Spring 2015

# A Common Knowledge Engineering Framework for Data Assimilation, Correlation, and Extrapolation (DACE)

Edward P. Weaver
*Old Dominion University*

# A COMMON KNOWLEDGE ENGINEERING FRAMEWORK FOR DATA ASSIMILATION, CORRELATION, AND EXTRAPOLATION (DACE)

by

Edward P. Weaver
B.S. May 2001, Old Dominion University
M.S. May 2007, George Washington University

A Doctoral Project Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF ENGINEERING

ENGINEERING MANAGEMENT AND SYSTEMS ENGINEERING

OLD DOMINION UNIVERSITY
May 2015

Approved by:

_____
Adrian Gheorghe (Director)

_____
Charlie Daniels (Member)

_____
Holly Handley (Member)

_____
Roderick Barnes (Member)

## ABSTRACT

A COMMON KNOWLEDGE ENGINEERING FRAMEWORK FOR DATA ASSIMILATION, CORRELATION, AND EXTRAPOLATION (DACE)

Edward P. Weaver
Old Dominion University, 2015
Director: Dr. Adrian Gheorghe

The Common Knowledge Engineering Framework for Data Assimilation, Correlation, and Extrapolation (DACE) project is focused on providing a software centric general framework for advanced processing and analysis of data. This translates to researchers, scientists, engineers, and system architects not having to program a new application but rather to define the system configuration, process, and processing that is needed to perform a specific functionality. This makes the limitation of the application the end users ability to fully define the functional requirements and setup the framework accordingly.

This doctoral project will provide the details to the system definition, standards, metrics, schedule, and evaluation that were utilized in the performance of this project. The project's framework allows multiple analysis methods to be utilized either individually or concurrently depending on the end user's configuration. The architecture will not provide limitations on what can be done. It will allow the end user to configure and define the analysis method to use.

I would like to dedicate this Doctoral Project to Meghan, Shane, Sarah and everyone that helped me on this endeavor. Without their understanding, support, and encouragement, this journey would not have been possible.

# ACKNOWLEDGMENTS

I would like to express my appreciation to my advisor, Dr. Adrian Gheorghe, for his support, guidance, and encouragement. His dedication and passion in the advancement in this topic's development have been an inspiration to me.

I would also like to thank my committee members, Dr. Charlie Daniels, Dr. Holly Handley, and Dr. Roderick Barnes, for their insights, suggestions, and comments to improve the quality of this project.

# NOMENCLATURE

| | |
|---|---|
| *ADO* | ActiveX Data Objects |
| *ASP* | Active Server Pages |
| *Atern* | Arctic Tern |
| *CLR* | Common Language Runtime |
| *COTS* | Commercial Off-The-Shelf |
| *CRC* | Cyclic Redundancy Check |
| *CPU* | Central Processing Unit |
| *D* | Difficulty |
| *DACE* | Data Assimilation, Correlation, and Extrapolation |
| *DCU* | DACE Configuration Utility |
| *DLL* | Dynamically Linked Library |
| *Dman* | Data Manager |
| *DOD* | Department of Defense |
| *DSDM* | Dynamic Systems Development Index |
| *DSM* | DACE System Manager |
| *DSQI* | Dynamic Structure Quality Index |
| *E* | Effort |
| ECMA | European Computer Manufacturers Association |
| *exe* | Executable |
| *FTP* | File Transfer Protocol |
| *GOTS* | Government Off-The-Shelf |
| *GUI* | Graphical User Interface |

| | |
|---|---|
| *HTTP* | Hypertext Transfer Protocol |
| *IP* | Internal Protocol |
| *IRAD* | Internal Research and Development |
| *LAN* | Local Area Network |
| *LISI* | Levels of Information Systems Interoperability |
| *ms* | Milliseconds |
| $n$ | Program Vocabulary |
| $N$ | Program Length |
| $n_1$ | Number of distinct operators |
| $N_1$ | Total number of operators |
| $n_2$ | Number of distinct operands |
| $N_2$ | Total number of operands |
| *NIPRNet* | Nonsecure Internet Protocol Router Network |
| *NRE* | Non-Recurring Engineering |
| *PEMs* | Process Entity Modules |
| *Pman* | Presentation Manager |
| *P&A* | Procedure & Analyses |
| *R&D* | Research & Development |
| *RTSP* | Real Time Streaming Protocol |
| *SIPRNet* | Secret Internet Protocol Router Network |
| *s* | Seconds |
| *SME* | Subject Matter Expert |
| $T$ | Time in Seconds |

| | |
|---|---|
| *TCP* | Transmission Control Protocol |
| *UDP* | User Datagram Protocol |
| *V* | Volume |
| *VB* | Visual Basic |
| *WCF* | Windows Communication Foundation |

# TABLE OF CONTENTS

Page

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

## INTRODUCTION

In today's technically complex world, there is a focus on system interaction and data integration. Historically, technology was designed to perform specific tasks given certain situations. These designs were utilized in conjunction with other designs to provide situational awareness to accomplish a goal. The information from each of the components or devices was gathered independent of one another and the data were analyzed in the effort to paint a heuristic picture of the environment. This process required that the resources gathering and analyzing the data had to thoroughly understand the components and what they do, as well as, have a detailed understanding of information that it provided. The process also geared heavily upon the ability of the analyzer to take in all of the information provided and be able to build an overall situational understanding.

This generated a focus more to a system level, where, in theory, the components interact at the machine level and data is correlated and evaluated based on a defined set of constructs. With this focus came integration issues which translated to increased costs and schedules due to the considerable effort needed to integrate components that were not designed in a fashion that would allow them to work in such a manner. The situation is further compounded when systems are integrated with both old and new technologies. The purpose of this project is an attempt to provide a common software framework to address data assimilation, correlation, and extrapolation. The intent is for this tool to become a foundation application for complex system design and integration efforts.

## 1.1 Project Problem Description

Over the past 20 years, system designs have become increasingly complex. The focus on internal system boundaries, as well as, external system boundaries has become more of a focal point. This is due to the level of information that is being distributed within the systems. Custom software development efforts, to connect within these systems, have increased dramatically and can be seen as one of the development tasks in almost any project. This scenario creates a common system development efficiency problem, meaning that the process of developing a system requires development teams to continue to develop custom interfacing software over and over, costing time and money.

In the federal sector, millions of dollars are spent each year by the government on projects that are focused on the integration of components into systems, and systems into system of systems. This focus is not only on military technology projects but on financial and business process systems as well. One can find multiple parallel efforts in each of the three focus areas listed above. What if the problem for the three focus areas was actually the same problem, viewed from a different perspective?

By moving the view perspective further from the system of observation, to a higher level to see more of the picture and provide more generality, there are considerable similarities. Each area requires an input, performs an operation, and produces an output. This is a very simplistic view for these areas and very ideal. If this model was accurate, then why do both the federal and commercial markets continue to develop integration modules that only address a specific system implementation? The answer lies in the thought process that it is easier, faster, and cheaper for a customer to focus only on the current system being developed. Most customers define a fixed set of

requirements to address a specific functional need. This provides a method to support the developers in the ability to complete designs, hopefully, within schedule and budget. Yet, what is actually missed in this process is the fact that there may be 20 other projects in development at the same time doing very similar types of work. Each project is spending a considerable amount of time and money, compounding the total effort that is being placed in system integration efforts. To address the problem, a Common Knowledge Engineering Framework for Data Assimilation, Correlation, and Extrapolation (DACE) is proposed.

## 1.2 Overview

To architect a system that can interconnect with various types of systems, the architecture must be designed in such a way that core functionality is abstracted. This means the application has been designed and developed with various layers to allow for flexibility in its capabilities. The proposed framework for DACE consists of three main layers: a Presentation, a Business, and a Data Service Layer (see Figure 1).

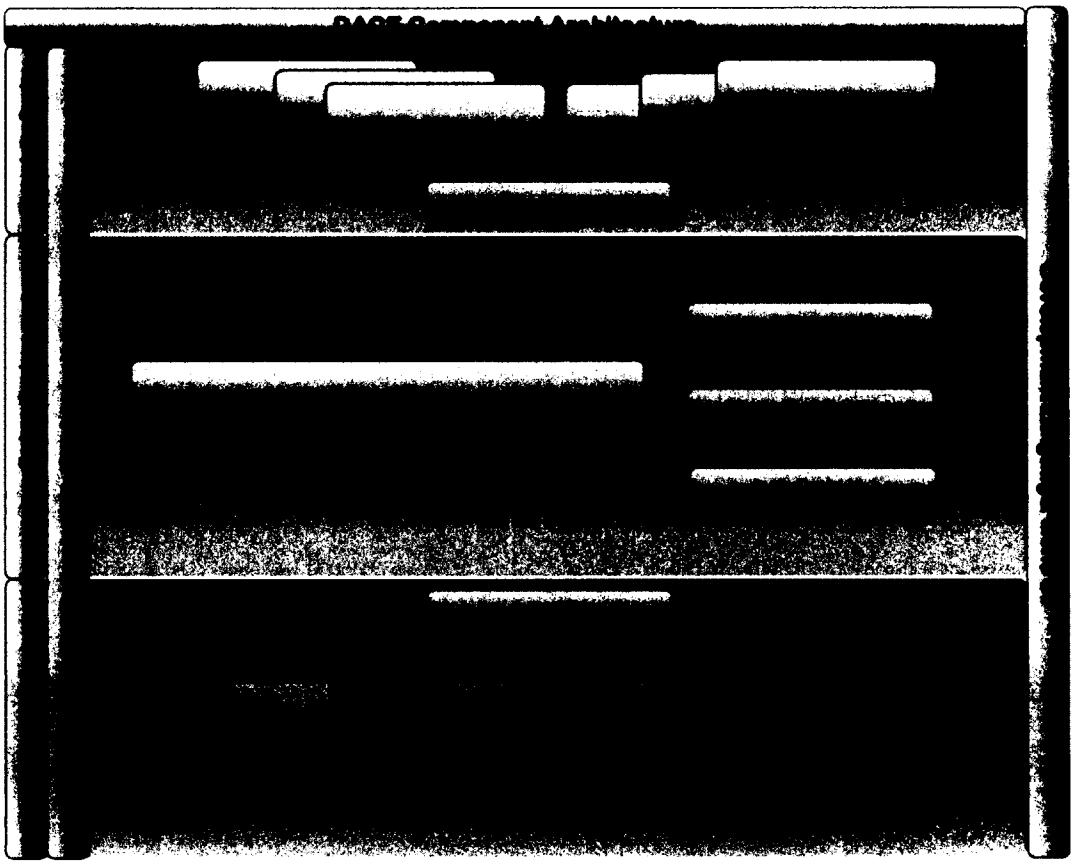


**Figure 1. DACE High Level Architectural Framework**

1.2.1 Presentation Layer

The Presentation Layer consists of user interfaces or Graphical User Interface (GUI) and a Presentation Manager (Pman). This comprises the front end of the application and allows the user to enter in data, configure application specific settings, run an analysis, analyze the results, and output the results in another form such as digital or hard copy. This layer only directly interacts with the Business Layer through the Pman. If data need to be retrieved from the Data Service Layer, the request is passed to the Business Layer for processing.

