contract vehicle so this did not become a problem.

The user evaluation in conjunction with the preliminary engineering analysis resulted in the selection of three applications (2 COTS applications and 1 GOTS application).

In the third phase, the final three applications were evaluated at a Navy lab under near live conditions on Navy hardware with live data feeds and displays. This phase consisted of a detailed engineering analysis on compatibility and interoperability. This phase also conducted another user evaluation in a typical user environment using the following detailed list of requirements.

- The Forecaster Application must have the capability to receive, process and display standard METOC data including:
  - Model Grids
    - Retrieve grid within 30 seconds for display.
    - Contour at standard intervals based on level of data.
    - Display streamlines for wind data.
    - Display wind barbs.
    - Display multiple grid forecasts in looping sequence.
  - WMO encoded observations (ship, surface, synoptic, upper air, bathythermograph, salinity profile,)
    - Retrieve observation within 30 seconds for display on a geographic background.
• Display standard METOC station plots.
• Overlay plots on satellite imagery.
• Overlay plots on grid contours.
• Display Skew-T/Log P plot of upper air data.
• Display BT plot of bathythermograph and salinity profile data.
• Display meteogram (time series) plot of surface observations.
• Display cross-sections.
• Forecasts
  • Display text message for reading and editing.
• Warnings
  • Display text message for reading and editing.
  • Plot area of warning on geographical display.
• Satellite imagery
  • Retrieve and display satellite imagery on a geographic background within 60 seconds.
  • Perform gray scale enhancements on satellite image data.
  • Operator controlled transparency over geographic display.
  • Display satellite loops.
• Briefing Capability
  • All the above products must be capable of being saved and imported into a briefing package for presentation.

In addition the applications were rated on subjective usability issues (quality of display, ease of use, enhanced functionally). A summary of the results of this evaluation are provided in table 4.

The engineering analysis consisted of evaluating the compatibility and interoperability of each of the three
applications by loading the application on the target Navy platform and analyzing its ability to work within the target system's operating environment:

- loads and runs on target system within sizing & timing limitations,
- operates simultaneously with other applications without interference,
- utilizes existing communication channels and data files,
- utilizes and shares system devices (printers, CDs, tape drives, displays),
- uses/converts to standard data formats and data units from available sources,
- exports data in compatible formats and units with other applications and external systems that utilize the data.

The results of all the evaluations were presented to the program manager who reviewed the evaluation. A selection was made using the relative rankings of both the user and engineering analysis. In summary Application 1 had the highest overall technical rating but was eliminated due to its high cost which resulted in the selection of Application 2, the second highest rated application.
<table>
<thead>
<tr>
<th>Criteria</th>
<th>App 1</th>
<th>App 2</th>
<th>App 3</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model Grids</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean time to retrieve &amp; display (single data type)</td>
<td>5s</td>
<td>7s</td>
<td>5s</td>
<td></td>
</tr>
<tr>
<td>Contour Surf Pressure</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour Surf Air Temperature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plot Surf Wind Barb (from U &amp; V)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plot Surf Wind Barb (from speed &amp; direction)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display Surf Wind as streamlines (from U &amp; V)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display Surf Wind as streamlines (from speed &amp; direction)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display Surf Anal (all of above) on a map background</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Loop Surf Anal for 12, 24, 36, 48 hr forecast</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour 1000MB HT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour 1000MB Temperature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plot 1000MB Wind Barb (from U &amp; V)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plot 1000MB Wind Barb (from speed &amp; direction)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display 1000MB Wind as streamlines (from U &amp; V)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display 1000MB Wind as streamlines (from speed &amp; direction)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour 850MB HT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour 850MB Temperature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plot 850MB Wind Barb (from U &amp; V)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plot 850MB Wind Barb (from speed &amp; direction)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display 850MB Wind as streamlines (from U &amp; V)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display 850MB Wind as streamlines (from speed &amp; direction)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour 500MB HT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour 500MB Temperature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plot 500MB Wind Barb (from U &amp; V)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plot 500MB Wind Barb (from speed &amp; direction)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display 500MB Wind as streamlines (from U &amp; V)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display 500MB Wind as streamlines (from speed &amp; direction)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour Total 12 hr Precipitation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour Frontal Analysis</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour Total Cloud Cover</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour Fog</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour Sea Water Temperature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display Wind Wave as streamlines</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display Primary Wave as streamlines</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display Swell Wave as streamlines</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WMO encoded Observations, Ship &amp; Surface</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean time to retrieve &amp; display (station model plot)</td>
<td>2s</td>
<td>3s</td>
<td>App 2 does not do standard station model plot</td>
<td></td>
</tr>
<tr>
<td>Station model plots (six selectable parameters w/wind barbs)</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contour Surf Pressure</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour Surf Air Temperature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Time series of METOC variables at selected points</td>
<td>✓</td>
<td></td>
<td>Apps 2 &amp; 3 do not do time series plots</td>
<td></td>
</tr>
<tr>
<td><strong>WMO encoded Observations, Upper Air</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculate &amp; Display Skew-T/Log P plot</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour 500MB HT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contour 500MB Temperature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>WMO encoded Observations, Bathythermograph &amp; Salinity</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display BT plot of bathythermograph and salinity profile data.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Apps do not handle salinity profile data</td>
</tr>
<tr>
<td>Display BT cross-section</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Calculate &amp; Display Sound Speed Profile</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Apps do not calculate SSP</td>
</tr>
<tr>
<td><strong>Text Message Forecast &amp; Warnings</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean time to retrieve &amp; display (single message)</td>
<td>1s</td>
<td>1s</td>
<td>1s</td>
<td>App3 awkward display</td>
</tr>
<tr>
<td>Display text message</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>App 2 has no message edit capability</td>
</tr>
<tr>
<td>Message edit</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>App 2 has no message edit capability</td>
</tr>
<tr>
<td>Message search</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>App 1 limited</td>
</tr>
<tr>
<td>Message delete</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Message create</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Apps do not create message</td>
</tr>
<tr>
<td>Draw area of warning on geographical display</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Apps 2 &amp; 3 do not draw area of warning</td>
</tr>
<tr>
<td><strong>Satellite</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean time to retrieve &amp; display (single satellite pass)</td>
<td>8s</td>
<td>9s</td>
<td>8s</td>
<td></td>
</tr>
<tr>
<td>Display DMSP vis</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display DMSP IR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display NOAA Ch 1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display NOAA Ch 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display NOAA Ch 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display NOAA Ch 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display NOAA Ch 5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Satellite gray scale enhancements</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display GOES WEFAX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Operator controlled transparency over geographic display</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Display satellite loops</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Apps 1 &amp; 3 have automatic update of loops</td>
</tr>
<tr>
<td><strong>Annotations &amp; Enhancements</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>App 1 had superior display capabilities</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add Text</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Draw lines, curves &amp; areas</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Draw standard weather boundaries</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Add standard weather symbols</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Modify text, lines, &amp; symbol size, color, and orientation</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Display up to 5 layers of variables on a plan view map including satellite</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Display 3-D contours of METOC variables</td>
<td>✔</td>
<td>Apps 2 &amp; 3 do not do 3-D contour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automatic product generation</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operator actions required to setup, operate, and perform tasks are reasonable and intuitive with useful displays and procedures</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>App 1 easiest to run due to automation. Apps 2 and 3 require more operator interaction.</td>
</tr>
<tr>
<td>Quality of displays</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>App 1 had superior display capability</td>
</tr>
<tr>
<td>Other features &amp; enhancements</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>App 1 had best set of features and enhancements</td>
</tr>
</tbody>
</table>

Table 4 - COTS/GOTS Phase 3 Evaluation Results

5. Restatement of Existing Environment in TOGAF Terms

The existing NITES environment follows a distributed computing architectural model. A client NITES application on a workstation communicates with the server application on the METOC Applications Server. The Communications application parses data as it comes in and passes it to the decoder application on the METOC Application Server. The decoders decode the data, transfer it to the TEDS Server where it is entered into the RAIDS for permanent storage.

The general diagram for the NITES distributed computing model is shown in the following figure:

105
Figure 5 - NITES Distributed Computing Architecture

The existing environment can be restated in TOGAF terms using the following table that maps the existing NITES components into the standard application platform services:

<table>
<thead>
<tr>
<th></th>
<th>Communications</th>
<th>Briefing</th>
<th>TEDS</th>
<th>Application</th>
<th>RAID Storage</th>
<th>Workstation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Interchange</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Management</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphics and Imaging</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Operating System</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Software Engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>User Interface</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 5 - Mapping of Services to Existing Architecture
The standard application platform services provided in NITES perform the following functions:

- **Data Interchange** - Data entering the system and leaving the system are in one of two formats, either WMO or XML. Products created on the system are in standard graphics formats, i.e. BMP, JPEG, TIFF. All data and products on the NITES system can be sent to common data output devices like printers and CRT screens.

- **Data Management** - These services are satisfied by the INFORMIX RDBMS, which resides a layer below TEDS.

- **Graphics and Imaging** - Scanners and compression software services are used by the NITES system. Drawing services are used by each of the NITES workstations.

- **Network** - All of the components in NITES use the functions provided by the TCP/IP protocol stack working over an Ethernet network.

- **Operating System** - The Operating System Services are found on all of the NITES subsystems.

- **Software Engineering** - Although programming language compilers and GUI builders are used to develop NITES, no Software Engineering Services map into existing NITES subsystems.

- **Transaction Processing** - The Informix RDBMS used by TEDS provides these services.

- **User Interface** - These functions are provided at the Communications Subsystem, TEDS Subsystem, and at the NITES workstations.
6. Views, Constraints and External Environments

a) Operations View

In the Operations view, the key operational aspect of the system is to store observed and forecast METOC information relevant to ongoing operations, and to create and manipulate the data and products to assess the impacts of present and forecast METOC conditions on operations. Products consist of horizontal weather depictions (HWDs), satellite image briefs, and outputs from tactical decision aids.

b) Management View

The management view of the system is dependent upon the role a user plays in the system. This view partitions the users of the system into the following profiles:

- Database Administrator - controls access to the METOC Database.
- System Administrator - controls the operation of the system (installation, system shutdown) and assigns privileges to METOC users.
- METOC Users - creates, views, manipulates, and prints data and products on the system.
c) Security

The security view (figure 6) for NITES has one type of security mechanism - access control. Each user has a unique user ID, password and group that allows him access to any of the Graphics Display Workstations and all output devices (i.e. printers and scanners). The group determines level of access, database, system administrator, or METOC user.

![Diagram of NITES Security View]

Figure 6 - NITES Security View

d) Constraints

One of the major constraints of the system is the database application suite. This database was developed prior to NITES and independent of the system but, because it represents a significant investment, it must be retained.

The system is also constrained due to a decision made during the development process. In the fielded NITES configuration, the implementation fails to adhere to the system architecture. During the development, a decision was made to forfeit the architecture to meet the system
schedule. The COTS forecaster applications do not communicate with the database. Instead, data is distributed straight to the applications directly from the communications feeds. Value added products created by the forecasters using the COTS applications are manually pushed to the briefing utility to be incorporated into existing briefs. All this is handled independent of the database, violating the original system architecture where all communications are supposed to go through the database and preventing a sharing of value added data and products between applications.

External systems communicate with the NITES using the published database APIs to extract the data it needs for its application. Without the capability for the forecaster to store his value added data into the database, the external systems extracting data from the database is not using the data the forecaster intends for them to use since that data is not in the database.

e) External Environments

The network used at each NITES site is part of the local environment and, as such, the existing NITES coexists with other networked systems. The new target architecture will also be required to fit into existing external
environments without disruption of the site's other missions.

7. **Target Architecture**

The target architecture for the new NITES software follows the goals and requirements outlined in the Software Requirements Specification outlined in Appendix B. The NITES program has an existing performance specification used to build the 4th generation system, which we also used for this case study. The SRS written for this case study lists requirements above and beyond those in the NITES Performance Specification to satisfy the architectural requirements.

The NITES currently consists of COTS, GOTS, newly developed source code, and legacy software residing on NT and UNIX platforms, all of which must interact together in a seamless fashion to enable the user (aerographer) to analyze, create and brief METOC products.

This case study prototypes a portion of the NITES system to redesign it to adhere to the original system architecture. The main goal was to bring the system back to conform to the original system architecture consisting of a central database residing on a UNIX computer, which is shared amongst the various NITES components as depicted in figure 7. In this topology, there is no direct interaction...
between the components. All interactions are through the central database. This topology allows ease of integration of COTS components as it minimizes the integration effort as each component only has one interconnection.

Figure 7 - NITES Functional Architecture

The Rational Rose tool is used to document the design. The Universal Modeling Language (UML) is used to describe the design as shown in the Software Design Specification in Appendix C.

In order for the COTS, GOTS and legacy components to interact seamlessly, wrappers and glue code are implemented. Wrappers are used to add/hide functionality of the COTS, GOTS and legacy software where access to source code is not available. Specifically for this prototype, a wrapper is
written to add functionality to the COTS briefing application to automatically update a brief as new data comes into the system and is stored in the database. [Refs. 26, 27]

The NITES database supports a set of public APIs. The glue code written for this prototype connects to the NITES database, retrieves data from the database using the APIs and, feeds the products and enhanced data to the database APIs to store back to the database.

Slides for the briefing package are generated by the operator using an external COTS/GOTS application. As each of these slides is generated, it is saved to a directory by the COTS/GOTS application. The system monitor polls the directory and when a file is found, notifies the controller.

When the controller receives notification from the monitor that a new file exists, the controller will create an instance of the glue component, which will connect to the database and store the file.

Once the products are stored in the database, the briefing utility (Microsoft Powerpoint) needs to extract this data to incorporate into a brief. Using the APIs provided with the Microsoft products, a wrapper was written to interface the Microsoft APIs with the database APIs to extract the data from the database and automatically
populate the brief. The full source code is provided in Appendix D.

8. Constraints of Case Study

Limitations of this case study include budget and schedule. The NITES project is evolutionary and the first evolution took many engineers and developers over 2.5 years to build the system. The team in this case study consists of 4 people and we had a 6 month schedule. Cost for this is out of pocket so we borrowed almost everything we needed.

In order to get the best performance out of the ORB, an evaluation process would have needed to be conducted to select the ORB best suited for our purposes. Because we are constrained by schedule and costs, we are not able to obtain the best ORBs, etc. so in certain areas, the performance is sacrificed to demonstrate the utility.
VI. CONCLUSION AND RECOMMENDATIONS

A. CONCLUSION

An architectural framework is critical in the development of a target architecture. With DoD's mandate to incorporate COTS in system architectures, existing architectural frameworks do not sufficiently address the development of target architectures using COTS.

This thesis proposes an architectural framework, which addresses heterogeneous, interoperable systems built upon COTS/GOTS/Legacy components. The proposed architectural framework incorporates a methodology for integrating COTS components into a system architecture as well as provide the tools necessary to build interoperable systems using COTS/GOTS/Legacy components. These tools include ORBs, wrapper and glue code technology, and XML.

The methodology provides a structured approach for integrating COTS components into a system architecture. It includes a market survey, the evaluation and selection of COTS components, and identifies interface and interoperability characteristics and deficiencies. This methodology enables the system developer to successfully design a target architecture and head off problems early in the system development using COTS/GOTS/Legacy components.

115
The tools in the proposed architectural framework enables a system developer to integrate standalone COTS applications into a cohesive, integrated system where software components interoperate and share data.

B. RECOMMENDATIONS

Conduct a study of the existing CORBA ORBs to evaluate the features and what percentage supports COM/DCOM and EJB. Publish this information so that users can more easily select between the different ORBs on the market.

The wrapper and glue code technologies have been widely discussed but need to be formalized into a specification that can be used as guidance for integrating COTS/GOTS/legacy components.

The separation of data and application in an architecture is very important in creating an integrated, heterorogeneous system using COTS/GOTS/legacy components. This was lightly touched upon in this thesis but could be expanded upon. Separating the data from the application introduces interoperability issues. Data and its meaning and interpretation between various COTS, GOTS, and legacy applications is an important factor to interoperability and a solution should be further explored.
APPENDIX A. SAMPLE COTS/GOTS SELECTION/EVALUATION FORM
COTS/GOTS Selection/Evaluation Form

<table>
<thead>
<tr>
<th>Name of COTS/GOTS Package</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Target system</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Cost</td>
<td>Multiple License cost</td>
</tr>
</tbody>
</table>

A. System Requirements

<table>
<thead>
<tr>
<th>Operating System(s)</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>Disk Storage</td>
</tr>
<tr>
<td>I/O Device(s)</td>
<td>Video Resolution</td>
</tr>
</tbody>
</table>

B. Documentation

<table>
<thead>
<tr>
<th>Document Title</th>
<th>Type</th>
<th>Rating</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Type – Indicate document type (Operator, installation, maintenance, etc.)
Rating – Scale from 1 to 5 where 1=poor 3=adequate 5=excellent
Cost – Enter Unit cost per license
### C. Integration

#### Data Requirements of COTS Application

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Format</th>
<th>units</th>
<th>Exchange Method¹</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Data/Product Export

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Format</th>
<th>units</th>
<th>Exchange Method¹</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Supported Application Program Interfaces (API)

<table>
<thead>
<tr>
<th>API</th>
<th>Functional Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. API, Directory/File, Proprietary database call, Communication channel
D. Interoperability

Evaluate the COTS application's ability to exchange/share data within the target system and with external systems, including required operator support and control.
## E. Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Rating&lt;sup&gt;1&lt;/sup&gt;</th>
<th>Evaluation Method&lt;sup&gt;2&lt;/sup&gt;</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use additional sheets as required

1. 0 to 5 where  
   0 = not satisfied  
   1-2 = poorly satisfied (below threshold for quantitative measurement)  
   3 = Adequate (meets threshold for quantitative measurement)  
   4-5 = Superior (exceeds threshold for quantitative measurement)  
   Enter value for quantitative measurement in comment field

2. Independent = independently tested on target system  
   Lab = independently tested on non-target system  
   Demo = observed on demonstration system  
   Vendor = vendor verified  
   Documentation = stated in vendor documentation
## F. Usability
Evaluate operator actions required to setup, operate, and utilize COTS data/products. Are procedures reasonable and intuitive with useful displays?

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## G. Supportability

**Vendor Maturity** *(Years in field, Size of Company, Market dominance, Competitors):*

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Product Maturity** *(How long has the COTS/GOTS package been in the market? What is the market share?):*

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Customer Support** *(Will the vendor support trouble calls? How responsive is vendor in providing resolution?):*

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Update/Maintenance Cost** *(update cycle & cost per platform):*

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX B. SOFTWARE REQUIREMENTS SPECIFICATION (SRS) FOR AN ARCHITECTURAL FRAMEWORK OF COTS/GOTS/LEGACY SYSTEM
Software Requirements Specification
For An
Architectural Framework
Of
DoD COTS/GOTS/Legacy System
1. **SCOPE**

1.1 **Introduction**

The trend towards using Commercial Off-The-Shelf (COTS) software within Department of Defense (DoD) has become the accepted way to build systems. Twenty years ago, almost all DoD software-intensive systems were built by awarding large multimillion-dollar contracts to defense contractors to build these systems from scratch. In the 90’s, with a constantly dwindling budget, the focus has shifted to building software-intensive systems by integrating COTS software components.

Building software systems from COTS components is quite different. The black box nature of the COTS software components along with the uncontrollable evolution process requires a different architectural approach in developing systems with COTS.

1.2 **Purpose**

The purpose of this requirements specification is to analyze and document the requirements in developing an architectural framework for COTS/GOTS/Legacy systems within the DoD. To focus the requirements of the architectural framework, a DoD Meteorological and Oceanographic (METOC) system, the Naval Integrated Tactical Environmental System I (NITES I), which is very representative of today’s DoD COTS/GOTS/Legacy systems, will be used.

1.3 **Background**

The NITES I project is a Space and Naval Warfare (SPAWAR) sponsored project within DoD. Like most other projects within DoD, the NITES I project is being developed in an environment that emphasizes the use of personal computers and COTS components.

NITES I acquires and assimilates various METOC data for use by US Navy and Marine Corps forecasters. The purpose of NITES I is to provide the METOC community (Users) with the tools necessary to support the warfighter (Customers).

The NITES I is the primary METOC data fusion platform and principal METOC analysis workstation, intended to be operated on both a classified and unclassified network environment by METOC personnel. This system receives, processes, stores and disseminates METOC data and provides analysis tools to render products for application to military and tactical operations. NITES I data and information/products are stored in a unified METOC database on the C4ISR network and available to local and remote planners and warfighters.

1.4 **References**

Performance Specification (PS) for the Tactical Environmental Support System / Next Century TESS(NC) (AN/UMK-3) (NITES version I and II)


2 GENERAL DESCRIPTION

2.1 Architecture Goals

Integration
COTS/GOTS/legacy components are usually created as standalone products. When these components are targeted for integration into a system, the architecture shall provide seamless integration of these COTS/GOTS/legacy components.

The architecture shall support middleware approaches to bind data, information and COTS/GOTS/legacy components.

Because evolution and upgrade of COTS/GOTS components are outside the control of the system integrators, the architecture of the COTS/GOTS/legacy system shall have an adaptable component configuration to reduce the effort of testing and reintegration when upgrades or new COTS/GOTS packages are introduced to the system.

Interoperability
COTS/GOTS and legacy systems reside on multiple platforms. This architecture shall address distributed, heterogeneous systems consisting of both UNIX and PC-based platforms.

In order to achieve and maintain information superiority on the battlefield, the architectural framework for DoD COTS/GOTS/legacy systems shall have the capability to share, receive and transmit on heterogeneous networks and hardware devices.

The exchange of data between two systems shall be in such a way that interpretation of the data is precisely the same. The data displayed on two different systems shall remain consistent. The architectural framework shall include standard application program interfaces (APIs). APIs specify a complete interface between the application software and the platform across which all services are provided. A rigorous definition of the interface results in application portability provided the platform supports the API as specified, and the application uses the specified API. The API definitions shall include the syntax and semantics of the programmatic interface as well as the necessary protocol and data structure definitions.

Adopted Framework Technology
Java/C++, web technologies, open systems, application program interfaces, common operating environment, object and component technology, commercial products and standards are all important to the COTS/GOTS/legacy system architecture.

The COTS/GOTS/legacy system shall adopt the Interface Definition Language (IDL) as the language for expressing the syntax of the framework services.
The COTS/GOTS/legacy system architecture shall be expressed as UML class and package diagrams, with detailed component descriptions using IDL with English narrative to provide semantics.

Security

DoD tactical systems are normally classified to some security level. In building this architectural framework, the architecture shall address the DoD Trusted Computer System Evaluation Criteria (TCSEC) to at least the C2 security level.

The architecture shall include discretionary access control (DAC).

Only single level classification systems shall be supported in this architecture (i.e. no multi-level security (MLS)).

Assembled components shall not require modification to add security services.

The security mechanisms shall be protected from unauthorized access.

The following security services shall be available to the component assembler:

1. Single login for users
   The single login for users means the user needs to identify himself once per session. It is the responsibility of the security services to protect and distribute the authentication information of a user.

2. Mutual authentication
   Mutual authentication ensures proper identification of the user to the system and the system to the user.

3. Auditing
   Auditing means significant security events are recorded for later analysis. Significant security events shall include logon and logoff, security policy changes, user and group management, and access to specified objects.

4. Secure key distribution
   Key distribution provides a secure transport mechanism for encryption keys.

5. Role based Access Control
   Role based access control assigns roles to users and privileges to roles, thereby simplifying access control if the number of roles is less than the number of users.

6. Data confidentiality
   Data confidentiality means data is disclosed according to a policy.

7. Data integrity
   Data integrity means the recipient gets the intended data.

8. Non-repudiation and authenticity
Non-repudiation means the sender of a message can not later deny he sent the message.

**Network Security**

The trend in DoD is for networked systems vice standalone monolithic systems and because most systems have some level of classification, this architecture shall address network security.

The architectural framework shall support a secure network.

The architectural framework shall support the network security mechanisms specific to the target architecture, including firewalls, routers, encryption, and proxy services.

**Network Communications**

The architectural framework shall support different network protocols (i.e. TCP/IP) and topologies dependent on the target architecture.

The application layer shall be able to execute a variety of data management commands without having knowledge of the data location, database, file type, operating system, network protocol, or platform location.

**Development Language**

The architectural framework shall support any development language that is supported by the legacy system as well as any development language that supports platform independence for newly developed code in the target architecture.

### 2.2 Assumptions and Dependencies

**Assumption 1:** Legacy systems are monolithic and not modifiable.

**Assumption 2:** Legacy systems have some existing mechanism for interaction.

**Assumption 3:** There are varying degrees of COTS. To be considered COTS, the component cannot be modified.

**Assumption 4:** Reliability, performance, safety and security must be weighed in the target architecture.

**Assumption 5:** Multilevel security systems are beyond the scope of this effort.

### 3. TARGET ARCHITECTURE FUNCTIONS

**Database**

COTS software applications which handle data tend to have their own mechanism and structure for the storage of the data internal to the COTS application. When the target
architecture includes a master database to store its data, the architectural framework shall support the target architecture's central storage of data. The architecture shall support remote access to the database.

Security
The target architecture shall support Discretionary Access Control (DAC).

Access to information controlled by an application shall be based on an access control list (ACL) of a parameter that can be used to distinguish between authorized and non-authorized entities. Entities include users, devices, and other applications.

The target architecture shall support non-repudiation.
   a. The data recipient shall be assured of the originator's identity.

   b. The data originator shall be provided with proof of delivery.

   c. The algorithm used to digitally sign data entries and receipts shall be either the Digital Signature Standard (DSS) FIPS 186 or RSA (1024 bit).

   d. The original transmitted data signed by the sender and the requested receipt signed by the recipient shall be time-stamped by a trusted third party.

Graphical User Interface (GUI)
The target architecture shall include a GUI style guide. If a GUI style guide does not exist for the target architecture, UNIX platforms shall adhere to the MOTIF standard and X-Windows standard, and PC platforms shall adhere to the Windows NT standard.

External System Interfaces
Because the target architecture exists in a network environment where it shares data with other external systems, the external system interfaces where information is exchanged shall be well defined to support interoperability.

Middleware Technology
The COTS/GOTS/legacy architecture shall support new component integration technologies (i.e. COM/DCOM) to broker between components that by themselves normally do not communicate to form an integrated system.

The target architecture shall support wrappers to enable COTS/GOTS applications to interface with each other.

The wrappers shall support the METOC data (listed in Table 6 of reference 1) and its various formats within NITES. The architecture shall ensure when an application updates a set of data, the update is consistently made throughout the rest of the database.
4. ARCHITECTURE ATTRIBUTES

4.1 Performance Requirements

The performance requirements for the target system are contained in Table 6B of the NITES Performance Specification. In addition to those performance requirements, the following requirements shall also be addressed in the target architecture.

The architecture shall optimize the database access over a network.

The architecture shall allow concurrent access of the database to multiple users.

The component technology shall not degrade the system performance by more than 10% of the target system's current performance requirements. Refer to Table 6B of the NITES Performance Specification.

4.2 Reliability Requirements

The target architecture shall use standard fault-tolerant technologies (i.e. Replication to maintain the reliability and availability requirements of DoD systems.)

While the data traverses throughout various applications, to different platforms, through the network and to/from database, it must remain consistent and not suffer any degradation.

4.3 Design Constraints

Because many existing legacy systems reside on UNIX platforms and the DoD has made a commitment to move towards a PC architecture, the architectural framework shall support both UNIX and PC platforms with the goal of moving towards a pure PC architecture. It is not required that all COTS/GOTS/legacy system components be executable on both platforms but the data must be able to be shared by components on different platforms.

Newly developed DoD systems must use COTS products to the greatest extent possible.

As most COTS/GOTS applications are designed to be standalone, these applications will usually have their own way of retrieving and storing data. When these applications are integrated into a system, the internals of the application of how it retrieves and stores data will not be modified.