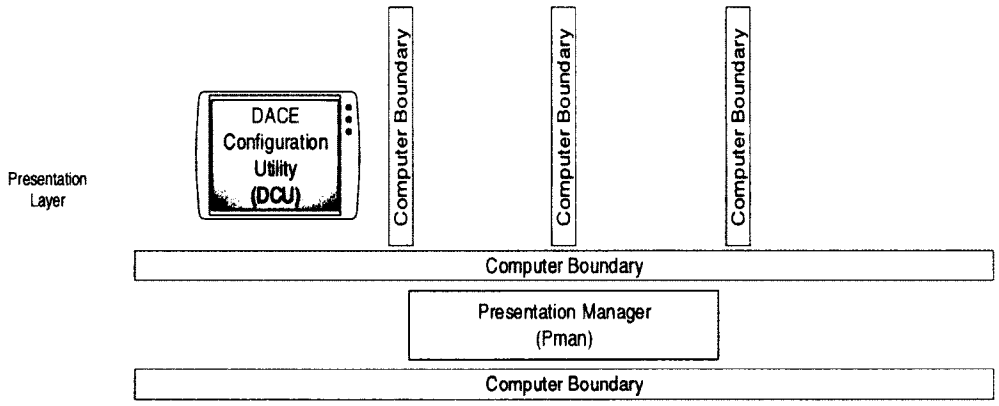



**Figure 2. Presentation Layer**

The GUI in the Presentation Layer (see Figure 2), when an analysis is configured, passes the analysis information to the Core Framework (Business Service Layer) so that an analysis can be performed with respect to the chosen methodology. As the analysis is running, modules in the Presentation Layer have the ability to receive status information from the Core Framework on its state, progress, and any issues it may be encountering.

The DACE Configuration Utility (DCU) is a tool that encapsulates utilities that support a defined system configuration allowing the ability to update and monitor system specific information. The tool allows for manual configuration of a system and was developed to support the development phase for configuration and testing purposes.

### 1.2.2 Business Service Layer – Core Framework

The Business Service Layer is the heart of the architecture which contains the Core Framework (see Figure 3).

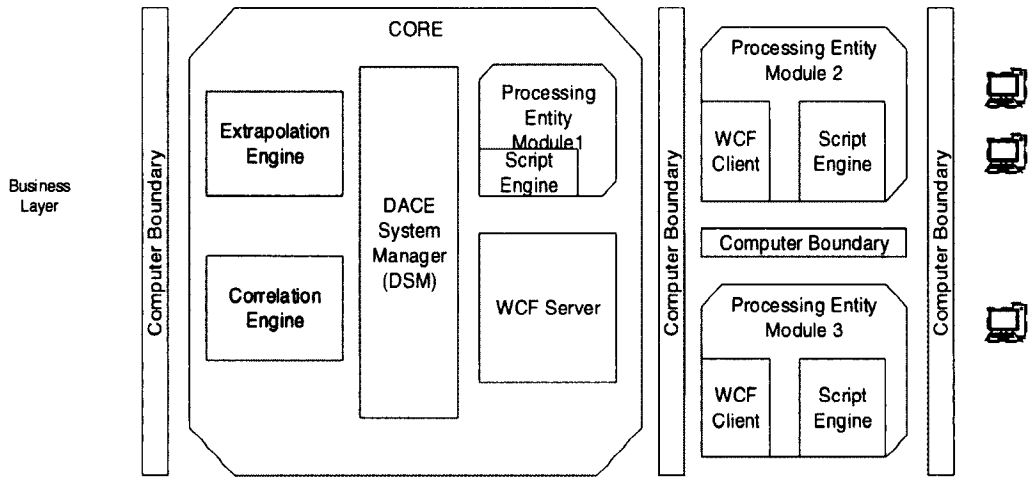



**Figure 3. Business Service Layer**

This layer is responsible for building the system architecture from the configuration defined in the Presentation Layer, execution of install packages, and distribution of operations and management of the system. The main components within this layer are the Core and the Process Entity Modules (PEMs).

The Core provides the main operating processes. It is comprised of a DACE System Manager (DSM), a Windows Communication Foundation (WCF) server, a PEMs, a Correlation Engine, and an Extrapolation Engine. These components provide the infrastructure that supports the overall capability of the system.

The DSM manages the overall system. It provides mechanisms to control the configuration, monitoring, and controlling of a defined DACE architecture implementation. As the system manager, it coordinates module execution, updating, health monitoring, and controls all of the PEMs within a systems design state. This module also contains and coordinates operations with both the Correlation and Extrapolation engines.

The WCF server controls all system component communications. It provides both internal and external network communication mechanisms. These mechanisms allow for file transfers from the Pman to PEMs, data transfers with PEMs, and mechanisms for PEMs state and operational control.

A PEMs is a component that performs a defined operation. It consists of operations and procedures that have been defined by an end user. A simple example would be a process where data are retrieved from a sensor, processed using mathematical equations, and produces a result. If this result has been marked as an output parameter from the PEMs to the Core, then this information is transferred to the WCF Server. In the Core, a PEMs component will always exist to support the DSM in processing data for system execution. As indicated in Figure 3, additional PEMs can reside outside of the Core. Additional PEMs can exist based on the end user's configuration preference. They can reside on the same machine as the Core and on a separate machine.

The Correlation Engine provides a mechanism for data association. If a system has various sources of data, this engine can perform higher level analysis to assist in determining links and relationships between system data. This engine's process is defined within the Presentation Layer applications. Since the end user is responsible for the functionality of this engine, the process is only as good as the configuration provided.

The Extrapolation Engine provides a mechanism for computed data to be forecasted beyond its current state. This can be done by analyzing the system's computed results and based on the end user's defined parameters, project future results. This engine's process is defined within the Presentation Layer applications. Since the end user

is responsible for the functionality of this engine, the process is only as good as the configuration provided.

### 1.2.3 Data Service Layer

The Data Service Layer is the primary data storage mechanism. It is comprised of a Data Manager (Dman) and data storage containers (see Figure 4). The Dman provides a common interface mechanism for the Core Framework to interact with the Data Service Layer. It resides in the Presentation Service Layer and provides the means to store and retrieve data that have been selected within the system configuration to be warehoused. The common interface implementation provides a set of standard operations that can be executed with no dependency on the underlining storage mechanism. This means that the type of data warehouse implemented, whether a database, a flat file, or a spreadsheet, has no bearing to a calling process. The Dman takes care of all the data warehousing interactions and the translation of the data to the specific formatting and processing requirements for that type of storage container.

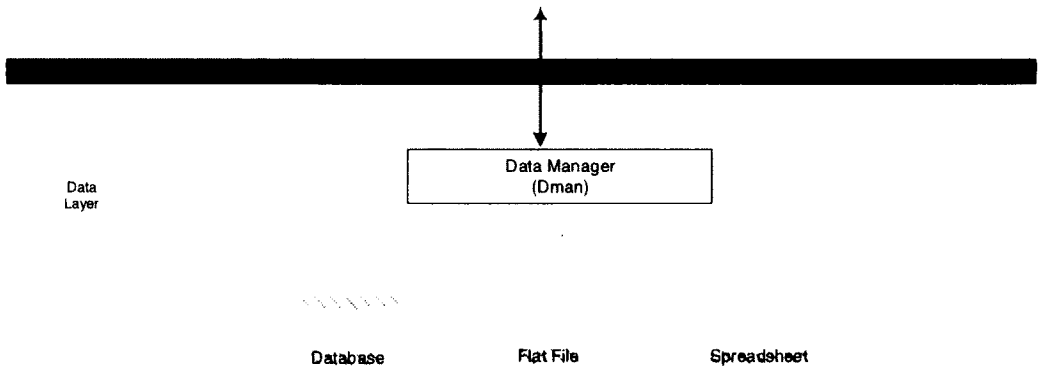



**Figure 4. Data Service Layer**

### 1.2.4 Significance and Impact

The framework would have significant impact to system development efforts. This is due to the framework being developed in a modular and configurable architecture, and its ability to support the three main focus areas of data assimilation, correlation, and extrapolation in one cohesive application. The focus areas would also be able to be deployed in a standalone fashion as well. This ability allows the framework to be deployed in a variety of situations.

When it comes to complex system designs, there is always a gap that needs to be addressed for component integration. Often, components have different types of interfaces and communication mechanisms. The DACE framework would be able to be the "glue" or translation element for these components. This would be achievable without writing custom code, which is essentially the current process, by defining the components communication schema and the information that needs to be translated. This is paramount because the system developer may not have the ability to modify the components within the system without providing Non-Recurring Engineering (NRE) funding to the original manufacturer.

There is often a need to have the ability to analyze data, whether they are live or historical in nature, utilizing multiple methods and performing cross correlation to the results of each analysis method to provide a final output model. The DACE framework is designed to support this type of analysis fusion. This provides value to both academic and industry, by providing a common mechanism where analysis methodology or methodologies is defined by the end user for a specific focus. For example, a system needs to be analyzed for an Internal Research and Development (IRAD) project to

determine the impact to a company's portfolio. It has been determined that the following methods would be utilized for this effort: Failure Modes and Effects Analysis (Reliability), Fault Tree Analysis (Reliability), Life-cycle Analysis (also known as Life-Cycle Assessment), and Value Chain Analysis (Firm Level). Within the DACE framework each of these methods could be analyzed in their own separate process. The results of each analysis would then, based on the user's defined cross relationship rules, be correlated together to provide a final solution set to the defined scenario (see Figure 5 for representation).

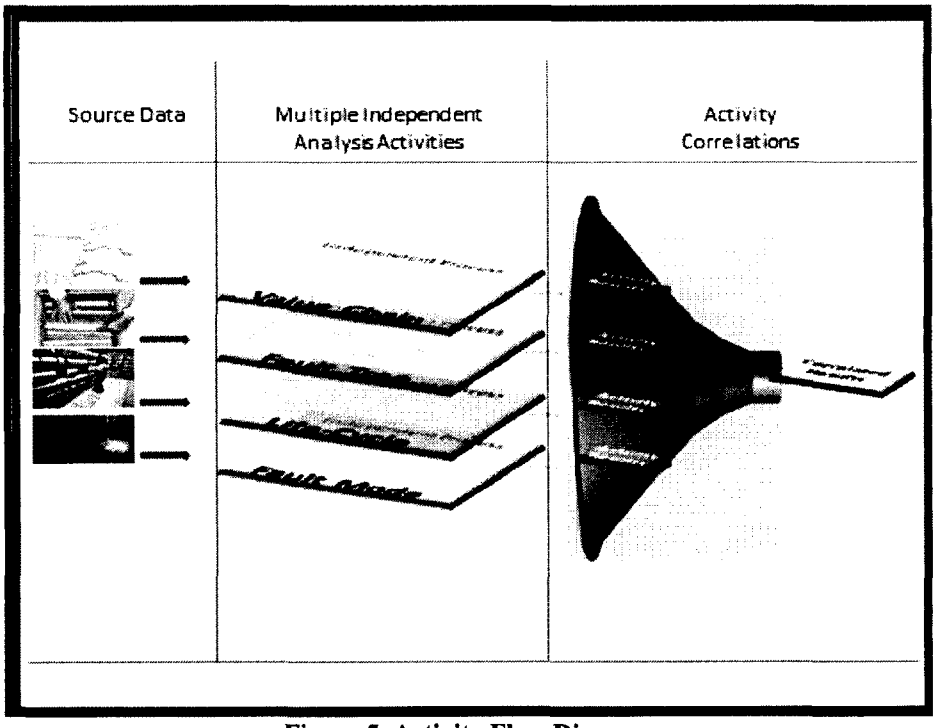



**Figure 5. Activity Flow Diagram**

These situational application areas provide value to both the engineering management and systems engineering disciplines. With the defined framework, significant efficiencies could be gained in both schedule and cost of a projects effort.

Engineering Management, from a federal market perspective, has a heavy focus on the cost of an overall product development effort. The utilization of a common application to integrate a system provides a significant cost savings relative to schedule and/or to resource allocation costs of developing a custom application to interface components. The cost savings gained could even be across multiple projects. This savings could translate to quicker break-even scenarios allowing companies to more quickly profit on their designs. If the project can leverage a reduction in schedule by eliminating the need for custom interfaces, the result could be a faster time to market.

System engineers will also gain significant benefit on projects that require quick prototypes, feasibility studies, complex analysis methodology, and custom analysis methods. This could possibly provide them additional justification to get buy-in for internal research and development projects based on a common tool which does not require a long programming cycle or purchase of a different tool for each IRAD exercise.

# CHAPTER 2

# DACE DESIGN DETAILS

This chapter provides details to the underlining architecture and methodology of the DACE design. It offers information on the design details, constraints, and the overall implementation process.

## 2.1 Overview of Technical Details

The reality is the DACE system is very complex and large in scope. To define such a system required definition and analysis of the system's requirements. Upon analysis, technologies and system design considerations were identified. This information further required the grouping and definition of a two tier classification scheme. The first tier construct, is the classification of the inclusion of the 12 variable elements of application configuration control or freedom. The second tier construct is the breakout of the pinnacle sections of the architecture to align with the Three-Tier software architecture model. These sections are the Presentation, the Business, and the Data Service Layers (see Figure 6).

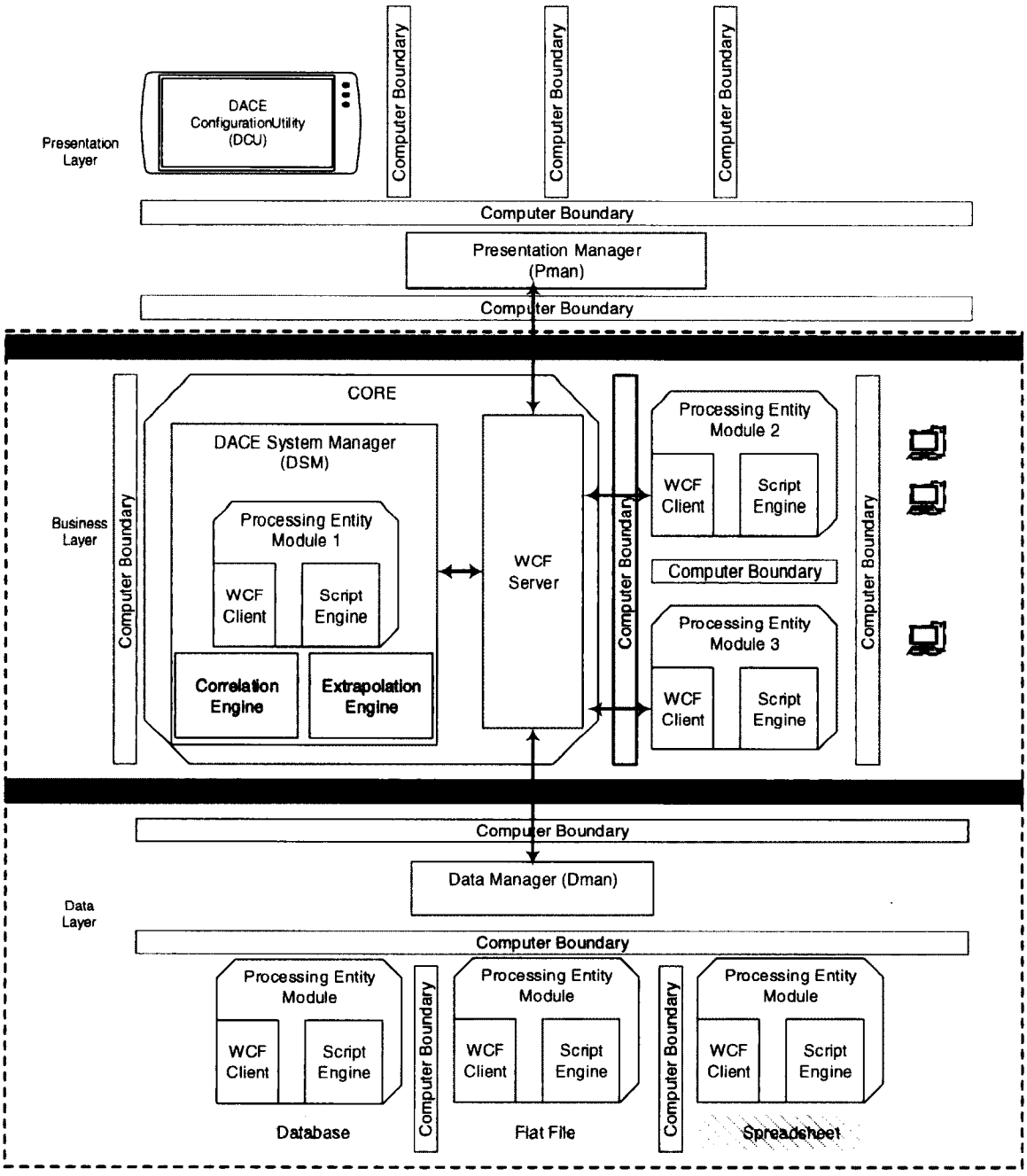



**Figure 6. Detailed DACE Architectural Framework**

The choice of using a multi-tier application with the integration of the 12 variable elements (see Figure 7) provides the optimal flexibility for both local and distributed

systems. This provides significant benefit to support both the system analysis, as well as the design development.

| 12 Variable Elements |
|---|
| Model |
| Process |
| Processing |
| Application |
| Data |
| Correlation |
| Extrapolation |
| Storage |
| Health |
| Metrics |
| Security |
| Reliability |

**Figure 7. 12 Variable Elements**

## 2.2 Twelve Variable Elements

The 12 variable elements definitions are a way to quantify major features of the system that provide dynamic configuration capability. To fully understand the importance of defining these elements, we must first quantify their intended function within the design. The variable elements defined are:

1. Model: Local vs. Distributed
2. Process: Work flow logic in Single Thread vs. Multiple Thread
3. Processing: Single Core Processing vs. Multi Core Processing
4. Application: Executable vs. Service Application
5. Data: Real-time Data vs. Captured Data

6. Correlation: Correlated Analysis (Fusion) vs. Uncorrelated
7. Extrapolation: Active vs. Inactive
8. Storage: Single vs. Multiple Storage Containers
9. Health: System Configuration Health monitoring: Active vs. Inactive
10. Metrics: System Operational metrics: Active vs. Inactive
11. Security: Access control, encryption, redundancy check
12. Reliability: Message Delivery, Redundancy

The "Model" element is defined as the classification of the system components locality. There are two main capabilities, local and distributed. The local option implies that all of the layers are physically located on the same machine. This means that one computer is running the full DACE application which consists of the Presentation, Business, and Data Layers. While the distributed option implies that at least one of the layers (Presentation or Data Layer) or a processing entity within the Business Layer is on at least one other machine. There could be multiple Presentation Layers and Data Layers but there will always only be one Business Layer which could contain multiple processing entities.