There are varying degrees of COTS products. Depending on whether the COTS product is an opaque or a black box will drive the wrapper design and implementation.
APPENDIX C. SOFTWARE DESIGN SPECIFICATION (SDS) FOR AN ARCHITECTURAL FRAMEWORK OF DOD COTS/GOTS/LEGACY SYSTEM
Software Design Specification

For An

Architectural Framework

Of

DoD COTS/GOTS/Legacy System
# TABLE OF CONTENTS

1. SCOPE .................................................................................................................. 134
   1.1 Identification .................................................................................................. 134
2. APPLICABLE DOCUMENTS .................................................................................. 134
3. SYSTEM ARCHITECTURE ................................................................................... 134
   3.1 System Architecture Diagram ....................................................................... 134
   3.2 Inter-task Communication ............................................................................. 139
      Monitor/Controller ............................................................................................. 139
      Controller/Glue Component .............................................................................. 139
      CBWrapper/Controller ..................................................................................... 139
      CBWrapper/Glue Component ............................................................................ 139
   3.3 Subsystem Description .................................................................................... 139
      Monitor ............................................................................................................... 139
      Controller .......................................................................................................... 139
      Glue Component ............................................................................................... 141
      CBWrapper ....................................................................................................... 145
      Initialization GUI ............................................................................................. 145
      Configuration GUI ........................................................................................... 145
      Naming Convention ......................................................................................... 145
      Thin Client Technology .................................................................................... 146
      Push Technology ............................................................................................... 146
      OMF ................................................................................................................. 146
1. SCOPE

1.1 Identification

The system defined by this document is the Naval Integrated Tactical Environmental System (NITES). This document establishes the system design for the NITES. The requirements are stated in the Performance Specification (PS) for the Tactical Environmental Support System / Next Century TESS(3)/NC (AN/UMK-3) (NITES version I and II) and the Software Requirements Specifications for An Architectural Framework of DoD COTS/GOTS/Legacy System. The requirements for the architectural framework as well as the target architecture are documented in the SRS. The SRS lists requirements above and beyond those in the NITES Performance Specification to satisfy the architectural requirements.

2. APPLICABLE DOCUMENTS

Performance Specification (PS) for the Tactical Environmental Support System / Next Century TESS(3)/NC (AN/UMK-3) (NITES version I and II)

Software Requirements Specifications for An Architectural Framework of DoD COTS/GOTS/Legacy System

3. SYSTEM ARCHITECTURE

3.1 System Architecture Diagram

The Naval Integrated Tactical Environmental System (NITES) software is designed to run in a distributed, heterogeneous environment on standard commercial-off-the-shelf (COTS) personal computers (PCs) and TAC-4 UNIX computers.

The NITES architecture consists of a central database residing on a UNIX computer, which is shared amongst the various NITES components (most of which reside on PCs with the exception of the tactical applications which reside on a TAC-4 UNIX computer) as depicted in figure 1. In this topology, there is no direct interaction between the components. All interactions are through the central database. This topology allows ease of integration of COTS components as it minimizes the integration effort as each component only has one interconnection.
Figure 1 – NITES Architecture Diagram

Forecaster applications (COTS/GOTS) - Manipulate METOC data to easily plot, analyze, display on a common geographical reference.

Serial Communications (Legacy code) - Handles the ingest and dissemination of METOC data through existing legacy communication channels.

Briefing (COTS) - Briefing utility used to brief tactical commanders, flight operators the environmental conditions that they will be operating in.

Tactical applications (Legacy code and newly developed code) - Tactical applications take in METOC data to predict the affects of the environmental conditions on the environment, tactical equipment, etc.

Database (GOTS) - The database is the central repository for all METOC data.

Network communications (GOTS) - Handles the ingest and dissemination of METOC data through SİPRNET.

The deployment diagram, as depicted in figure 2, consists of a NITES Server, a NITES Database Server, and NITES workstations with a communications package, an applications package, a database package, a system controller package, a security package and a briefer package residing on multiple hardware platforms.
In the NITES architecture, all interactions are through the NITES database. However, in the initial delivery of the NITES software, this architecture was violated since none of the COTS applications were able to communicate with the NITES database to retrieve and/or store data and products.

A prototype of a portion of the NITES system will be developed to demonstrate the NITES architecture using the architectural design pattern as depicted in figure 3. A system controller package, wrapper and the glue packages are newly developed for the NITES. The COTS applications packages and the briefer package will be modified to use wrapper and glue technology to enable it to communicate with the database.
package. These packages will be designed and developed to move the system in the direction of conforming to the existing NITES architecture.
Figure 3 - Architectural Design Pattern
3.2 Inter-task Communication

The tasks on the NITES will be implemented to run asynchronously. Communications are broken down between the following tasks:

- Monitor/Controller
- Controller/Glue Component
- CBWrapper/Glue Component
- CBWrapper/Controller

The Application Wrapper is responsible for making the object available to a COTS viewer application.

Monitor/Controller

Slides for the briefing package are generated by the operator using an external COTS/GOTS application. As each of these slides is generated, it is saved to a directory by the COTS/GOTS application. The system monitor polls the directory and when a file is found, notifies the controller.

Controller/Glue Component

When the controller receives notification from the monitor that a new file exists, the controller will create an instance of the glue component.

CBWrapper/Controller

CBWrapper registers interest in new products with the controller. When the controller is notified by the glue component that a file is successfully stored in the database, it will broadcast the information to all the wrappers running on client workstations. It is the responsibility of the CBWrapper to ignore image types not appropriate for the current brief. This assumes there is at least one wrapper running.

CBWrapper/Glue Component

The CBWrapper requests an image product from the glue code, which will use the existing database APIs to connect to the database, retrieves the product and returns it to the CBWrapper. The request mechanism is used to initialize and update the brief.

3.3 Subsystem Description

The object diagram and sequence diagram depicts objects required to design the update of a briefing package and the scenarios of initializing and updating a briefing package, and storing data to the database are shown in figures 4, 5, 6 and 7 respectively.

Monitor

The Monitor component is responsible for detecting the presence of a new object.

Controller

The controller component is handled by the Distributed Component Object Model (DCOM) and is responsible for coordinating multiple concurrent asynchronous activities. The controller runs on the
application server. It serves two functions within the system, handling notifications from the monitor and the glue component.
Figure 4 – Wrapper & Glue Code Object Diagram
Figure 5 – Continuous Brief Initialization Sequence Diagram
Figure 6 – Continuous Brief Update Sequence Diagram
Figure 7 - Store Object Into Database Sequence Diagram
Glue Component

The glue component is responsible for connecting, storing and retrieving objects from an ODBC compliant relational database.

CBWrapper

Wrappers are software code developed to add, modify, and hide functionality from COTS, GOTS or legacy software components to align them with the overall system requirements and architecture. In the design, wrapper and glue code technology is being implemented to enable the COTS applications to adhere to the existing NITES architecture.

The briefing package consists of Microsoft PowerPoint, a COTS application package. The PowerPoint application contains APIs, which can be used by CBWrapper to create the added functionality of automatically creating and updating the briefing package in the background.

The PPT APIs used for the wrapper interface include:
- Presentations.Add
- Slides.Add
- SlideShowTransition
- SlideShowSetting
- Shapes.AddPicture
- Shapes.PictureFormat

Initialization GUI

The Initialization GUI is used to initialize each component with the number of images, starting from the most current; the image type; the display duration of each image in seconds; and the height and width of the display area. Default values are 24 images, 0 second duration, and display area equal to the workstation’s screen size.

Configuration GUI

The Configuration GUI defines the set of image types available for the brief. Associated with each image type is the working directory containing the current set of brief images and a web server virtual directory corresponding to the working directory. The CBWrapper uses the configuration file to initialize the image type options available to the briefer. The monitor uses the configuration file to build a list of directories to poll.

The Configuration GUI is not restricted to the image types settings. It can be used for defining various sets of key values. For instance, we can use this Configuration GUI to define the key set values for network configuration, or application’s initial default settings. This provides the extensibility for future development of applications.

Naming Convention

The filename associated with each image type consists of the fields representing the created date and time, the file format (i.e., gif, jpeg, etc.), and other information for a particular image (i.e., the channel, the location, etc.)

The filename begins with the date and time, followed by other information. For instance, a file named "20000523.1331.gms5.IR.MODEL_OVERLAY.500HT.NOGAPS" indicates that the file was created on
May 23, 2000, at 13:31. The CBWrapper uses the date and time embedded in the filename for updating the continuous brief. The other information of the filename is used by the Glue component for storing and retrieving images to and from the database.

**Thin Client Technology**

CBWrapper is implemented using modern thin client technology. When a user opens an HTTP page from a browser, the CBWrapper is then automatically downloaded and installed on the client machine. Once the CBWrapper is up and running, all images needed for creating the brief are dynamically downloaded from the server using the OpenURL method. OpenURL uses the current open HTTP connection to transfer image files. The continuous brief is created on the client machine using the PowerPoint APIs. PowerPoint is used to display the brief.

**Push Technology**

The advantage of using this technique is that the client needs not poll the server periodically for new data. The server notifies its clients (CBWrapper) when new data (images) arrives. The CBWrapper receives the notification and compares the image type with the type being shown. If the image types match, the CBWrapper downloads a new set of images from the server and updates the brief.

**OMF**

Sharing different formatted data requires a common representation of data to interpret, send, and receive any data, any format, anywhere. Within NITES, meteorological and oceanographic observations, and certain types of bulletins (SIGMETS, JOTS warnings, and Tropical Cyclone Warnings, for example) are received and transmitted in an Extensible Markup Language (XML)-based format called Weather Observation Markup Format (OMF). OMF preserves the original text of each observation or bulletin, and also includes information decoded from the observation/bulletin and other metadata concerning the message.

OMF solves the data interoperability problem by providing self-describing tags along with the data so that the receiving applications can consistently interpret the data correctly. These self-describing tags are detailed in the Document Type Definition (DTD). When drafting the NITES data into OMF, three things must be agreed on: which tags will be allowed, how tagged elements may nest within one another and how they should be processed. The first two, the language’s vocabulary and structure, are codified in the DTD.

OMF is an application of XML, and by its virtue, an application of SGML. SGML is used extensively within DoD for documenting of various types of information (military standards, procurement materials, service manuals). OMF brings weather observations into the same fold. Thus, the design goals of OMF are:

- Mark up (annotate) raw observation reports with additional description and derived, computed quantities.
- The raw report data must not be modified in any way, and should be conveniently extractable (by simply stripping all the tags away).
- OMF must be concise. While providing useful annotations to a client, OMF markup should not impose undue overhead on communication channels.
- It should be possible to extend the markup with additional annotations, without affecting applications that do not use this information.

The OMF contains the following elements:
• Reports - defines a group of weather observation reports

• METAR for a single METAR report

• SPECI for a single SPECI report

• UAR for a combined Rawinsonde and Pibal Observation report

• BTSC for ocean profile data (temperature, salinity, current)

• SYN for a surface synoptic report from a land or sea station

• Advisories - defines a collection of weather hazard warnings

• SIGMET - SIGNificant METeorological Information

• Forecasts - defines a set of weather forecasts

• TAF - Terminal Aerodrome Forecasts

• Messages - defines a set of plain-text bulletins.