The "Process" element is defined as the classification of how the system processes programmed instructions. There are two main capabilities, single threaded or multi-threaded. These are called Procedure and Analysis Threads in the DACE framework for clarification. The system could be single threaded, meaning the instructions are linear, one operation at a time in order. Within DACE, this would effectively apply to the Business Layer where the processing occurs. For DACE to perform as a single threaded application, all processing would be contained within one

processing entity and be performed in order. This configuration within DACE provides "State Machine" functionality. The second option is that the system could be multi-threaded, meaning that selections of instructions are placed in groups to be processed separately, as if they were isolated. The operating system would time slice between each thread so that each could perform its intended functionality. In a single core processor, these threads are not concurrent but essentially thought of as operating in that state. Examples of these configurations will be provided in a later section.

The "Processing" element is defined as the classification of the system's utilization of the computers processing capability. Processors in computers can be single, dual, or quad core, meaning that there can be multiple processing units on a single chip. A single core processor has only one processing unit on it and, when using threads, can only process one thread at a time. A dual core processor has two processing units and, when using threads, can be running two threads concurrently. A quad core processor has four processing units and, when using threads, can run four threads concurrently providing greater efficiencies in processing. It should be noted that when using multi-threading coding techniques with multi-core processors, great care needs to be applied to guarantee there are no data contingencies between tasks. If there are, this could lock up the designed system in question.

The "Application" element is defined as the classification of the system's method of instantiation. There are three main types of instantiation, an executable (.exe), a service (also called a damon in UNIX), or a dynamically loaded assembly. Executable applications are general applications that reside on machines and typically must be selected to execute. Services are applications that run in the background and are

instantiated when the system starts and typically execute whenever the computer is running. Dynamically loaded assemblies are compiled code that is loaded at runtime without any direct reference within a running application. This allows for applications to be extended without having to recompile.

The "Data" element is defined as the classification of the system's collection of input data. There are two main types, real-time and captured data. Real-time data are considered live data, meaning a sensor or some device is collecting and providing these data for usage. Captured data are considered data that have been recorded and stored in a digital format such as a flat file, an excel spreadsheet, or even a database.

The "Correlation" element is defined as the classification of the system's ability to perform correlation on data, based on predefined constraints for the dataset under analysis. There are two options, either correlated or uncorrelated. The correlated option tells the system to perform a specific correlation based on either incoming data or data that have already been processed. The data to use and the procedure to perform the operations are inputted by the subject matter expert (SME) or the end user. This process is executed within the Correlation Component of the Processing Entity within the Business Layer. The uncorrelated option tells the system that there is no correlation to perform on the individual Processing Entity in question.

The "Extrapolation" element is defined as the classification of the system's ability to take processed data and perform estimation beyond the original interval of observation. If this functionality is set "Active", then the Process Entity in question will execute a predefined set of data and procedures, which have been entered by the SME. An example would be to predict the location of a vehicle in a failed GPS state. By using other sources

of sensor information i.e., a last known GPS state, and navigation algorithms, one can predict the current location accurately over a certain length of time. The Extrapolation component is located within each Process Entity that has configured this option to "Active" within the Business Layer.

The "Storage" element is defined as the classification of the system's capability to store and retrieve data from various data storage containers. The application has the ability to set multiple data storage containers or a single storage container for a given system.

The "Health" element is defined as the classification of the system's ability to provide system health monitoring. This option can either be set to "Active" or "Inactive". If set to "Active," then within each of the Presentation, Business, and Data Layers, active system monitoring will occur. The system state information is then collected at the Business Layer and can be distributed for status applications within the Presentation Layer stored in the Data Layer containers.

The "Metric" element is defined as the classification of the system's ability to collect and quantify performance within each of the layers. Metrics such as processing time, memory allocations, and communication throughput are examples of items that are collected to determine system metrics. Since the design is very flexible, and a system can be configured in many ways to perform the same functionality, the ability to capture and quantify system performance metrics provides a way to determine the most optimal configuration.

## 2.3 Three-Tier Architecture

The definition of the 12 variable elements of freedom drove the determination of a highly flexible architecture. To be able to fulfill the requirements of the 12 variable elements classification, a "Three-Tier Architecture" was selected. The benefits of this architecture are in its scalability, higher level of security, faster execution, and the ability to allow client side applications to be less complex. These features align with the DACE model; however this also increases the level of complexity within the DACE framework implementation. This architecture is comprised of a Presentation Layer, a Business Layer and a Data Layer.

### 2.3.1 Presentation Service Layer

The Presentation Service Layer conceptually entails the user interfaces for human interaction with the system, as well as, the Presentation Manager. A base GUI application called the DACE Configuration Utility provides the full system interaction functionality.

#### 2.3.1.1 DACE Configuration Utility

The DCU is a tool that has been designed to support the system configuration, updating, and monitoring. The DCU interacts with the system via a Windows Communication Foundation client module to the Pman. The intention of this tool is to support the design development phase for configuration and testing purposes, as well as, providing the base application user interface. It is comprised of four configuration objects, two functional objects, and two operational objects (see Figure 8).

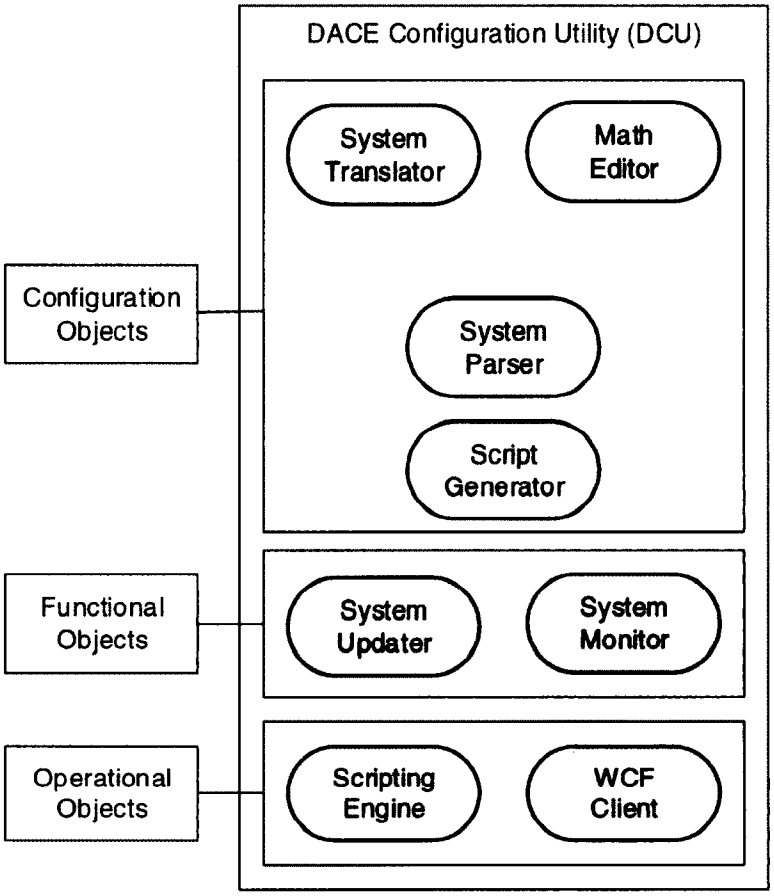



**Figure 8. DACE Configuration Utility (DCU)**

The tool allows for manual configuration of a system. The tool takes a defined system configuration which captures the following parameters:

1. Definition of client access
    a. Number of clients
        i. Each Client
            1. Definition of Client ID
            2. Definition of Client IP
            3. Definition of Client Operating System

4. Definition of Client Access Level
5. Definition of Data Encryption
6. Definition of Data Cyclic Redundancy Check (CRC)
7. Definition of Performance Metrics Reporting

2. Definition of number of components (Each component)
    a. Each component
        i. Definition of Component ID
        ii. Definition of Components IP
        iii. Definition of Components Operating System
        iv. Definition of Execution Process (Threads)
        v. Definition of Processing (CPU Cores)
        vi. Definition of components operations
            1. Definition of input data
                a. Definition of data source
                b. Definition of data format
                c. Definition of data types
            2. Definition of operations on data
                a. Definition of process
                b. Definition of internal variables
            3. Definition of output data
                a. Definition of distribution rights
                b. Definition of encryption option
                c. Definition of storage options

4. Definition of performance metrics

3. Definition of correlation operations
    a. Definition of input data
        i. Definition of data source
        ii. Definition of data format
        iii. Definition of data types
    b. Definition of operations on data
        i. Definition of process
        ii. Definition of internal variables
    c. Definition of output data
        i. Definition of distribution rights
        ii. Definition of encryption option
        iii. Definition of storage options
    d. Definition of performance metrics
4. Definition of extrapolation operations
    a. Definition of input data
        i. Definition of data source
        ii. Definition of data format
        iii. Definition of data types
    b. Definition of operations on data
        i. Definition of process
        ii. Definition of internal variables
    c. Definition of output data

i. Definition of distribution rights

ii. Definition of encryption option

iii. Definition of storage options

d. Definition of performance metrics

The DCU takes the inputted data and builds the defined system. If the system has been built and is already running, the DCU allows for updates to individual components without rebuilding the whole system.

When the DCU builds the system, it goes through a series of operations to build in the system implementation. Using the information provided the DCU captures and translates this information in the System Translator module. Mathematical equations that have been entered into the DCU through a mathematical editor are translated from MathML™ and then sent to the System Translator module for inclusion. This information is stored in a readable Microsoft Excel file. The format was selected to provide flexibility and options in building a DACE system. Designers can develop any tool to interact with the template as their situation deems necessary. It is even possible with the DACE configuration template, for an end user to generate a system configuration right from Excel without having a GUI.