The following sections define the major elements along with the minor elements that are relevant to them. In each section, XML DTD declarations are provided for precise definition of elements and attributes. The collection of XML DTD declarations found in this specification can be arbitrarily extended to add new elements and attributes for new enhancements. Some of the element attributes are common. For compactness, they are defined in the following table.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Brief Description</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
</table>
| TStamp    | Time Stamp        | unsigned integer | UTC time in seconds since the Epoch, 00:00:00 Jan 1, 1970 UTC. This is the value returned by a POSIX function `time()`.
Example: `Tstamp='937507702'` |
| TRange    | Time Interval     | a string of form "aaa, bbb", where aaa and bbb are unsigned integer numbers specifying the beginning and the end timestamps of the interval. | Timestamps are in seconds since the Epoch, 00:00:00 Jan 1, 1970 UTC. These are the values returned by a POSIX function `time()`.
Example: `Trange='937832400, 937915200'` |
| LatLon    | Specification of a point on the globe | A string of a form "aaa.bbb, ccc.ddd", where aaa.bbb and ccc.ddd are signed floating point numbers | The latitude and longitude, respectively, of a point on the globe, in whole and fractional degrees. The numbers are positive for Northern latitudes and Eastern longitudes, and negative for Southern latitudes and Western longitudes.
The range of the numbers is [-90.0, 90.0] for latitudes, (-180.0, 180.0] for longitudes.
Example: `LatLon='32.433, -99.850'` |
| LatLons   | Specification of a sequence of points on the globe | a string of a form "lat1, lon1, lat2, lon2, latn, lonn" where each pair (lat1, lon1, etc.) are signed floating point numbers | A sequence of pairs of numbers, each pair giving the latitude and longitude of a single point in the sequence, in whole and fractional degrees.
See the LatLon attribute above for more details.
Example: `LatLons='38.420, -111.125, 36.286, -111.492, 36.307, -112.630, 37.700, -113.223, 38.420, -111.125'` |
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Brief Description</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBox</td>
<td>Bounding box, which tells the latitudinal and the longitudinal spans of an area of the globe</td>
<td>A string of a form &quot;lat-N, lon-W, lat-S, lon-E&quot;, where the lats and lons are signed floating-point numbers, in degrees</td>
<td>Specification of the bounding box for an area of interest. Here lat-N is the latitude of the Northern-most point of the area, lat-S is the latitude of the Southern-most point, lon-W is the longitude of the Western-most point of the area, and lon-E is the Eastern-most longitude. It is required that lat-N &gt;= lat-S. The left-lon (lon-W) may however be greater than the right-lon (lon-E). For example, a range of longitudes [-170,170] specifies the entire world but Indonesia. On the other end, the range [170, -170] includes Indonesia only. By the same token, [-10,10] pertains to a 21-degree longitude strip along the Greenwich meridian, while [10,-10] specifies the whole globe except for that strip. Example: Bbox='60.0, -120.0, 20.0, -100.0'</td>
</tr>
<tr>
<td>Bld</td>
<td>Station identification group</td>
<td>Unsigned integer</td>
<td>WMO Block Station ID, or other identifier for buoy or ship</td>
</tr>
<tr>
<td>SName</td>
<td>Call sign and full name of an observing station</td>
<td>A string of the form &quot;ccccc, name&quot;, where ccccc are the call letters of the station (ICAO station id: 4 or 5 upper-case letters, may be omitted), name is an arbitrary string describing the station</td>
<td>The observing stations ICAO, aircraft, or ship call sign, plus a plain-text station name (e.g. &quot;KMRY, Monterey CA Airport&quot;) Example: Sname='KYNL, YUMA (MCAS)'</td>
</tr>
<tr>
<td>Elev</td>
<td>Elevation</td>
<td>A non-negative integer, or omitted if unknown.</td>
<td>Station elevation relative to sea level, in meters. This attribute may specify a surface elevation of an observation station, or an upper-air elevation for an upper-air report. Example: Elev='16'</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------</td>
<td>-----------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>TStamp</td>
<td>Time Stamp</td>
<td>&lt;---See Table 1-1---</td>
<td></td>
</tr>
<tr>
<td>LatLon</td>
<td>Station latitude and longitude</td>
<td>&lt;---See Table 1-1---</td>
<td></td>
</tr>
<tr>
<td>BId</td>
<td>Station Identification Group</td>
<td>Unsigned integer</td>
<td>WMO Block Station ID</td>
</tr>
<tr>
<td>SName</td>
<td>Call sign and full name of an observing station</td>
<td>&lt;---See Table 1-1---</td>
<td></td>
</tr>
<tr>
<td>Elev</td>
<td>Station elevation</td>
<td>&lt;---See Table 1-1---</td>
<td></td>
</tr>
<tr>
<td>Vis</td>
<td>Visibility</td>
<td>a number of meters, omitted, or a special token &quot;INF&quot;</td>
<td>Horizontal visibility in meters</td>
</tr>
<tr>
<td>Ceiling</td>
<td>Ceiling</td>
<td>a number of feet, omitted, or a special token &quot;INF&quot;</td>
<td>Ceiling in feet</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TStamp</td>
<td>Time Stamp</td>
<td></td>
<td>See Table 1-1</td>
</tr>
<tr>
<td>LatLon</td>
<td>Station latitude and longitude</td>
<td></td>
<td>See Table 1-1</td>
</tr>
<tr>
<td>Bid</td>
<td>WMO Block Station Number</td>
<td>String</td>
<td>For a buoy or other observation platform, this id is a combination of a WMO region number, subarea number (per WMO Code Table 0161), and the buoy type and serial number. This information is reported in Section 0 of a synoptic report. If Section 0 contains a call sign rather than a numerical id (as typical with FM 13 SHIP reports), the Bid attribute is computed as itoa(1000009 + hc) % 2^30, where hc is a numerical representation of the call letters considered as a number in radix 36 notation. For example, &quot;0000&quot; hashes to 0, and &quot;ZZZZ&quot; hashes to 1,679,615. Note this formula makes the Bid attribute a unique numeric identifier for the station.</td>
</tr>
<tr>
<td>SName</td>
<td>Call sign and full name of an observing station</td>
<td></td>
<td>See Table 1-1</td>
</tr>
<tr>
<td>Elev</td>
<td>Station elevation</td>
<td></td>
<td>See Table 1-1</td>
</tr>
<tr>
<td>Title</td>
<td>Report title</td>
<td>String</td>
<td>Title defining type of report: AAXX (FM-12), BBXX (FM-13), or ZZYY (FM-18)</td>
</tr>
<tr>
<td>Stype</td>
<td>Station type</td>
<td>String</td>
<td>Type of station: automated (AUTO) or manned (MANN); defaults to MANN</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------</td>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>T</td>
<td>Air Temperature</td>
<td>positive, zero, or negative number</td>
<td>Air temperature in degrees Celsius</td>
</tr>
<tr>
<td>TD</td>
<td>Dew point Temperature</td>
<td>positive, zero, or negative number</td>
<td>Dew point temperature in degrees Celsius</td>
</tr>
<tr>
<td>Hum</td>
<td>Relative humidity</td>
<td>non-negative number</td>
<td>Relative humidity in per cent</td>
</tr>
<tr>
<td>Tmm</td>
<td>Extreme temperatures over the last 24 hours</td>
<td>a string of a form &quot;mmmm, mmmm&quot; or omitted</td>
<td>Minimum and maximum temperatures (degrees Celsius) over the last 24 hours</td>
</tr>
<tr>
<td>P</td>
<td>Station pressure</td>
<td>positive number</td>
<td>Atmospheric pressure at station level, in hectoPascals</td>
</tr>
<tr>
<td>P0</td>
<td>Sea level pressure</td>
<td>positive number</td>
<td>Atmospheric pressure at station, reduced to sea level, in hPa</td>
</tr>
<tr>
<td>Pd</td>
<td>Pressure tendency</td>
<td>String of form &quot;dddddd&quot;, or omitted</td>
<td>Pressure tendency during the 3 hours preceding the observation</td>
</tr>
<tr>
<td>Vis</td>
<td>Visibility Number of meters, omitted, or a special token &quot;INF&quot;</td>
<td>Horizontal visibility in meters</td>
<td>Horizontal visibility in meters</td>
</tr>
<tr>
<td>Ceiling</td>
<td>Ceiling</td>
<td>Number of feet, omitted, or a special token &quot;INF&quot;</td>
<td>Ceiling in feet</td>
</tr>
<tr>
<td>Wind</td>
<td>Wind speed and direction</td>
<td>String of form &quot;nnn, mm&quot; or omitted</td>
<td>$\text{nnn}$ is a true direction from which the wind is blowing, in degrees, or $\text{VAR}$ if &quot;the wind is variable, or all directions or unknown or waves confused, direction indeterminate.&quot; This is an integer number within ${0, 360}$, with 0 meaning the wind is blowing from true North, 270 stands for the wind blowing from due West. Normally this number has a precision of 10 degrees. $\text{mm}$ is the wind speed in meters per second.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>Wx</td>
<td>Past and present weather conditions and phenomena</td>
<td>String of four digits, &quot;NOSIG&quot;, or omitted</td>
<td>See WMO-306, Code tables 4677 and 4561 for the meaning of the four digits. This attribute is coded as &quot;NOSIG&quot; if there is no significant phenomenon to report. The attribute is omitted if not observed or data is not available (see 1ร indicator, Code table 1860).</td>
</tr>
<tr>
<td>Prec</td>
<td>Precipitation amount</td>
<td>String of form &quot;nnn, hh&quot; or &quot;&quot; or omitted</td>
<td>nnn is the amount of precipitation which has fallen during the period preceding the time of observation. The precipitation amount is a non-negative decimal number, in mm. hh is the duration of the period in which the reported precipitation occurred, in whole hours. This attribute is encoded as &quot;&quot; if no precipitation was observed. The attribute is omitted if unknown or not available (see 1ร indicator, Code table 1819). Sea stations typically never report precipitation.</td>
</tr>
<tr>
<td>Clouds</td>
<td>Amounts and types of cloud cover</td>
<td>String of five symbols &quot;cplmh&quot; or omitted</td>
<td>The first digit is the total cloud cover in octas (Code table 2700). The second digit is the cloud cover of the lowest clouds, in octas. The other three symbols are types of low, middle, and high clouds, resp. See WMO-306 Code tables for more details.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>T</td>
<td>Sea surface temperature</td>
<td>Positive, zero, or negative number</td>
<td>Sea surface temperature in degrees Celsius</td>
</tr>
<tr>
<td>Wave</td>
<td>Sea wave period and height</td>
<td>String of form &quot;pp, hh&quot; or omitted</td>
<td>pp is the period of wind waves in seconds. hh is the height of wind waves, in meters. If a report carries both estimated and measured wind wave data, the instrumented information is preferred.</td>
</tr>
<tr>
<td>SDir</td>
<td>Ship's course and speed</td>
<td>String of form &quot;nnn, mm&quot; or omitted.</td>
<td>nnn is a true direction of resultant displacement of the ship during the three hours preceding the time of observation. The number is in degrees, or VAR if &quot;variable, or all directions or unknown or waves confused, direction indeterminate.&quot; This is an integer number within [0, 360), with 0 meaning the ship has moved towards the true North; 270 means the ship has moved to the West. Normally this number has a precision of 45 degrees. mm is the average speed made good during the three hours preceding the time of observation, in meters per second.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>Ref</td>
<td>Reference to sounding Part</td>
<td>String - &quot;TTAA&quot;, &quot;TTBB&quot;, etc.</td>
<td>Reference to the part of the sounding from which the level data were derived</td>
</tr>
<tr>
<td>P</td>
<td>Pressure</td>
<td>positive number</td>
<td>Atmospheric pressure at sounding level, in hectoPascals</td>
</tr>
<tr>
<td>H</td>
<td>Geopotential height</td>
<td>Non-negative number of geopotential meters, or 'SURF' for surface, 'TROP' for tropopause, 'MAXW' for level of maximum winds, 'MAXWTOP' for maximum wind level at the top of the sounding, or omitted</td>
<td>Geopotential height of the reported level, or a special height indicator</td>
</tr>
<tr>
<td>T</td>
<td>Air Temperature</td>
<td>positive, zero, or negative number</td>
<td>Air temperature in degrees Celsius at the reported level</td>
</tr>
<tr>
<td>DP</td>
<td>Dew point temperature</td>
<td>positive, zero, or negative number</td>
<td>Dew point temperature in degrees Celsius at the reported level</td>
</tr>
<tr>
<td>Wind</td>
<td>Wind speed and direction</td>
<td>String of form &quot;nnn, mm&quot; or &quot;nnn, mm bbb&quot; or &quot;nnn, mm ,aaa&quot; or &quot;nnn, mm bbb, aaa&quot; or omitted</td>
<td>nnn is a true direction from which the wind is blowing, in degrees, or VAR if &quot; the wind is variable, or all directions or unknown or waves confused, direction indeterminate.&quot; This is an integer number within [0, 360), with 0 meaning the wind is blowing from true North, 270 stands for the wind blowing from due West. Normally this number has a precision of 10 degrees. mm is the wind speed in meters per second. If specified, bbb stands for the absolute value of the vector difference between the wind at a given level, and the wind 1 km below that level, in meters per second. The number aaa if given is the absolute value of the vector difference between the wind at a given level, and the wind 1 km above that level, in meters per second.</td>
</tr>
</tbody>
</table>
Table 1-7. OMF Attributes for the BTSC Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Brief Description</th>
<th>Format</th>
<th>Description</th>
<th>Req’d?</th>
</tr>
</thead>
<tbody>
<tr>
<td>TStamp</td>
<td>Time Stamp</td>
<td>&lt;-------- See Table 1-1 --------&gt;</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>LatLon</td>
<td>Latitude and Longitude of observation</td>
<td>&lt;-------- See Table 1-1 --------&gt;</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Bld</td>
<td>Station identifier group</td>
<td>positive integer</td>
<td>For a buoy or other observation platform, this ID is a combination of a WMO region number, subarea number (per WMO-306 Code Table 0161), and the buoy type and serial number. This information is reported in Section 4 of a BTSC report. If Section 4 contains a call sign rather than a numerical ID, the Bld attribute is computed as itoa(1000009 + hc), where hc is a numerical representation of the call letters considered as a number in radix 36 notation. For example, &quot;0000&quot; hashes to 0, and &quot;zzzz&quot; hashes to 1,679,615. Note this formula makes the Bld attribute a unique numeric identifier for the station.</td>
<td>Yes</td>
</tr>
<tr>
<td>SName</td>
<td>Call sign</td>
<td>string</td>
<td>Ship’s call sign, if reported</td>
<td>Yes</td>
</tr>
<tr>
<td>Title</td>
<td>Report type</td>
<td>string</td>
<td>&quot;Jyyyy&quot; - FM 63 X Ext. BATHY report &quot;kxxxx&quot; - FM 64 IX TESAC report &quot;nnnxx&quot; - FM 62 TRACKOB report</td>
<td>Yes</td>
</tr>
<tr>
<td>Depth</td>
<td>Water depth</td>
<td>positive number</td>
<td>Total water depth at point of observation</td>
<td>No</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
<td>Req'd?</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>DZ</td>
<td>Indicator for digitization</td>
<td>&quot;7&quot; or &quot;8&quot; or omitted</td>
<td>Indicator for method of digitization used in the report (k1 field). See WMO-306 Code Table 2262. Required for BATHY and TESAC reports</td>
<td>No</td>
</tr>
<tr>
<td>Rec</td>
<td>Instrument type code</td>
<td>5-digit code</td>
<td>Code for expendable bathythermograph (XBT) instrument type and fall rate (WMO-306 Code Table 1770).</td>
<td>No</td>
</tr>
<tr>
<td>WS</td>
<td>Wind speed units code</td>
<td>&quot;0&quot;, &quot;1&quot;, &quot;2&quot;, &quot;3&quot;, or omitted</td>
<td>Indicator for units of wind speed and type of instrumentation (u field). See WMO-306, Code Table 1853.</td>
<td>No</td>
</tr>
<tr>
<td>Curr-s</td>
<td>Method of current speed measurement</td>
<td>&quot;2&quot;, &quot;3&quot;, &quot;4&quot;, or omitted</td>
<td>Indicator for the method of current measurement (k5 field). See WMO-306 Code Table 2266.</td>
<td>No</td>
</tr>
<tr>
<td>Curr-d</td>
<td>Indicators for the method of subsurface current measurement</td>
<td>3-digit numerical code</td>
<td>Indicators for the method of subsurface current measurement (k6x4k3 codes). See WMO-306, Code Tables 2267, 2265, and 2264.</td>
<td>No</td>
</tr>
<tr>
<td>AV-T</td>
<td>Averaging period for sea temperature</td>
<td>&quot;0&quot;, &quot;1&quot;, &quot;2&quot;, &quot;3&quot;, or omitted (if no sea temperature data are reported)</td>
<td>Code for the averaging period for sea temperature (mR code). See WMO-306, Code Table 2604</td>
<td>No</td>
</tr>
<tr>
<td>AV-SAL</td>
<td>Averaging period for salinity.</td>
<td>&quot;0&quot;, &quot;1&quot;, &quot;2&quot;, &quot;3&quot;, or omitted (if no salinity data are reported)</td>
<td>Code for the averaging period for sea salinity (mS code). See WMO-306, Code Table 2604</td>
<td>No</td>
</tr>
<tr>
<td>AB-Curr</td>
<td>Averaging period for surface current direction and speed</td>
<td>&quot;0&quot;, &quot;1&quot;, &quot;2&quot;, &quot;3&quot;, or omitted (if no current data are reported)</td>
<td>Code for the averaging period for surface current direction and speed (mC code). See WMO-306, Code Table 2604</td>
<td>No</td>
</tr>
<tr>
<td>Sal</td>
<td>Method of salinity/depth measurement</td>
<td>&quot;1&quot;, &quot;2&quot;, &quot;3&quot;, or omitted (if no salinity data are reported)</td>
<td>Code for the method of salinity/depth measurement (k2 code). See WMO-306, Code Table 2263.</td>
<td>No</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
<td>Req'd?</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
<td>--------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>T</td>
<td>Air temperature</td>
<td>Positive, zero, or negative number, or omitted</td>
<td>Air temperature just above the sea surface, in degrees Celsius.</td>
<td>No</td>
</tr>
<tr>
<td>Wind</td>
<td>Wind vector</td>
<td>String of form &quot;nnn, mm&quot;, or omitted</td>
<td>Here nnn is a true direction from which the wind is blowing, in degrees, or VAR if &quot; the wind is variable, or all directions or unknown or waves confused, direction indeterminate.&quot; This is an integer number within [0, 360), with 0 meaning the wind is blowing from the true North:, 270 means the wind is blowing from the West. Normally this number has a precision of 10 degrees. mm is the wind speed in meters per second.</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 1-10. OMF Attributes for the BTLEVEL Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Brief Description</th>
<th>Format</th>
<th>Description</th>
<th>Req'd?</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Depth</td>
<td>Non-negative number</td>
<td>Depth of the level in meters.</td>
<td>Yes</td>
</tr>
<tr>
<td>T</td>
<td>Water temperature</td>
<td>Positive, zero, or negative number, or omitted</td>
<td>Water temperature at the reported level.</td>
<td>No</td>
</tr>
<tr>
<td>S</td>
<td>Salinity</td>
<td>Positive number, or omitted</td>
<td>Salinity at the reported level, in parts per thousand.</td>
<td>No</td>
</tr>
<tr>
<td>C</td>
<td>Current vector String of form</td>
<td>&quot;###, ##&quot;, or omitted</td>
<td>## is the true direction toward which the sea current is moving, in degrees, or VAR if &quot;the current is variable, or all directions or unknown, direction indeterminate.&quot; This is an integer number within (0,360), with 0 meaning the current flows toward true North; 270 means the current is flowing toward the West. Normally this number has a precision of 10 degrees. ## is the speed of current in meters per second.</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 1-11. OMF Attributes for the TAF Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Brief Description</th>
<th>Format</th>
<th>Description</th>
<th>Req'd?</th>
</tr>
</thead>
<tbody>
<tr>
<td>TStamp</td>
<td>Time Stamp</td>
<td></td>
<td>See Table 1-1</td>
<td>Yes</td>
</tr>
<tr>
<td>LatLon</td>
<td>Latitude and Longitude of observation</td>
<td></td>
<td>See Table 1-1</td>
<td>Yes</td>
</tr>
<tr>
<td>Bld</td>
<td>Block Station ID</td>
<td>positive integer</td>
<td>WMO Block Station ID of the reporting station</td>
<td>Yes</td>
</tr>
<tr>
<td>SName</td>
<td>Call sign</td>
<td>string</td>
<td>Ship's call sign, if reported</td>
<td>Yes</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
<td>Req'd?</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>class</td>
<td>SIGMET type</td>
<td>&quot;CONVICTIVE&quot;, &quot;HOTEL&quot;, &quot;INDIA&quot;, &quot;UNIFORM&quot;, &quot;VICTOR&quot;, &quot;WHISKEY&quot;</td>
<td>Identifier for the type of SIGMET message</td>
<td>Yes</td>
</tr>
<tr>
<td>id</td>
<td>Identifier for a particular advisory</td>
<td>String</td>
<td>Identifier for the advisory; value depends on the advisory class.</td>
<td>Yes</td>
</tr>
<tr>
<td>TStamp</td>
<td>Time Stamp</td>
<td></td>
<td>See Table 1-1</td>
<td>Yes</td>
</tr>
<tr>
<td>BBox</td>
<td>Bounding box for advisory area</td>
<td></td>
<td>See Table 1-1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Brief Description</th>
<th>Format</th>
<th>Description</th>
<th>Req'd?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape</td>
<td>Type of area specification</td>
<td>&quot;AREA&quot;, &quot;LINE&quot;, &quot;POINT&quot;</td>
<td>Type of area shape specified</td>
<td>Yes</td>
</tr>
<tr>
<td>LatLons</td>
<td>List of latitudes and longitudes defining the area</td>
<td>Positive, zero, or negative numbers in lat/lon pairs</td>
<td>Control points (vertices) for a polygon/polyline representing the affected area</td>
<td>Yes</td>
</tr>
<tr>
<td>Attribute</td>
<td>Brief Description</td>
<td>Format</td>
<td>Description</td>
<td>Req'd?</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------</td>
<td>--------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>id</td>
<td>Message identifier</td>
<td>A NMTOKEN, a four-to-six-character string of a form r1r2a1a2ii</td>
<td>Designator for the message type and subtype (r1r2), area (a1a2), and sequence code (ii) of the message, as described in WMO-386.</td>
<td>Yes</td>
</tr>
<tr>
<td>Type</td>
<td>Message type</td>
<td>2-letter string (r1r2)</td>
<td>Designator for the message type and subtype (r1r2) as specified in WMO-386, Tables A and B1 through B6</td>
<td>Yes</td>
</tr>
<tr>
<td>TStamp</td>
<td>Time Stamp</td>
<td>String</td>
<td>See Table 1-1</td>
<td>Yes</td>
</tr>
<tr>
<td>SName</td>
<td>Originating station name</td>
<td>String</td>
<td>String containing the identification of the station that originated the message (normally its ICAO call sign)</td>
<td>Yes</td>
</tr>
<tr>
<td>BBB</td>
<td>Annotation group</td>
<td>3-character string</td>
<td>So-called &quot;BBB groups&quot; from the abbreviated message line. They indicate that the message has been delayed, corrected or amended. A BBB group can also be used for segmentation. See the WMO-386 for more detail.</td>
<td>No</td>
</tr>
<tr>
<td>Descr</td>
<td>Description</td>
<td>String</td>
<td>Keywords and other information describing the message.</td>
<td>No</td>
</tr>
<tr>
<td>BBox</td>
<td>Bounding box</td>
<td>String</td>
<td>See Table 1-1</td>
<td>No</td>
</tr>
<tr>
<td>layer</td>
<td>Description</td>
<td>Example</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adiabatic-cond</td>
<td>Adiabatic condensation level (parcel lifted from surface)</td>
<td>(layer adiabatic-cond)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atm-top</td>
<td>Level of the top of the atmosphere</td>
<td>(layer atm-top)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cloud-base</td>
<td>Cloud base level</td>
<td>(layer cloud-base)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cloud-top</td>
<td>Cloud top level</td>
<td>(layer cloud-top)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv-cld-base</td>
<td>Level of bases of convective clouds</td>
<td>(layer conv-cld-base)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv-cld-top</td>
<td>Level of tops of convective clouds</td>
<td>(layer conv-cld-top)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>entire-atm</td>
<td>Entire atmosphere</td>
<td>(layer entire-atm)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>entire-ocean</td>
<td>Entire ocean</td>
<td>(layer entire-ocean)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>height</td>
<td>Height above ground (meters)</td>
<td>(layer height 1500)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>height-between</td>
<td>Layer between two heights above ground in hundreds meters (followed by top and bottom level values)</td>
<td>(layer height-between 50 30) for layer between 5000 and 3000 meters above ground</td>
<td></td>
<td></td>
</tr>
<tr>
<td>height-between-ft</td>
<td>Layer between two heights above ground, in feet (followed by top and bottom level values)</td>
<td>(layer height-between-ft 15000 10000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>height-ft</td>
<td>Height above ground (feet)</td>
<td>(layer height-ft 50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>high-cld-base</td>
<td>Level of high cloud bases</td>
<td>(layer high-cld-base)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>high-cld-top</td>
<td>Level of high cloud tops</td>
<td>(layer high-cld-top)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hybrid</td>
<td>Hybrid level (followed by level number)</td>
<td>(layer hybrid 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hybrid-between</td>
<td>Layer between two hybrid levels (followed by top and bottom level numbers)</td>
<td>(layer hybrid 2 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isobar</td>
<td>Level of an isobar surface (followed by the isobar value of the surface in hectoPascals (hPa) (1000, 975, 950, 925, 900, 850, 800, 750, 700, 65, 0, 500, 550, 500, 450, 400, 350, 300, 250, 200, 150, 100, 70, 50, 30, 20, 10)</td>
<td>(layer isobar 500)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isobar-between</td>
<td>Layer between two isobar surfaces (followed by top and bottom isobar values in kPa, separated by a space)</td>
<td>(layer isobar-between 50 100) for layer between 500 and 1000 hPa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isobar-between-mp</td>
<td>Layer between two isobar surfaces, mixed precision (followed by pressure of top in kPa and 1100 minus pressure of bottom in hPa)</td>
<td>(layer isobar-between-mp 50 100) for layer between 500 and 1000 hPa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Layer</td>
<td>Description</td>
<td>Example</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isobar-between-xp</td>
<td>Layer between two isobaric surfaces, extra precision (followed by top and</td>
<td>(layer isobar-between 600 100) for layer between 500 and 1000 hPa</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bottom isobar values expressed as 1100 hPa-isobar level, separated by a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>space)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isotherm-0</td>
<td>Level of the zero-degree (Celsius) isotherm (or freezing level)</td>
<td>(layer isotherm-0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>land-depth</td>
<td>Depth below land surface in centimeters</td>
<td>(layer land-depth 5.0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>land-depth-between</td>
<td>Layer between two depths in ground (followed by the depth of the top of</td>
<td>(layer land-depth-between 0 30) for layer from ground surface to 30 cm</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the layer and the depth of the bottom of the layer centimeters)</td>
<td>depth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>land-height-cm</td>
<td>Height level above ground (high precision) (followed by height in centimeters)</td>
<td>(layer land-height-cm 50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>land-isobar</td>
<td>Pressure above ground level in hPa</td>
<td>(layer land-isobar 500)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>land-isobar-between</td>
<td>Layer between two isobars above levels (followed by top and bottom isobaric</td>
<td>(layer land-isobar-between 500 1000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>levels in hPa)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>low-cld-base</td>
<td>Level of low cloud bases</td>
<td>(layer low-cld-base)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>low-cld-top</td>
<td>Level of low cloud tops</td>
<td>(layer low-cld-top)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max-wind</td>
<td>Level of maximum wind</td>
<td>(layer max-wind)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mid-cld-base</td>
<td>Level of middle cloud bases</td>
<td>(layer mid-cld-base)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mid-cld-top</td>
<td>Level of middle cloud tops</td>
<td>(layer mid-cld-top)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>msl</td>
<td>Mean sea level</td>
<td>(layer msl)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>msl-height</td>
<td>Height above mean sea level (in meters)</td>
<td>(layer msl-height 50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>msl-height-between</td>
<td>Layer between two heights above mean sea level in hundreds of meters (followed</td>
<td>(layer msl-height-between 10 5) for layer between 1000 and 500 meters</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>by top and bottom height values)</td>
<td>above ground</td>
<td></td>
<td></td>
</tr>
<tr>
<td>msl-height-ft</td>
<td>Height above mean sea level (in feet)</td>
<td>(layer msl-height-ft 5000)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sea-bottom</td>
<td>Bottom of the ocean</td>
<td>(layer sea-bottom)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sea-depth</td>
<td>Depth below the sea surface (meters)</td>
<td>(layer sea-depth 50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sigma</td>
<td>Sigma level in 1/10000</td>
<td>(layer sigma 9950) for sigma level .995</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sigma-between</td>
<td>Layer between two sigma surfaces (followed by top and bottom sigma values</td>
<td>(layer sigma-between 99.5 100.0) for layer between .995 and 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>expressed in 1/100, separated by a space)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Layer</td>
<td>Description</td>
<td>Example</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sigma-between-xp</td>
<td>Layer between two sigma levels (followed by top and bottom sigma values expressed as 1.1-sigma)</td>
<td>(layer sigma-between-xp .105 .100) for layer between .995 and 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>surface</td>
<td>Earth’s surface</td>
<td>(layer surface)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theta</td>
<td>Isentropic (theta) level (followed by potential temperature in degrees K)</td>
<td>(layer theta 300)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theta-between</td>
<td>Layer between two isentropic surfaces (followed by top and bottom values expressed as 475-theta in degrees K)</td>
<td>(layer theta-between 150 200)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tropopause</td>
<td>Level of tropopause (top of troposphere)</td>
<td>(layer tropopause)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### PowerPoint API Function Description Table

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Represents the entire Microsoft PowerPoint application.</td>
<td>MyPath = Application.Path</td>
</tr>
<tr>
<td>ActivePresentation</td>
<td>Returns a Presentation object that represents the presentation open in the active window. (Read-only)</td>
<td>Application .ActivePresentation.SaveAs MyPath</td>
</tr>
<tr>
<td>Presentations</td>
<td>Returns a Presentation object that represents the presentation in which the specified document window or slide show window was created. (Read-only)</td>
<td>firstPresSlides = Windows(1).Presentation.Slides.Count Windows(2).Presentation.PageSetup FirstSlideNumber = firstPresSlides + 1</td>
</tr>
<tr>
<td>Presentations.Add</td>
<td>Creates a presentation. Returns a Presentation object that represents the new presentation.</td>
<td>This example creates a presentation, adds a slide to it, and then saves the presentation. With Presentations.Add .Slides.Add 1, ppLayoutTitle .SaveAs &quot;Sample&quot; End With</td>
</tr>
<tr>
<td>Slides</td>
<td>A collection of all the Slide objects in the specified presentation.</td>
<td>Use the Slides property to return a Slides collection: ActivePresentation.Slides.Add 2, ppLayoutBlank</td>
</tr>
<tr>
<td>Slides.Add</td>
<td>Creates a new slide and adds it to the collection of slides in the specified presentation. Returns a Slide object that represents the new slide.</td>
<td>This example adds a blank slide at the end of the active presentation. With ActivePresentation.Slides .Add .Count + 1, ppLayoutBlank End With</td>
</tr>
<tr>
<td>Shapes</td>
<td>A collection of all the Shape objects on the specified slide. Each Shape object represents an object in the drawing layer, such as an AutoShape, freeform, OLE object, or picture.</td>
<td>Use the Shapes property to return the Shapes collection. The following example selects all the shapes on myDocument. Set myDocument = ActivePresentation.Slides(1) myDocument.Shapes.SelectAll</td>
</tr>
<tr>
<td>Shapes.AddPicture</td>
<td>Creates a picture from an existing file. Returns a Shape object that represents the new picture.</td>
<td>Set myDocument = ActivePresentation.Slides(1) myDocument.Shapes.AddPicture &quot;c:\microsoft office&quot; &amp; _ &quot;clipart\music.bmp&quot;, True, True, 100, 100, 70, 70</td>
</tr>
<tr>
<td>Shapes.PictureFormat</td>
<td>Contains properties and methods that apply to pictures and OLE objects. The LinkFormat object contains properties and methods that apply to linked OLE objects only. The OLEFormat object contains properties and methods</td>
<td>Set myDocument = ActivePresentation.Slides(1) With myDocument.Shapes(1).PictureFormat</td>
</tr>
</tbody>
</table>
that apply to OLE objects whether or not they're linked.