The Excel configuration file is then used to define the system scheme. The System Parser module loads the file and determines the system boundaries and the arrangement of the components with respect to the boundaries (see Figure 9). In the figure below, the boundaries are represented by the "Computer Boundary" objects, which indicate that this boundary may or may not exist based on the system configuration. For example, in the Presentation Layer, there are "Computer Boundary" indicators between

each of the possible user interface devices. This could be the case where multiple client user applications are on various machines. But if there were no boundaries with multiple devices, then this would indicate that there are multiple client user applications on the same machine.

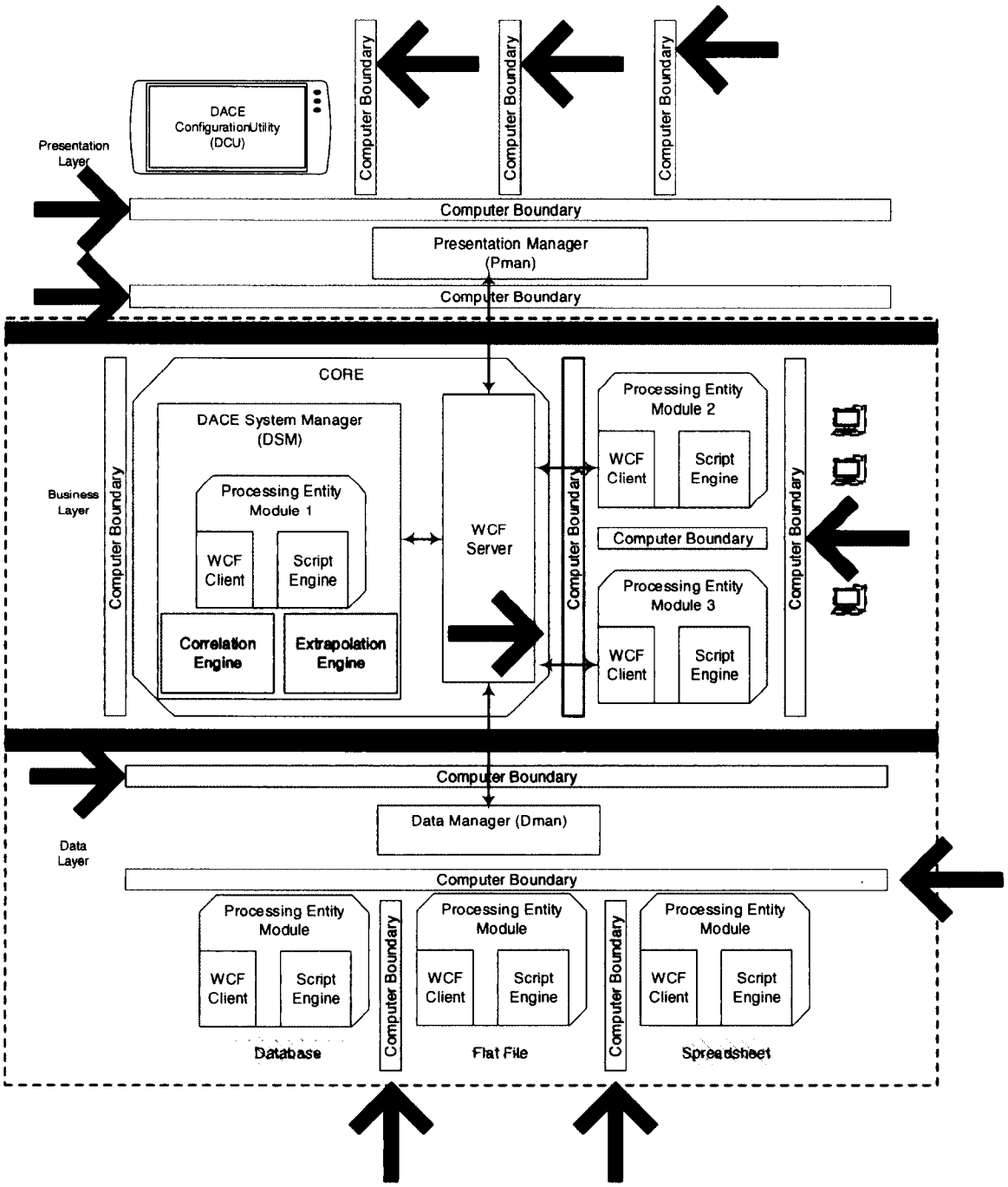



Figure 9. DACE Boundaries

In the Business Layer in Figure 9, "Computer Boundaries" depict separation of the Core and additional components, however, it could be the case that all of these components could reside on the same machine if configured as such (see Figure 10).

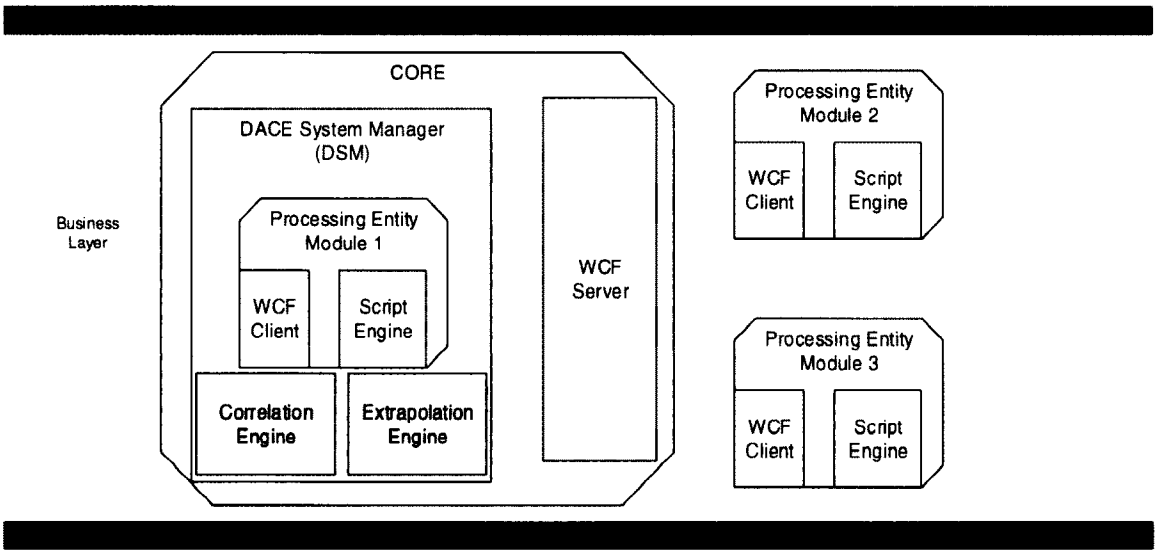



**Figure 10. DACE Business Service Layer, Single Machine**

Once the components of the system have been generated, each component's procedure is sent to the Script Generator module. This module essentially translates the module's procedure from a general pseudo format to a C# script file. Each generated script is then tested to make sure that it can be compiled by the Mono compiler. If there is no issue in the compilation of the script, then the component is considered ready for deployment and the System Updater module is signaled.

The System Updater first determines if the system component is already deployed or if a new system has been designed. If the component is new, then an installation package is generated with the component's application framework, which consists of a Scripting Engine, a WCF Client, and its correlated script file, if applicable. If the component has already been deployed and this is a modification or update, then a review occurs to determine what the change actually entails. If the script file has changed, then this file will be sent to the appropriate component for update. If a configuration setting has changed, then a new installation package will be generated for manual installation for

the component. The System Updater module will perform the above procedure for each component within the system. During this process, the System Monitor module will provide information on the status of the process, as well as, the status of the system. During a live update, meaning a components script is being updated, the System Monitor will control the state of the component to allow for the transaction to occur.

#### 2.3.1.2 Presentation Manager (Pman)

The Presentation Manager, or Pman, is a critical component within this layer. It provides an abstract interface to the DACE system that is language and platform independent. This means that user interface applications can be developed that fit the purpose of the system and can run on an operating system of the end user's choosing as long as the Mono framework can run on that operating system. The Pman, acting as a gateway to the DACE system, can also allow multiple user interfaces to connect and communicate with the system at the same time with multiple levels of access.

This component is system independent, meaning it can reside on any machine within the system. Figure 9 shows that the Pman is bound by Computer Boundaries. These boundaries may or may not exist depending on the defined system implementation. The Pman component could reside on a separate machine, a machine that contains a client access application, or even on the same machine where the Business Layer Service resides. This is accomplished by utilizing the WCF framework for communications. It contains a server that provides and controls client access and a backend client that communicates with the Presentation Service Layer's Core WCF Server.

The WCF Server module within the Pman provides the communication infrastructure for the external DACE architecture. WCF is a framework that was designed by Microsoft for providing service-oriented communications implemented in the .NET Framework. This framework was selected due to its extensibility, reliability, security, interoperability, and service orientation. The DACE architecture implements this framework and provides configuration options for communication protocols, security, reliability, and durability.

Three communication protocols have been identified to support Presentation Service Layer client applications, these are: Named Pipes, Transmission Control Protocol (TCP), and Hypertext Transfer Protocol (HTTP). Each type provides a specific functional requirement that supports the "Model" axis for local versus distributed system characteristics. For system components that are physically located on the same machine (local) and need to communicate with each other, the named pipes communication protocol is used. While system components that are on different machines (distributed) can utilize the TCP or HTTP protocol based on client application implementations.

Client applications that have been implemented in a popular website programming language such as JavaScript or ASP.NET would most likely use the HTTP protocol for interaction with the DACE Pman. Although this is not a requirement, typical web browser clients use HTTP. They could implement a client side backend that would implement a TCP protocol.