.Brightness = 0.3
.Contrast = 0.7
.ColorType = msoPictureGrayScale
.CropBottom = 18
End With

PowerPoint API Function Description Table (Cont.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SlideShowTransition</td>
<td>Contains information about how the specified slide advances during a slide show.</td>
<td>With ActivePresentation.Slides(1).SlideShowTransition .Speed = ppTransitionSpeedFast End With</td>
</tr>
<tr>
<td>SlideShowSetting</td>
<td>Represents the slide show setup for a presentation.</td>
<td>With ActivePresentation.SlideShowSettings .RangeType = ppShowSlideRange End With</td>
</tr>
</tbody>
</table>
Figure 3 – Wrapper & Glue Code Object Diagram
Figure 5 – Continuous Brief Update Sequence Diagram
APPENDIX D. SOURCE CODE
1. Configuration GUI (CBcfg)

VERSION 5.00
Begin VB.Form CBform
  BackColor = &H80000004&
  Caption = "CBcfg"
  ClientHeight = 9195
  ClientLeft = 60
  ClientTop = 345
  ClientWidth = 8490
  LinkTopic = "Form1"
  ScaleHeight = 9195
  ScaleWidth = 8490
  StartUpPosition = 3 'Windows Default
End
Begin VB.TextBox VirtualDirText
  Height = 375
  Left = 1080
  TabIndex = 3
  Tag = "3"
  Top = 7320
  Width = 6375
End
Begin VB.TextBox TypeText
  Height = 375
  Left = 1080
  TabIndex = 1
  Top = 5160
  Width = 6375
End
Begin VB.CommandButton Delete
  Caption = "Delete"
  Enabled = 0 'False
BeginProperty Font
  Name = "MS Sans Serif"
  Size = 9.75
  Charset = 0
  Weight = 700
  Underline = 0 'False
  Italic = 0 'False
  Strikethrough = 0 'False
EndProperty
  Height = 375
  Left = 4440
  TabIndex = 6
  Top = 8160
Width = 1335

End

Begin VB.CommandButton Add
Caption = "Set"
Enabled = 0 'False

BeginProperty Font
Name = "MS Sans Serif"
Size = 9.75
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False

EndProperty
Height = 375
Left = 6120
TabIndex = 7
Top = 8160
Width = 1335

End

Begin VB.CommandButton Cancel
Caption = "Cancel"

BeginProperty Font
Name = "MS Sans Serif"
Size = 9.75
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False

EndProperty
Height = 375
Left = 2760
TabIndex = 5
Top = 8160
Width = 1335

End

Begin VB.CommandButton OK
Caption = "OK"

BeginProperty Font
Name = "MS Sans Serif"
Size = 9.75
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False

EndProperty
Height = 375
Left = 1080
TabIndex = 4
Top = 8160
Width = 1335
End

Begin VB.TextBox LocationText
    Height = 375
    Left = 1080
    TabIndex = 2
    Tag = "3"
    Top = 6240
    Width = 6375
End

Begin VB.ListBox dataList
    Height = 3570
    Left = 1080
    TabIndex = 0
    Top = 720
    Width = 6375
End

Begin VB.Label Label2
    Caption = "Virtual directory (optional):"
    BeginProperty Font
        Name = "MS Sans Serif"
        Size = 9.75
        CharSet = 0
        Weight = 700
        Underline = 0 'False
        Italic = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height = 255
    Left = 1080
    TabIndex = 11
    ToolTipText = "A virtual directory associated with the key used by the Web server."
    Top = 6840
    Width = 2775
End

Begin VB.Label Label4
    Caption = "Key:
    BeginProperty Font
        Name = "MS Sans Serif"
        Size = 9.75
        CharSet = 0
        Weight = 700
        Underline = 0 'False
        Italic = 0 'False
        Strikethrough = 0 'False
End
EndProperty
Height = 255
Left = 1080
TabIndex = 10
ToolTipText = "An image type or any other variable name."
Top = 4680
Width = 615
End
Begin VB.Label Label3
Caption = "Directory:"
EndProperty
Name = "MS Sans Serif"
Size = 9.75
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
Height = 255
Left = 1080
TabIndex = 9
ToolTipText = "An actual directory associated with the key."
Top = 5760
Width = 1095
End
Begin VB.Label Label1
Caption = "Current configuration:"
EndProperty
Name = "MS Sans Serif"
Size = 9.75
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
Height = 255
Left = 1080
TabIndex = 8
ToolTipText = "The current setting for Continuous Brief application."
Top = 240
Width = 2295
End
End
Attribute VB_Name = "CBform"
Private cfgfile As String
Private cftmp As String

' Unload the CBCfg form when the Cancel button is clicked.

Private Sub Cancel_Click()
    Unload Me
End Sub

' Display information for each record selected from the
' current configuration list box.

Private Sub dataList_Click()
    Dim listStr As String
    Dim typeStr As String
    Dim locationStr As String
    Dim virtualStr As String

    listStr = dataList.Text
    Call lineInfo(listStr, typeStr, locationStr, virtualStr)
    ' Display the key name in the Key text box.
TypeText.Text = typeStr
' Display the directory associated with the key in the
' Directory text box.
LocationText.Text = locationStr
' Display the virtual directory associated with the key
' in the Virtual Directory text box
VirtualDirText.Text = virtualStr
Add.Enabled = False
Delete.Enabled = True
End Sub

Tasks done when deleting an item from the list.
First, copy all lines from the cfgfile to the cfgtmp
file except the line that's being deleted. Then copy
back to the cfgfile from the cfgtmp.

Private Sub Delete_Click()
Open cfgfile For Input As #1
Open cfgtmp For Output As #2
Do While Not EOF(1)
    Line Input #1, inputStr
    If Not (InStr(1, inputStr, TypeText.Text & "=")) Then
        Print #2, inputStr
    End If
End Loop
Close #1
Close #2
' Copy the cfgtmp to the cfgfile
Open cfgtmp For Input As #1
Open cfgfile For Output As #2
Do While Not EOF(1)
    Line Input #1, inputStr
    Print #2, inputStr
End Loop
Close #1
Close #2
Call updateList
End Sub

Tasks done when the application is load.
This requires two system environment variables set,
which are CB_HOME, where the cbdata.cfg is located, and
CB_TMP, where the temporary file is created.

175
Private Sub Form_Load()
cfgfile = Environ("CB_HOME") & "\cbdata.cfg"
cfgtmp = Environ("TEMP") & "\cbdata_tmp"
Call updateList
End Sub

******************************************************************************

' Activate the Add button if new value is entered from
' the Image type box.

******************************************************************************
Private Sub KeyText_Change()
    Add.Enabled = True
End Sub

******************************************************************************

' Save the changes (if any), and close the CBcfg form
' when the OK button is clicked.

******************************************************************************
Private Sub OK_Click()
    If (Add.Enabled) Then
        Call Add_Click
    End If
    Unload Me
End Sub

******************************************************************************

' The lineInfo subroutine parses a line input from the
' configuration file (cbdata.cfg). It separates information
' of the key, the directory, and the virtual directory
' from the line string input.
' Parameters:
'    in:
'        searchString - the string is being parsed.
'    in/out:
'        K - a variable that holds the key string
'        D - a variable that holds the directory string
'        V - a variable that holds the virtual directory string
'
******************************************************************************
Private Sub lineInfo(searchString As String, K As String, D As String, V As String)
    istart = 1
    istop = 0
    istop = InStr(istart, searchString, ",", vbTextCompare)
    ' Get the key string

    istart = istop + 1
    istop = InStr(istart, searchString, ":", vbTextCompare)
Print #2, inputStr
    Loop
    Close #1
    Close #2
    Call updateList
End Sub

'******************************************************************************
' Activate the Add button if new value is entered from
' the Key text box.
'******************************************************************************
Private Sub TypeText_Change()
    Add.Enabled = True
End Sub

'******************************************************************************
' Activate the Add button if new value is entered from
' the Directory text box.
'******************************************************************************
Private Sub locationText_Change()
    Add.Enabled = True
End Sub

'******************************************************************************
' Refresh the GUI after adding or deleting an item from
' the list.
'******************************************************************************
Private Sub updateList()
    Dim intFile As Integer
    dataList.Clear

    intFile = FreeFile()
    Open cfgfile For Input As #intFile
    Do While Not EOF(intFile)  ' Check for end of file.
        Line Input #intFile, inputStr  ' Read line of data.
        dataList.AddItem inputStr
    Loop
    Close #intFile
    TypeText.Text = ""
    LocationText.Text = ""
    VirtualDirText.Text = ""
    Add.Enabled = False
    Delete.Enabled = False
End Sub

'******************************************************************************
' Activate the Add button if new value is entered from
' the Virtual Directory text box.
'
'*******************************************************************************
Private Sub VirtualDirText_Change()
    Add.Enabled = True
End Sub

2. Application Wrapper (CBWrapper)

VERSION 5.00
Object = "{48E59290-9880-11CF-9754-00AA00C00908)#1.0#0";
"MSINET.OCX"
Begin VB.UserControl WebInterface
    BackColor = &H80000001&
    ClientHeight = 5475
    ClientLeft = 0
    ClientTop = 0
    ClientWidth = 8430
    ScaleHeight = 5475
    ScaleWidth = 8430
Begin InetCtlsObjects.Inet Inet1
    Left = 120
    Top = 120
    _ExtentX = 1005
    _ExtentY = 1005
    _Version = 393216
End
Begin VB.TextBox ImagesText
    BeginProperty Font
        Name = "Arial"
        Size = 9.75
        CharSet = 0
        Weight = 700
        Underline = 0 'False
        Italic = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height = 375
    Left = 5880
    TabIndex = 7
    Text = "24"
    Top = 1680
    Width = 735
End
Begin VB.TextBox HeightText
    BeginProperty Font
        Name = "Arial"
        Size = 9.75
    EndProperty
    Height = 375
    Left = 5880
    TabIndex = 7
    Text = "24"
    Top = 1680
    Width = 735
End
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
Height = 375
Left = 5880
TabIndex = 6
Text = "540"
Top = 2520
Width = 735
End
Begin VB.TextBox WidthText
  BeginProperty Font
    Name = "Arial"
    Size = 9.75
    CharSet = 0
    Weight = 700
    Underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
  EndProperty
  Height = 375
  Left = 5880
  TabIndex = 5
  Text = "720"
  Top = 3360
  Width = 735
End
Begin VB.TextBox DurationText
  BeginProperty Font
    Name = "Arial"
    Size = 9.75
    CharSet = 0
    Weight = 700
    Underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
  EndProperty
  Height = 375
  Left = 5880
  TabIndex = 4
  Text = "0"
  Top = 4200
  Width = 735
End
Begin VB.CommandButton Start
  Caption = "Start"
End
BeginProperty Font
    Name = "Arial"
    Size = 9.75
    Charset = 0
    Weight = 700
    Underline = 0 'False
    Italic = 0 'False
    Strikethrough = 0 'False
EndProperty
Height = 495
Left = 720
TabIndex = 3
Top = 2400
Width = 1215
End

Begin VB.CommandButton
    Default = True
    Caption = "Default"
    BeginProperty Font
        Name = "Arial"
        Size = 9.75
        Charset = 0
        Weight = 700
        Underline = 0 'False
        Italic = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height = 495
    Left = 720
    TabIndex = 2
    Top = 4080
    Width = 1215
End

Begin VB.ComboBox
    ImageType = False
    BeginProperty Font
        Name = "Arial"
        Size = 9.75
        Charset = 0
        Weight = 700
        Underline = 0 'False
        Italic = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height = 360
    Left = 720
    TabIndex = 1
    Text = "Select an image type"
    Top = 1680
    Width = 2895
End
Begin VB.CommandButton Stop
BackColor = &H0C0C0C0C&
Caption = "Stop"
BeginProperty Font
  Name = "Arial"
  Size = 9.75
  CharSet = 0
  Weight = 700
  Underline = 0 'False
  Italic = 0 'False
  Strikethrough = 0 'False
EndProperty
Height = 495
Left = 720
MaskColor = &H80000004&
TabIndex = 0
Top = 3240
Width = 1215
End
Begin VB.Label images
BackColor = &H80000001&
Caption = "Images:"
BeginProperty Font
  Name = "Arial"
  Size = 9.75
  CharSet = 0
  Weight = 700
  Underline = 0 'False
  Italic = 0 'False
  Strikethrough = 0 'False
EndProperty
ForeColor = &H8000000E&
Height = 255
Left = 4800
TabIndex = 14
Top = 1680
Width = 855
End
Begin VB.Label Label1
BackColor = &H80000001&
Caption = "Height:"
BeginProperty Font
  Name = "Arial"
  Size = 9.75
  CharSet = 0
  Weight = 700
  Underline = 0 'False
  Italic = 0 'False
  Strikethrough = 0 'False
EndProperty
182
EndProperty
ForeColor = &H8000000E&
Height = 255
Left = 4800
TabIndex = 13
Top = 2520
Width = 735
End
Begin VB.Label Label2
BackColor = &H80000001&
Caption = "Width:"
BeginProperty Font
Name = "Arial"
Size = 9.75
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
ForeColor = &H8000000E&
Height = 255
Left = 4800
TabIndex = 12
Top = 3360
Width = 735
End
Begin VB.Label Label3
BackColor = &H80000001&
Caption = "Duration:"
BeginProperty Font
Name = "Arial"
Size = 9.75
Charset = 0
Weight = 700
Underline = 0 'False
Italic = 0 'False
Strikethrough = 0 'False
EndProperty
ForeColor = &H8000000E&
Height = 255
Left = 4800
TabIndex = 11
Top = 4200
Width = 855
End
Begin VB.Label Label4
BackColor = &H80000001&
Caption = "Second(s)"
End
BeginProperty Font
  Name = "Arial"
  Size = 9.75
  Charset = 0
  Weight = 700
  Underline = 0 'False
  Italic = 0 'False
  Strikethrough = 0 'False
EndProperty
ForeColor = &H80000000E&
Height = 255
Left = 6840
TabIndex = 10
Top = 4200
Width = 975
End
Begin VB.Label Label5
  Alignment = 2 'Center
  BackColor = &H80000001&
  Caption = "CONTINUOUS BRIEF"
BeginProperty Font
  Name = "MS Sans Serif"
  Size = 18
  Charset = 0
  Weight = 700
  Underline = 0 'False
  Italic = 0 'False
  Strikethrough = 0 'False
EndProperty
ForeColor = &H80000000E&
Height = 495
Left = 2280
TabIndex = 9
Top = 360
Width = 3975
End
Begin VB.Label type
  BackColor = &H80000001&
  Caption = "Image type:"
BeginProperty Font
  Name = "Arial"
  Size = 9.75
  Charset = 0
  Weight = 700
  Underline = 0 'False
  Italic = 0 'False
  Strikethrough = 0 'False
EndProperty
ForeColor = &H80000000E&

184
Height = 255
Left = 720
TabIndex = 8
Top = 1200
Width = 1215

End

Attribute VB_Name = "WebInterface"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True

'###################################################################################################################
'# File: WebInterface.ctl
'# Date Author History
'# 5/31/2000 Tam Tran Created.
'###################################################################################################################
Option Explicit

' The Continuous Brief wrapper (CBWrapper) is an ActiveX Control that represents the Graphical User Interface (GUI) via the Web browser (Internet Explorer). It allows an user to select the type of images that he/she wants to view. Also, it allows the user to set the number of images, the size, and the duration for the display.

Private mControllerConnector As ControllerConnector
Private mMonitor As Monitor
Private mMonitorConnector As MonitorConnector
Private WithEvents mController As Controller
Attribute mController.VB_VarHelpID = -1
' Get reference to Application object from the PowerPoint API.
Public mPPT As PowerPoint.Application
Public AppRunning As Boolean
Private BriefStarted As Boolean
Private downloadFolder As String
Private cfgFolder As String
Private ServerURL As String

'###################################################################################################################
' Reset the Continuous Brief GUI to its default values.
' Set slide show to fullscreen size.
' Set number of images to 24
' Set duration of the slide show to 0.
'
Private Sub Default_Click()
    ImageType.Text = "Select an image type"
    ImagesText.Text = "24"
    HeightText.Text = "540"
    WidthText.Text = "720"
    DurationText.Text = "0"
End Sub

Private Sub Start_Click()
    Dim imageloc As String
    BriefStarted = True
    Call mController_UpdateBrief(ImageType.Text)
End Sub

Private Sub Stop_Click()
    If AppRunning Then
        myPPT.ActivePresentation.Close
        myPPT.Quit
        Set myPPT = Nothing
        AppRunning = False
        BriefStarted = False
    End If
End Sub

Private Sub UserControl_Initialize()
Set mControllerConnector = New ControllerConnector
Set mController = mControllerConnector.Controller
Set mMonitorConnector = New MonitorConnector
Set mMonitor = mMonitorConnector.Monitor
AppRunning = False
BriefStarted = False

' Add image types to the drop-box in the Continuous
Brief GUI
Dim intFile As Integer    ' FreeFile variable
Dim inputStr As String
Dim cfgFile As String
Dim typeStr As String
Dim locationStr As String
Dim virtualDirStr As String
Dim tmpFolderStr As String
Dim tmpFileStr As String
Dim downloadFileStr As String

' Set values for the URL, download folder, and a
temporary filename
' ****************************
' Change config here:
ServerURL = "http://tampc.spawar.navy.mil/"
' ****************************
cfgFile = "cbdata.cfg"
downloadFolder = Environ("TEMP") & "\cbdownload"
cfgFolder = downloadFolder & "\cbdata"
tmpFileStr = cfgFolder & "\" & cfgFile

' Download the "cbdata.cfg" file
downloadFileStr = ServerURL & "/" & cfgFile

' Create a temporary directory for downloading data
Call createFolder(downloadFolder)
Call createFolder(cfgFolder)
Call downloadFile(downloadFileStr, tmpFileStr)

intFile = FreeFile()
Open tmpFileStr For Input As #intFile
Do While Not EOF(intFile)
    Line Input #intFile, inputStr
    Call lineInfo(inputStr, typeStr, locationStr,
    virtualDirStr)
    ImageType.AddItem typeStr
Loop
Close #intFile
End Sub
Private Sub mController_UpdateBrief(DataType As String)

    ' Check for the right type of data that the CBWrapper is showing.
    If (StrComp(ImageType.Text, DataType, vbTextCompare) = 0) And BriefStarted Then
        Dim virtualDir As String
        Dim fileName As String
        Dim tmpFileStr As String
        Dim tmpURLStr As String
        Call mController.GetImageInfo(ImageType.Text, ImagesText.Text, _
            virtualDir,
            fileName)

    ' Local variables declarations
    Dim myArray() As String
    Dim myPres As Presentation
    Dim fs, f, fc, f1, i, j, K
    Dim s As Slide
    Dim LeftVal As Long
    Dim TopVal As Long
    Dim imageW As Long
    Dim imageH As Long
    Dim ImgFile As String
    Dim intFile As Integer
    Dim inputStr As String

    ' Download the list of image filenames from server
    tmpURLStr = ServerURL & virtualDir & "/CB_listfile/" & fileName
    Call downloadFile(tmpURLStr, tmpFileStr)

    ' Download image files from server
    intFile = FreeFile()
    Open tmpFileStr For Input As #intFile
    Do While Not EOF(intFile)
        Line Input #intFile, inputStr
        tmpURLStr = ServerURL & virtualDir & "/" & inputStr
    
    188
tmpFileStr = downloadFolder & "\" & inputStr
Call downloadFile(tmpURLStr, tmpFileStr)
Loop
Close #intFile

' Get reference to the PowerPoint Application object.
On Error Resume Next
Set myPPT = GetObject(, "PowerPoint.application")
If Err.Number <> 0 Then
    Set myPPT = CreateObject("PowerPoint.application")
End If

' Set the AppRunning flag so that it will be checked when the STOP button is clicked.
AppRunning = True

' Stop the current running slide show (if any)
If myPPT.Presentations.Count <> 0 Then
    myPPT.ActivePresentation.Close
End If

' Create new presentation with the new update data
Set myPres = myPPT.Presentations.Add(True)

' Create a FileSystemObject for manipulating the file system
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(downloadFolder)
Set fc = f.Files
i = 1
K = 1

' Store all filenames from the image directory to an array for sorting purpose.
ReDim myArray(1 To fc.Count)
For Each f1 In fc
    myArray(i) = f1.Name
    i = i + 1
Next
' Sort the array.
Call mMonitor.dhBubbleSort(myArray)

' Calculate the positions and dimensions for the images.
Call GetDimensions(LeftVal, TopVal, imageW, imageH)

' Add the images to the PowerPoint presentation.
For j = (fc.Count - ImagesText.Text + 1) To fc.Count
    ImgFile = downloadFolder & "\" & myArray(j)
    myPres.Slides.Add K, ppLayoutBlank
    myPres.Slides.Item(K).Shapes.AddPicture ImgFile, True, True, _
    LeftVal, TopVal, imageW, imageH
    K = K + 1
    Next
    'Free up the FileSystemObject when done
    Set fs = Nothing
    Set f = Nothing
    Set fc = Nothing
    ' Configure the slide show properties and run the show
    For Each s In myPPT.ActivePresentation.Slides
        With s.SlideShowTransition
            .AdvanceOnTime = True
            .AdvanceTime = DurationText.Text
        End With
        Next
    With myPPT.ActivePresentation.SlideShowSettings
        .StartingSlide = 1
        .EndingSlide = ImagesText.Text
        .AdvanceMode = ppSlideShowUseSlideTimings
        .LoopUntilStopped = True
        .Run
    End With
    ' Delete the images when done creating the brief
    For i = 1 To fc.Count
        If fs.FileExists(downloadFolder & "\" & myArray(i))
        Then
            Set f = fs.DeleteFile(downloadFolder & "\" & myArray(i), True)
        End If
    Next
    End If
End Sub

'******************************************************************************
' The GetDimensions subroutine calculates the positions
' (Left, Top), and the dimensions (Height, Width)
' for the images.
' Parameters:
'   in/out: L - the Left value

190
Private Sub GetDimensions(L As Long, T As Long, W As Long, H As Long)

' Local variables declarations
Dim deltaX As Long
Dim deltaY As Long

DeltaX = myPPT.ActivePresentation.PageSetup.SlideWidth - WidthText.Text
DeltaY = myPPT.ActivePresentation.PageSetup.SlideHeight - HeightText.Text
If deltaX <= 0 Then
    L = 0
Else
    L = deltaX / 2
End If
If deltaY <= 0 Then
    T = 0
Else
    T = deltaY / 2
End If
W = WidthText.Text
H = HeightText.Text
If W > 720 Then W = 720
If H > 540 Then H = 540
End Sub

Private Sub lineInfo(searchStr As String, K As String, D As String, V As String)
Dim istart As Integer
Dim istop As Integer
istart = 1
istop = 0
istop = InStr(istart, searchString, "=", vbTextCompare)
' Get the key string
K = Mid(searchString, istart, istop - 1)
istart = istop + 1
istop = InStr(istart, searchString, ",", vbTextCompare)
' Get the directory string
If istop > istart Then
    D = Mid(searchString, istart, istop - istart)
    istart = istop + 1
    ' Get the location string
    V = Mid(searchString, istart)
Else
    D = Mid(searchString, istart)
    V = ""
End If
End Sub
'**********************************************************************
'
The downloadFile subroutine uses the OpenURL method to
download a file from the current open connection using
HTTP protocol.
Parameters:
in:
    URLStr - the URL for download the file from.
    saveFile - the filename for storing the
downloaded file on the client machine.

**********************************************************************
Private Sub downloadFile(URLStr As String, saveFile As String)
    Dim bData() As Byte  ' Data variable
    Dim intFile As Integer  ' FreeFile variable
    intFile = FreeFile()     ' Set intFile to an unused file.

    ' The result of the OpenURL method goes into the Byte
    ' array, and the Byte array is then saved to disk.
bData() = Inet1.OpenURL(URLStr, icByteArray)
Open saveFile For Binary Access Write As #intFile
Put #intFile, , bData()
Close #intFile
End Sub
'**********************************************************************
'
' Creating a folder on client machine.
' Parameter:
'   in: path - a qualify name of the folder being created.
'
'******************************************************************************
Private Sub createFolder(path As String)
    Dim fs, f
    Set fs = CreateObject("Scripting.FileSystemObject")
    If Not fs.FolderExists(path) Then
        Set f = fs.createFolder(path)
    End If
    Set fs = Nothing
    Set f = Nothing
End Sub
'******************************************************************************
'
' Deleting a folder on a client machine.
' Parameter:
'   in: path - a qualify name of the folder being deleted.
'
'******************************************************************************
Private Sub deleteFolder(path As String)
    Dim fs, f
    Set fs = CreateObject("Scripting.FileSystemObject")
    If fs.FolderExists(path) Then
        fs.deleteFolder path, True
    End If
    Set fs = Nothing
End Sub
'******************************************************************************
'
' Clean up all temporary folder created when exiting.
'
'******************************************************************************
Private Sub UserControl_Terminate()
    ' Delete the download folder
    deleteFolder downloadFolder
End sub

3. Object Components (Continuous Brief)

a) Global Variable Declarations

Attribute VB_Name = "GlobalDeclarations"
'###########################################################################
' # File: GlobalDeclarations.bas
' # Date          Author      History
' # 5/31/2000     Tam Tran    Created.

193
'#########################################################################################################
Option Explicit
'#########################################################################################################
'
The cfgInfo type is a record that stores the information
that read from the cvdata.cfg file (i.e., Key, Directory,
Virtual Directory, and the stamped date, which is the last
time the data is checked.)
'
'#########################################################################################################
Public Type cfgInfo
    key As String
    path As String
    vir_path As String
    stampdate As Date
End Type
'
'#########################################################################################################
'
'Global variables used by the ControllerConnector
'
'#########################################################################################################
Public gController As Controller   ' Reference to controller object
Public gControllerUseCount As Long  ' Global reference count
'
'#########################################################################################################
'
' Global variables used by the MonitorConnector
'
'#########################################################################################################
Public gMonitor As Monitor        'Reference to monitor object
Public gMonitorUseCount As Long    ' Global reference count
'
'#########################################################################################################
'
' Global variables used by the Monitor and Controller objects.
'
'#########################################################################################################
Public gCfgArray() As cfgInfo
    b) Timer

VERSION 5.00
Begin VB.Form Timing
    Caption =  "Form1"
ClientHeight  =  3195
ClientLeft    =   60
ClientTop  =   345
ClientWidth  =   4680
LinkTopic    = "Form1"
ScaleHeight  =  3195
ScaleWidth   =  4680
StartUpPosition =  3 'Windows Default
Begin VB.Timer Clock
    Left     =   2160
    Top      =   1200
End
End
Attribute VB_Name = "Timing"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
'########################################################################
' # File: Timing.frm
' # Date           Author      History
' # 5/31/2000       Tam Tran    Created.
'########################################################################
'########################################################################
' Set the clock interval to 5 second.
' The Monitor component uses this timer event to poll the
' storage directory for new data (images).
'########################################################################
Private Sub Form_Load()
    Clock.Interval = 5000
End Sub

c) Controller

VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior  = 0 'vbNone
    MTSTransactionMode   = 0 'NotAnMTSObject
END
Attribute VB_Name = "Controller"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True

195
Option Explicit

'********************************************************************************
' The Controller component uses this UpdateBrief event to notify the Continuous Brief wrapper (CBWrapper) for updating the brief.
' Event's parameters:
'    imageType: the type of images
'    imageLoc: the location where to find the images.
'
' The Glue component will raise the event to notify the Controller when it's done with storing data.
'
' The Monitor component will raise the event to notify the Controller when the new data come in.
' WithEvents causes the component(s) which raise the event(s)
' to run asynchronously.
' MonitorConnector component allows multiple connections to single Monitor object.
'
'********************************************************************************
Event UpdateBrief(imageType As String)

Public WithEvents mGlue As Glue
Attribute mGlue.VB_VarHelpID = -1
Private WithEvents mMonitor As Monitor ' Get Monitor events
Attribute mMonitor.VB_VarHelpID = -1
Private mMonitorConnector As MonitorConnector

'********************************************************************************
'
' Connect to the Monitor component
'
'********************************************************************************
Private Sub Class_Initialize()

    Set mMonitorConnector = New MonitorConnector
    Set mMonitor = mMonitorConnector.Monitor

End Sub
Receive the notification from the Monitor component
The Controller passes the information to the Glue component
for storing data to the database.
Event's parameter:
   DataType: the data (images) type