The design also allows for various options for interaction by client applications. These actions allow for control of system operational states, as well as, receiving both status and computational information. Based on the general functionality, these have been

categorized into three levels of client access control for client applications to connect to the system (see Table 1). Access control is useful in systems were there are multiple types of access permissions allowed within a system. An example might be in a ship's power plant system where there are several stations for interaction. The engineering station may need full access to change parameters and monitor. A bridge station might only have monitoring capabilities to provide situational awareness information to the captain on the vessel's power plant efficiency. Access control is configured and controlled via the DACE Configuration Utility. This information is sent and utilized within the Pman to control system access. User access control levels defined for interaction include: Full Access Control, Limited Access Control, and Monitor Access Control.

**Table 1. Access Control Levels**

| Capability | Full Access | Limited Access | Monitor Access | No Access | Description |
|---|---|---|---|---|---|
| System Control Procedure: Terminate | √ | If Granted | | | Allows the user to terminate one or more processes executing within the system. |
| System Control Procedure: Restart | √ | If Granted | | | Allows the user to restart one or more processes executing within the system. |
| System Control Procedure: Suspend | √ | If Granted | | | Allows the user to suspend one or more processes executing within the system at their current state. |
| System Control Procedure: Resume | √ | If Granted | | | Allows the user to resume one or more processes executing within the system if they are currently in the halt state. |
| System Control Procedure: Update | √ | If Granted | | | Allows the user to update one or more processes executing within the system. |
| System Health Status | √ | If Granted | If Granted | | Allows client applications to receive system status information through the Pman. |
| Computational Results | √ | If Granted | If Granted | | Allows client applications to receive computational results updates through the Pman. |
| Remote Desktop | √ | If Granted | | | Allows client DCU terminals to access system computers using a remote desktop connection. |
| Direct Warehouse Data Retrieval | √ | If Granted | If Granted | | Allows client applications to perform queries on the stored data in the systems warehouse. |

### 2.3.2 Business Service Layer (Core)

The Business Layer is considered the heart of the system. It is comprised of a Core and one or more PEMs. The Core is subdivided into the DACE System Manager, a WCF Server, a Correlation Engine, an Extrapolation Engine, and a PEMs. Each of these modules provides a distinct function for execution of a defined system implementation. It is responsible for all of the input and output data translation, the layer's communications

and data acquisition (Assimilation), the execution of the defined procedure methods (Analysis), the processes information Fusion (Correlation), and the factorization/prediction of data (Extrapolation). This layer is the most complex layer within the design. All of the 12 variable elements apply to this layer.

#### 2.3.2.1 Components

The DSM manages the overall system. It provides mechanisms for configuration, monitoring, and controlling of a defined DACE application implementation. It coordinates system updates from client applications through the Pman module, controls the system process workflow among PEMs, and coordinates the storage and retrieval of data through the Dman module.

The WCF Server module provides the communication infrastructure for the internal DACE architecture. WCF is a framework that was designed by Microsoft for providing service-oriented communications implemented in the .NET Framework. This framework was selected due to its extensibility, reliability, security, interoperability, and service orientation. The DACE architecture implements this framework and provides configuration options for communication protocols, security, reliability, and durability.

Two communication protocols have been identified to support inner process system communications, these are: Named Pipes and TCP. Each type provides a specific functional requirement that supports the "Model" axis for local versus distributed system characteristics. For system components that are physically located on the same machine (local) and need to communicate with each other, the Named Pipes communication protocol is used for Windows based operating systems. For UNIX based machines, there

is currently no equivalent. There is current work being done in this area under the Mono framework. System components that are on different machines (distributed) utilize the TCP protocol.

The PEMs is responsible for procedure processing (see Figure 11). In a general case, this aligns with the general software program that executes a set of operations and performs a set of actions. The difference lies in its implementation. This module essentially acts as an internal client to the DSM, based on the idea that all PEMs reside within the Presentation Service Layer boundaries of the Three-Tier Architecture. It contains processing logic and two sub modules which are: a WCF client module and a Script Engine module. The PEMs processing logic coordinates execution of the WCF and the Script Engine.

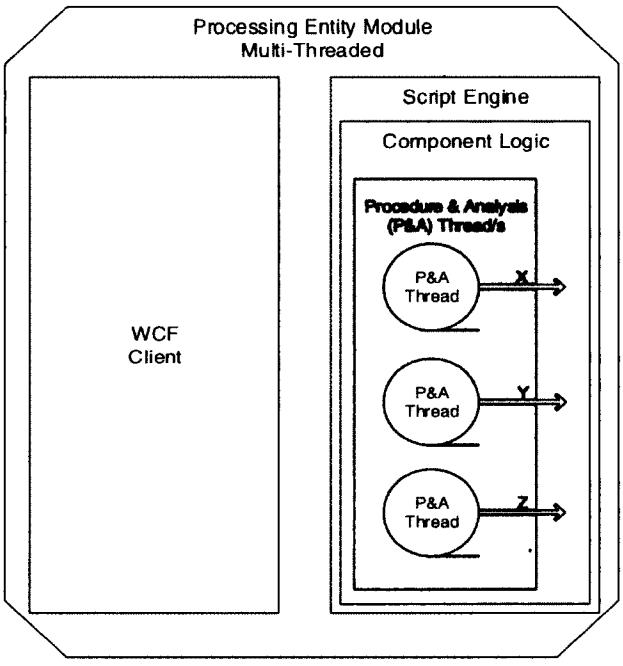



**Figure 11. Detailed Processing Entity Module**

The WCF client module provides an infrastructure that supports internal communications from components to the WCF servers. This module is the only WCF client module used within the system. It was designed to dynamically load configuration scripts that provide the communication schema that aligns with the appropriate WCF server based on a defined system implementation. For system PEMs, if a PEMs is local then then the Named Pipes communication protocol is used (Windows only). For a PEMs that is on another machine or on a UNIX based machine, the TCP communication protocol is used.

To achieve the ability to dynamically create and execute the PEMs in the DACE architecture and fulfill the defined 12 variable elements required the flexibility of a scripting engine. The Script Engine is a C# scripting system that interacts directly with any Common Language Runtime (CLR) for .NET and Mono. The use of a scripting engine is useful when there are frequent code changes, development and deployment time is expected to be faster, and the solution requires more flexibility in application deployment instances. This module allows for script files to be loaded, compiled, debugged, and executed on a target machine. This functionality was needed in the Application element of the 12 variable elements. Since it ties into the CLR, it uses the CLR compiler and debug engine to provide direct target support mechanisms for in-system updates, testing, and debugging system configurations. The engine can also compile scripts into an executable, a dynamically linked library (dll), service application, or run them in a debug mode. Running scripts in a debug mode provides in-system debugging even for multiple components in a distributed configuration.

2.3.2.2 Configurations

The Business Service Layer is very dynamic in nature. It can be configured in various configurations based on system definitions. This layer can be configured as a simple single threaded application or service, to a multi-threaded multi-core distributed set of services on multiple operating systems. These options have been categories into four main items: Single Processing, Single Processing Multi-Threaded, Single Processing Mixed-Threaded Multi-PEMs, and a Mixed Processing Mixed Threaded Multi-PEMs system.

A single processing system is defined as a system comprised of only containing the Core module (see Figure 12). Single processing refers to the component residing on a single machine. All logic and procedures are executed within a single PEMs and in a single processing or execution thread called Procedure & Analysis (P&A) threads. This is very similar to a simple linear software application, where operations are performed in order.

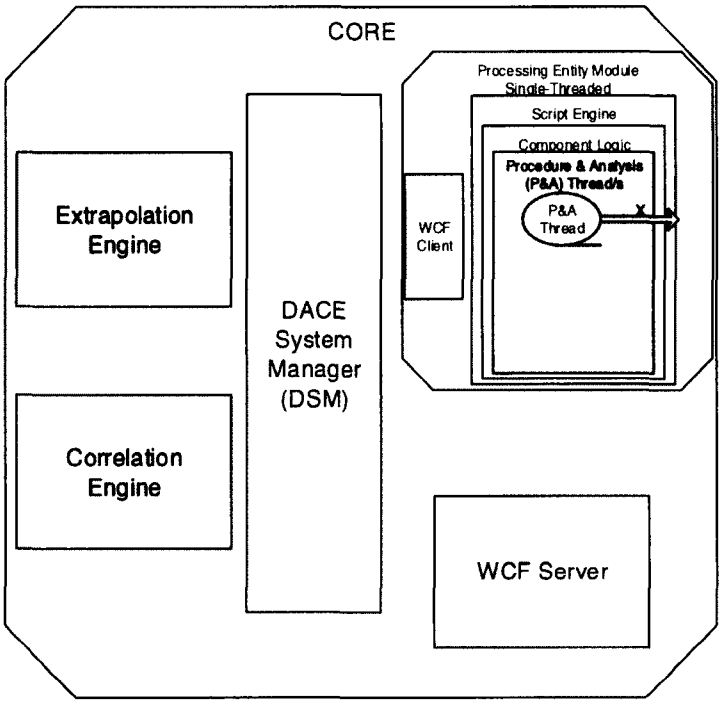



**Figure 12. Single Processing System**

A single processing multi-threaded system is defined as a system comprised of only containing the Core module (see Figure 13). Single processing refers to the component residing on a single machine. All logic and procedures are executed within a single PEMs but disjoint or parallel processes are separated in multiple P&A threads to be executed.

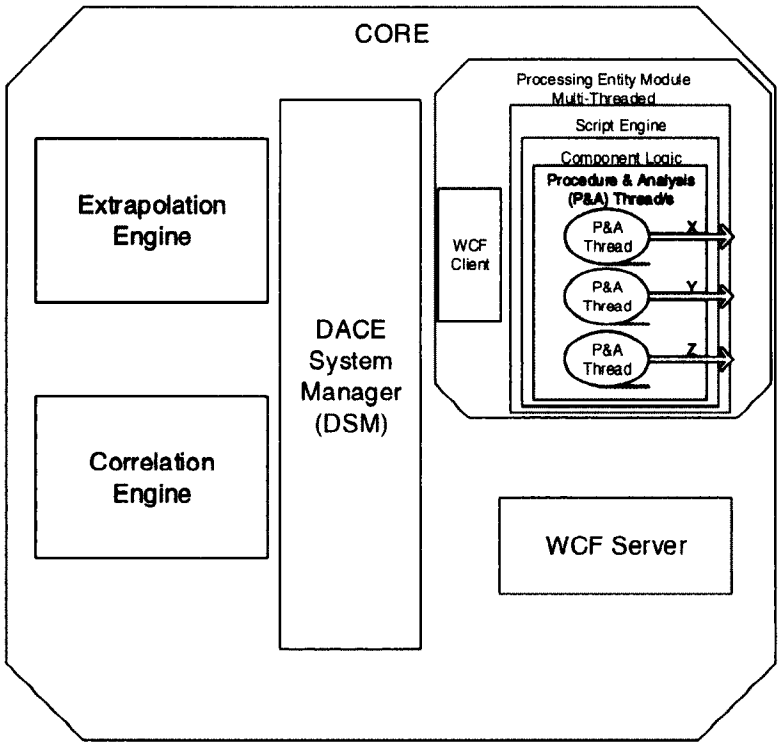



**Figure 13. Single Processing Multi-Threaded System**

A single processing mixed threaded multi-PEMs system is defined as a system comprised of containing a Core module and one or more PEMs (see Figure 14). Single processing refers to the components residing on a single machine. All logic and procedures are executed within various PEMs where each may have either single or parallel processes in P&A threads to be executed.

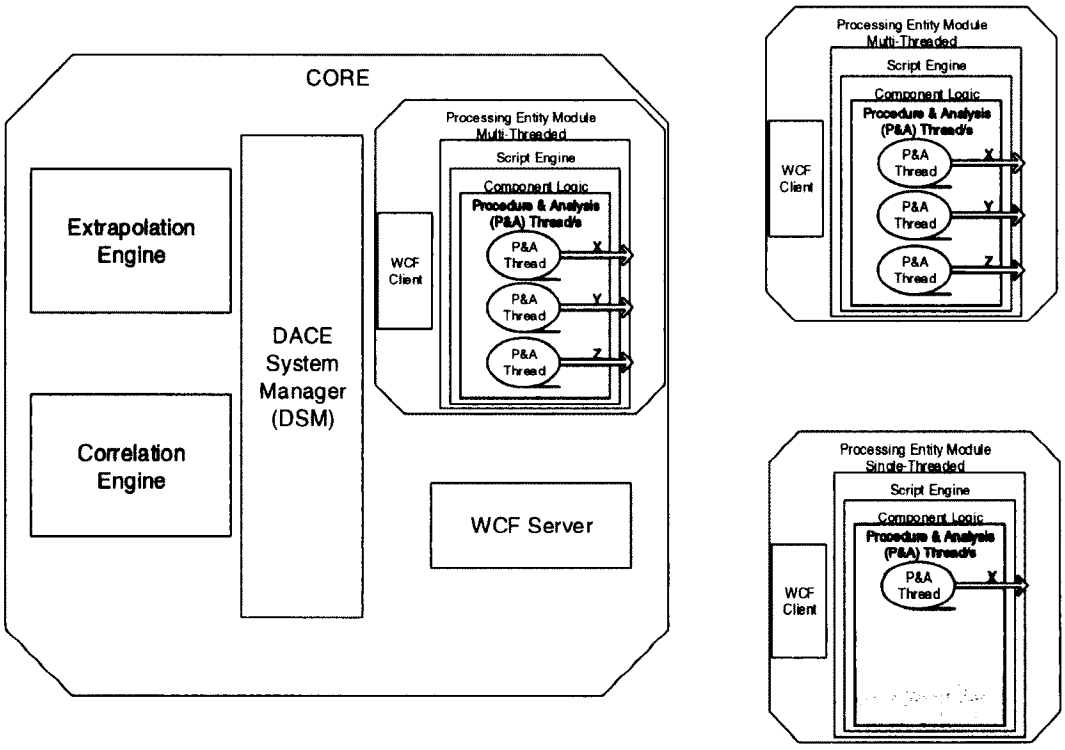



**Figure 14. Single Processing Mixed Threaded Multi-PEMs System**

A mixed processing mixed threaded architecture is defined as a system comprised of the Core module and at least one Process Entity Module on different machines. This means that any of the modules could be executing in either a single or multiple thread capability. In Figure 15, the Business Service Layer within the system is configured to reside on four machines. The Computer Boundaries depict physical separation of the components which translate to physical machines. As you can see, multiple PEMs can reside on the same machine, as depicted on Machine #2.

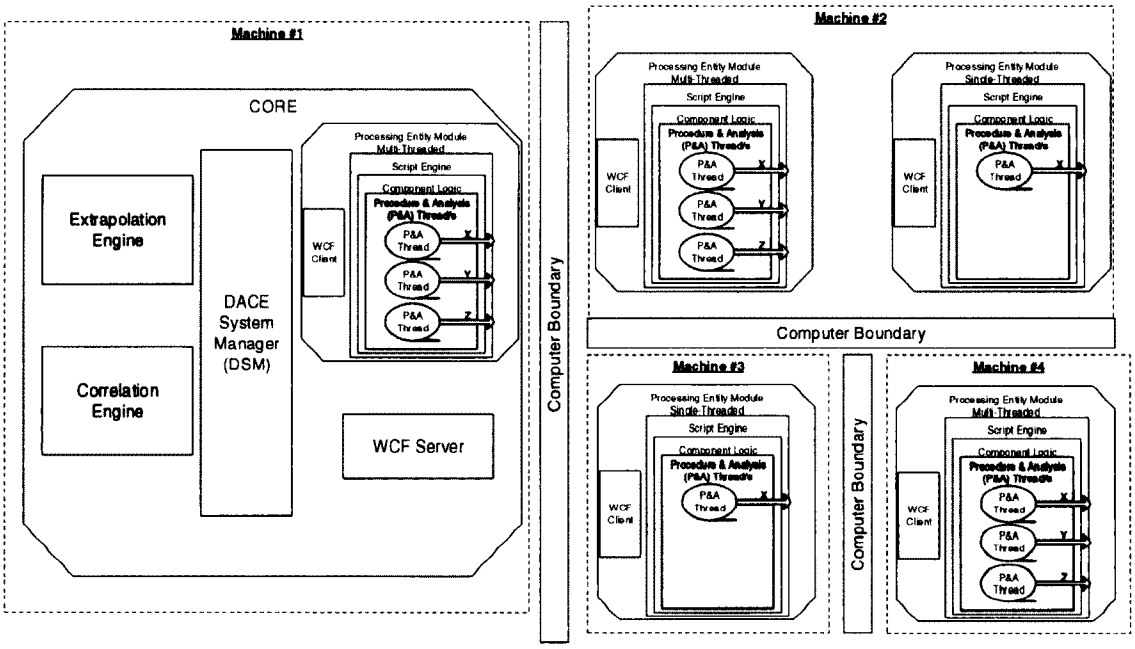



**Figure 15. Mixed Processing Mixed Threaded Multi-PEMs System**

### 2.3.3 Data Service Layer

The Data Service Layer is the data warehousing component of the architecture. It allows the Business Service Layer to interface with a data storage container in a consistent fashion without working knowledge on the how the data storage container is implemented. This layer was designed in this fashion not to only allow for a common storage mechanism but also to allow for flexibility, portability, and expandability. This layer allows for multiple storage containers to be utilized even when of different types. This layer contains the Dman and the PEMs system components (see Figure 16).

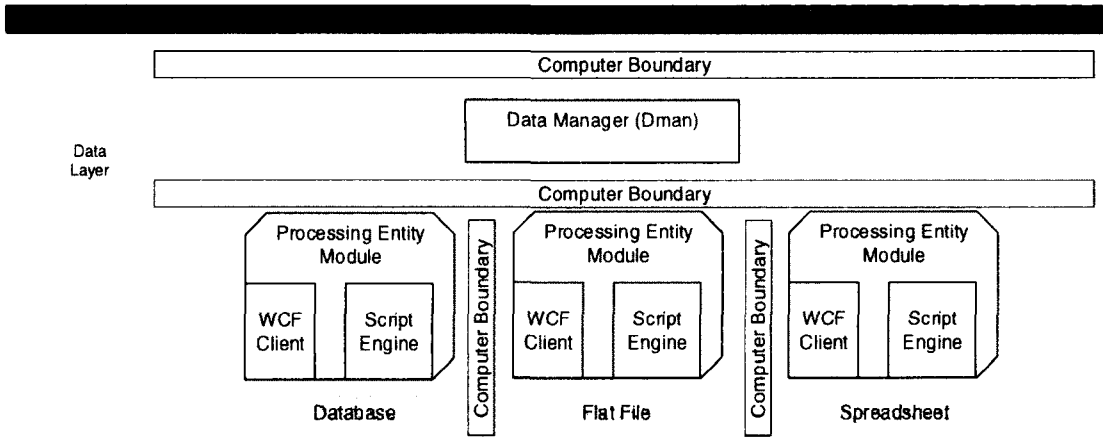



**Figure 16. DACE Data Service Layer**

2.3.3.1 Components

The Data Manager, or Dman, provides an interface mechanism for the Business Service Layer. Its primary focus is to provide a common interface for storing and retrieving data, independent of the storage mechanisms deployed. The common data warehouse model is intended to provide the same interfacing functionality to storage containers. The design aligns with standard database operations such as Query, Insert, Update, and Delete. This component has been designed in a fashion similar to the Pman but containing additional logic to perform synchronization and transfer operations for the PEMs warehousing components. It is system independent, meaning it can reside on any machine within the system. Figure 16 shows that the Pman is bound by Computer Boundaries. These boundaries may or may not exist depending on the defined system implementation. The Dman component could reside on a separate machine, a machine that contains a data warehousing mechanism, or even on the same machine that Business Layer Service resides on. This is accomplished by utilizing the WCF framework for communications. It contains a server that provides and controls the PEMs for each

implementation of the data warehousing and a backend client that communicates with the Presentation Service Layer's Core WCF Server.