Private Sub mMonitor_NewData(DataType As String)
    Set mGlue = New Glue
    Call mGlue.StoreData(DataType)
End Sub

Receive the notification from the Glue component that
Asynchronous glue component is done.
The Controller notifies the CBWrapper(s) and passes the
information for the wrapper(s) to update the brief(s).
Event's parameter:
   DataType: the data (images) type

Private Sub mGlue_GlueDone(DataType As String)
    Set mGlue = Nothing ' Free the Glue object

   ' Notify the CBWrapper for updating the brief
   RaiseEvent UpdateBrief(DataType)
End Sub

Get all the image's filenames, which is being requested
from the CBWrapper, and make the makeFileList function
call to store the filenames to the CB_DATA.LST file.
Parameters:
in:
   ImageID - the image type
   fileCounts - the number of images requested.
   virtualDir - the virtual directory associated
   with the images' directory.
in/out:
   fileListName - a variable that holds the
   filename,
   which contains the list of images' filenames.
Public Sub GetImageInfo(ImageID As String, fileCounts As Integer, _
                      virtualDir As String, fileListName As String)

    Dim i As Integer
    For i = 1 To UBound(gCfgArray)
        If (StrComp(ImageID, gCfgArray(i).key, vbTextCompare) = 0) Then
        virtualDir = gCfgArray(i).vir_path
        fileListName = "CB_DATA.LST"
        Call makeFileList(fileCounts, gCfgArray(i).path, fileListName)
        End If
    Next
End Sub

'***********************************************************************************************************************************************
'
' Write all filenames from a specified directory to a file.
' This subroutine is called by GetImageInfo()
' Parameters:
'   in:
'       fileCounts - number of files is being read.
'       path - a specified directory for getting the filenames.
'       filename - the file used for storing the filenames.
'
'***********************************************************************************************************************************************
Private Sub makeFileList(fileCounts As Integer, path As String, _
                      filename As String)

    Dim fs, f, fc, f1, i, j, a
    Dim myCount As Integer
    Dim listfileStr As String
    Dim myArray() As String

    ' Create a FileSystemObject for manipulating the file system.
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.GetFolder(path)
    Set fc = f.Files
    myCount = fc.Count
    i = 1

    ' Store the name of the files to an array for sorting purpose
    ReDim myArray(1 To myCount)
    For Each f1 In fc

198
myArray(i) = f1.Name
i = i + 1

Next

' Sort the array
Call mMonitor.dhBubbleSort(myArray)
listfileStr = path & "\" & "CB_listfile"
createFolder listfileStr
Set a = fs.CreateTextFile(listfileStr & "\" & filename, True)
For j = (myCount - fileCounts + 1) To myCount
   a.WriteLine (myArray(j))
Next
a.Close
' Free up the objects, which are no longer be used.
Set fs = Nothing
Set f = Nothing
Set fc = Nothing
Set a = Nothing
End Sub

*************************************************************************
'
' This createFolder is used for creating a specified folder.
' Parameter:
'    in: path - the qualified name of the folder being created.
'
*************************************************************************
Private Sub createFolder(path As String)
   Dim fs, f
   Set fs = CreateObject("Scripting.FileSystemObject")
   If Not fs.FolderExists(path) Then
      Set f = fs.createFolder(path)
   End If
   Set fs = Nothing
   Set f = Nothing
End Sub

d) Controller Connector

VERSION 1.0 CLASS
BEGIN
   MultiUse = -1 'True
   Persistable = 0 'NotPersistable
   DataBindingBehavior = 0 'vbnNone
   DataSourceBehavior = 0 'vbnNone
   MTSTransactionMode = 0 'NotAnMTSObject
END
Attribute VB_Name = "ControllerConnector"
Attribute VB_GlobalNameSpace = False
Option Explicit

' This property allows other components to get reference to the Controller object.

Public Property Get Controller() As Controller
Set Controller = gController
End Property

Private Sub Class_Initialize()
    If gController Is Nothing Then
        Set gController = New Controller
    End If
    gControllerUseCount = gControllerUseCount + 1
End Sub

Private Sub Class_Terminate()
    If gControllerUseCount = 0 Then
        Set gController = Nothing
    End If
End Sub

e) Monitor

VERSION 1.0 CLASS
BEGIN

200
MultiUse = -1  'True
Persistable = 0  'NotPersistable
DataBindingBehavior = 0  'vbNone
DataSourceBehavior = 0  'vbNone
MTSTransactionMode = 0  'NotAnMTSObject
END
Attribute VB_Name = "Monitor"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
'############################################################
'#  File: Monitor.cls
'#  Date                Author         History
'#  5/31/2000            Tam Tran       Created.
'############################################################
Option Explicit
'########################################################################
' The VISStampDate, IRStampDate, and VAPORStampDate variables
' store the created date of the latest stored data.
'
' WithEvents causes the component(s) which raise the event(s)
' to run asynchronously.
' Event's parameter:
'     DataType: the data (images) type
'
' The Monitor component will raise the event to notify the
' Controller when the new data come in.
'########################################################################
Private VISStampDate As Date
Private IRStampDate As Date
Private VAPORStampDate As Date

Private mTiming As Timing
Private WithEvents mClock As Timer
Attribute mClock.VB_VarHelpID = -1

Event NewData(DataType As String)
'########################################################################
'
' The tasks done when a new Monitor object is created.
'
'########################################################################
Private Sub Class_Initialize()

' Start Monitor Timer and create instance of form
Set mTiming = New Timing
Load mTiming
' Connect timers' events to associated event procedures in Monitor
   Set mClock = mTiming.Clock

' Get the config information from the configuration file
Call GetConfig
End Sub

'********************************************************************
'
' The tasks done when the Monitor object is terminated.
'
'********************************************************************
Private Sub Class_Terminate()   ' Terminate Monitor

' Free up the timer object.
Set mClock = Nothing

' Unload and free up the form.
Unload mTiming
Set mTiming = Nothing
End Sub

'********************************************************************
'
' Process Timer Event.
' This timer event causes the Monitor to poll the storage directories for new data.
' The Monitor will raise the event(s) if it found a new data.
'
'********************************************************************
Private Sub mClock_Timer()
   Dim i As Integer
   For i = 1 To UBound(gCfgArray)
      If IsNewFile(gCfgArray(i).path, i) Then
         RaiseEvent NewData(gCfgArray(i).key)
      End If
   Next
End Sub

'********************************************************************
'
' The IsNewFile function is used to determine whether or not a new data exists.
' Parameters:
'   in: StrDir - the directory where to check for new data.
in: StampDate - the created date of the latest
data from the previous checked.
Return:
TRUE if there's new data, and FALSE otherwise.

**********************************************************************
Private Function IsNewFile(StrDir As String, arrayIndex As Integer) As Boolean
' Local variables declarations.
Dim fs, f, fc, f1, i
Dim myStamp As Date
Dim myArray() As String

' Create a FileSystemObject for manipulating the file
system.
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(StrDir)
Set fc = f.Files
i = 1

' Store the name of the files to an array for sorting
purpose.
ReDim myArray(1 To fc.Count)
For Each f1 In fc
    myArray(i) = f1.Name
    i = i + 1
Next

' Sort the array
Call dhBubbleSort(myArray)

' Check for new file based on the file's created date.
myStamp = fs.GetFile(StrDir & "\" &
myArray(fc.Count)).DateCreated
If (DateDiff("s", gCfgArray(arrayIndex).stampdate,
myStamp) <> 0) Then
    gCfgArray(arrayIndex).stampdate = myStamp
    IsNewFile = True
Else
    IsNewFile = False
End If

' Free up the objects, which are no longer be used.
Set fs = Nothing
Set f = Nothing
Set fc = Nothing
End Function

**********************************************************************
' Standard bubblesort.
'DON'T USE THIS unless you know the data is already
' almost sorted! It's incredibly slow for
' randomly sorted data.

' There are many variants on this algorithm.
' There may even be better ones than this.
' But it's not even going to win any
' speed prizes for random sorts.

' From "Visual Basic Language Developer's Handbook"
' by Ken Getz and Mike Gilbert
' Copyright 2000; Sybex, Inc. All rights reserved.

' In:
' varItems:
' Array of items to be sorted.
' Out:
' VarItems will be sorted.
'******************************************************************************
Public Sub dhBubbleSort(varItems As Variant)

    Dim blnSorted As Boolean
    Dim lngI As Long
    Dim lngJ As Long
    Dim lngItems As Long
    Dim varTemp As Variant
    Dim lngLBound As Long

    lngItems = UBound(varItems)
lngLBound = LBound(varItems)

    ' Set lngI one lower than the lower bound.
lngI = lngLBound - 1
Do While (lngI < lngItems) And Not blnSorted
    blnSorted = True
    lngI = lngI + 1
    For lngJ = lngLBound To lngItems - lngI
        If varItems(lngJ) > varItems(lngJ + 1) Then
            varTemp = varItems(lngJ)
            varItems(lngJ) = varItems(lngJ + 1)
            varItems(lngJ + 1) = varTemp
            blnSorted = False
        End If
    Next lngJ
Loop
End Sub
'******************************************************************************
' The lineInfo subroutine parses a line input from the configuration file (cbdata.cfg). It separates information of the key, the directory, and the virtual directory from the line string input.
' Parameters:
' in:
' searchStr - the string is being parsed.
' in/out:
' K - a variable that holds the key string
' D - a variable that holds the directory string
' V - a variable that holds the virtual directory string
'
'**********************************************************************************************
Private Sub lineInfo(searchStr As String, K As String, D As String, V As String)
    Dim istart As Integer
    Dim istop As Integer

    istart = 1
    istop = 0
    istop = InStr(istart, searchStr, "="#, vbTextCompare) ' Get the key string
    K = Mid(searchStr, istart, istop - 1)
    istart = istop + 1
    istop = InStr(istart, searchStr, "/", vbTextCompare) ' Get the directory string
    If istop > istart Then
        D = Mid(searchStr, istart, istop - istart)
        istart = istop + 1
        ' Get the location string
        V = Mid(searchStr, istart)
    Else
        D = Mid(searchStr, istart)
        V = ""
    End If
End Sub
'**********************************************************************************************

' The GetDateArrayIndex function returns an index of the dateArray, where the specified image type (ID) is stored.
'
'***************************************************************************************
Public Function GetArrayIndex(key As String) As Integer
    Dim tmpInfo As cfgInfo
    Dim bFound As Boolean
    Dim i As Integer
    bFound = False
    i = 1

    205
Do While Not bFound
    tmpInfo = gCfgArray(i)
    If (StrComp(tmpInfo.key, key) = 0) Then
        GetArrayIndex = i
        bFound = True
    End If
    i = i + 1
Loop
End Function

'******************************************************************************
'
The GetConfig subroutine reads information stored in
' the configuration file, and adds them to the link list.
'
'******************************************************************************
Private Sub GetConfig()

    Dim cfgpath As String
    Dim inputStr As String
    Dim keyStr As String
    Dim dirStr As String
    Dim virDirStr As String
    Dim intFile As Integer
    Dim tmpInfo As cfgInfo

    ' Initialize the size the gCfgArray
    ReDim gCfgArray(0)
    ' Get the path for the configuration file
    cfgpath = Environ("CB_HOME") & "/cbdata.cfg"

    ' Store the configured info to the array
    intFile = FreeFile()
    Open cfgpath For Input As #intFile
    Do While Not EOF(intFile)
        Line Input #intFile, inputStr
        Call lineInfo(inputStr, keyStr, dirStr, virDirStr)
        With tmpInfo
            .key = keyStr
            .path = dirStr
            .vir_path = virDirStr
            .stampdate = -1 ' initialize the date to
        End With
        ReDim Preserve gCfgArray(UBound(gCfgArray) + 1)
        gCfgArray(UBound(gCfgArray)) = tmpInfo
    Loop
    Close #intFile
End Sub
f) Monitor Connector

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1 'True
  Persistable = 0 'NotPersistable
  DataBindingBehavior = 0 'vbNone
  DataSourceBehavior = 0 'vbNone
  MTSTransactionMode = 0 'NotAnMTSObject
END
Attribute VB_Name = "MonitorConnector"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
'#########################################################################
' #  File: MonitorConnector.cls
' #  Date          Author      History
' #  5/31/2000      Tam Tran    Created.
'#########################################################################

Option Explicit
'#########################################################################
'    This property allows other components to get reference
'    to the Monitor object.
'    ########################################################################
Public Property Get Monitor() As Monitor
  Set Monitor = gMonitor
End Property
'#########################################################################
'    Initialize Monitor and reference count.
'    ########################################################################
Private Sub Class_Initialize()
  If gMonitor Is Nothing Then
    ' Creates a new link list for holding the configuration info.
    Set gMonitor = New Monitor
  End If
  gMonitorUseCount = gMonitorUseCount + 1
End Sub
'#########################################################################
'    Terminate Monitor when reference count = 0
'    ########################################################################

Private Sub Class_Terminate()
    gMonitorUseCount = gMonitorUseCount - 1
    If gMonitorUseCount = 0 Then
        Set gMonitor = Nothing
    End If
End Sub

g) Glue

VERSION 1.0 CLASS
BEGIN
    MultiUse = -1 'True
    Persistable = 0 'NotPersistable
    DataBindingBehavior = 0 'vbNone
    DataSourceBehavior = 0 'vbNone
    MTSTransactionMode = 0 'NotAnMTSObject
END

Attribute VB_Name = "Glue"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True

'********************************************************************************
' # File: Glue.cls
' # Date Author History
' # 5/31/2000 Tam Tran Created.
'********************************************************************************

Option Explicit

'********************************************************************************
' The Glue component uses this event to notify the Controller when done with its task.
' Event's parameter:
'   DataType: the data (images) type.
'********************************************************************************

Event GlueDone(DataType As String)

'********************************************************************************
' Notify the Controller when done storing data.
'********************************************************************************

Public Sub StoreData(DataType As String) ' Start glue task
    ' <Insert glue task here>
    ...
    RaiseEvent GlueDone(DataType)
End Sub

208
LIST OF REFERENCES


INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.................. 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library.................................. 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, California 93943-5101

3. Chairman, Code CS................................... 1
   Naval Postgraduate School
   Monterey, California 93943-5100

4. Dr. Luqi, CS/Lq...................................... 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5100

5. Dr. Mantak Shing, Code CS/Sh........................ 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5100

6. Karen M. Gee........................................ 1
   12560 Salmon River Rd.
   San Diego, California 92129

213