Two communication protocols have been identified to support PEMs warehousing, these are: Named Pipes and TCP. Each type provides a specific functional requirement that supports the "Model" axis for local versus distributed system characteristics. For system components that are physically located on the same machine (local) and need to communicate with each other, the Named Pipes communication protocol is used. While system components that are on different machines (distributed) can utilize the TCP protocol.

The PEMs modules depicted in Figure 16 are the same modules utilized within the Business Service Layer. They are configured to connect to the Dman WCF server using either TCP or Named Pipes. Each PEMs in this layer loads a warehousing interaction script depending on what type of data storage container is selected. The script contains the logic to correctly translate commands to and from the common data format and the desired data storage container format and functionality.

#### 2.3.3.2 Execution/Work Flow

Operations within the Data Service Layer are simplistic from a higher point of view. Messages to store or retrieve data are received by the Dman. In the case where there are multiple storage containers, the request is then evaluated and distributed to the appropriate PEMs. If there is more than one storage method and all warehouses are storing the same data, then the message is distributed to all relative PEMs. Once a

message is received by a PEMs, it is then translated from the common messaging format to the appropriate interfacing format for that specific storage container.

If the operation was a "Store" procedure then the operation is complete. If the operation was a "Retrieve" or "Query" procedure, then the PEMs performs the appropriate action. Once the data have been received within the PEMs, the data are then packaged into the common message format and transmitted back to the calling module.

## 2.4 Technology

The DACE design premise is based on the concept of flexibility. This means that the underlying technologies and architecture need to be designed with a variety of flexible characteristics. The two primary elements that influence the underlying technologies are operating system independence and language support.

### 2.4.1 Operating System Independence

The DACE architecture is designed to be able to run on multiple operating systems. The reason for this is so that there would not be a limitation to the users based on their system operational requirements. The primary operating systems focused on for this design are Windows 7, Windows 8, UNIX, Linux, and Mac OSx.

To open the design up to multiple operating systems required a development environment and technologies that enabled cross compatibility. The Mono Framework was chosen as the underlining mechanism to fulfill this requirement. The Mono framework is a cross platform development platform that is an open source .NET

development framework. It closely parallels Microsoft's .NET framework which is based on the ECMA standards for the Common Language Runtime (CLR).

### 2.4.2 Language Support

The DACE framework, by utilizing Mono, allows various programming languages and technologies to be utilized. The .NET model defines a common language runtime and a common class library which provides common operations to technologies such as C++, C#, J#, Visual Basic (VB), Active Server Pages (ASP), and ActiveX Data Objects (ADO). The variety of technologies that can interact with the DACE architecture allows for other developers to use the language or technology of choice to interact with the DACE application. In the design of the DACE framework the primary technologies are C# and Windows Communication Foundation or WCF.

## 2.5 Infrastructure Relationship Matrix

The Infrastructure Relationship Matrix provides a visual representation of the relationship of the architecture's components, modules, and constructs with the technologies defined for implementation. The Infrastructure Relationship Matrix, shown in Appendix B, shows the full mapping of the various aspects of the DACE design. The matrix is sectioned into four groups. It is essentially built around Group 1 which is the focus point. The focus point consists of the "Components" and the "Modules." These are the implementation elements of the design (see Figure 17). This focal point was chosen due to the criticality of items in the implementation of the proposed solution. By working from the bottom up, or implementation, to the theoretical details of the actual

development items can be explored and evaluated against the system design and the requirements. Relationships are represented by either an "X" or an abbreviation in the case of the Three-Tier Architecture designators. The designators for this item in the matrix are PSL for Presentation Service Layer, BSL for Business Service Layer, and DSL for Data Service Layer.

The focus point of the matrix correlates the system Components to the system Modules. For example, in Figure 17, by tracing from the System Manager horizontally and the Core vertically, you can see that there is an identified relationship between the Component and Module.

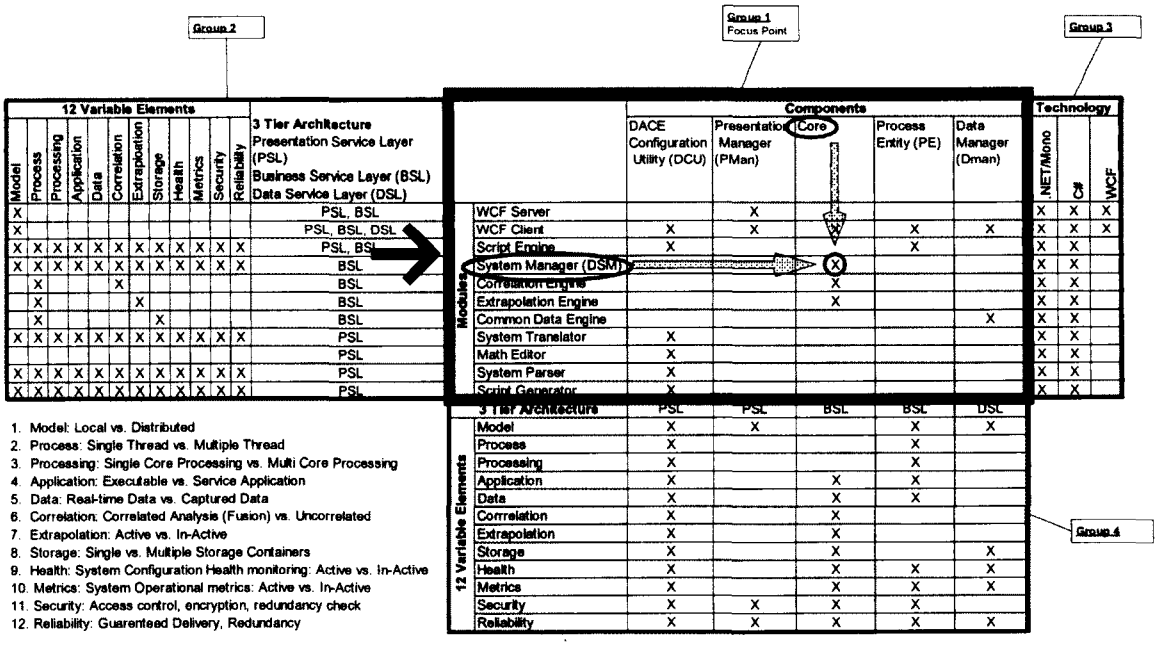



**Figure 17. Infrastructure Relationship Matrix Focus Point**

When looking at the relationships from the Three-Tier Architecture to either the design Modules or the Components, the matrix must be evaluated horizontally or vertically independently by using pairs of groups. For example, to determine if there is a relationship between the Three-Tier Architecture and a Module, Groups 1 and 2 are utilized. Tracing from the Three-Tier Architecture items in Group 2 vertically and the Module items in Group 1 horizontally, one can see that if there is an identified relationship between these items with the design framework, either one or more abbreviations is indicated (see Figure 18).

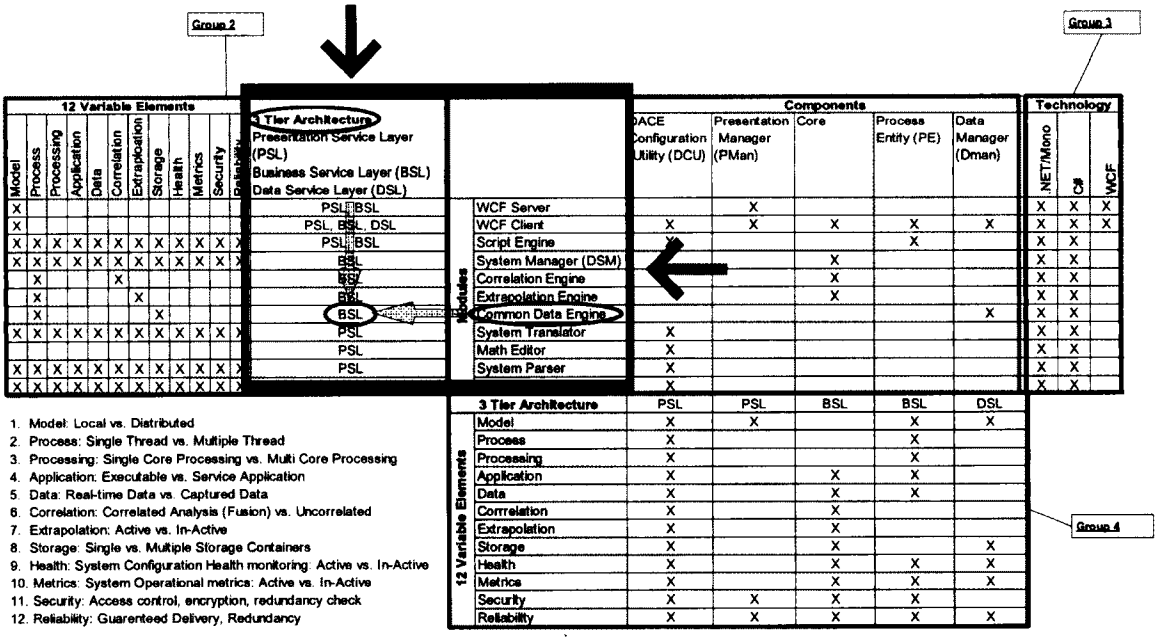



**Figure 18. Infrastructure Relationship Matrix, Three-Tier vs. Modules**

When looking at the relationships between the "Three-Tier Architecture" to a Component, the matrix must be evaluated using Groups 1 and 4. For example, by tracing from the Three-Tier Architecture items in Group 4 horizontally and the Component items

in Group 1 vertically, one can see if there is an identified relationship within the design framework, either one or more abbreviations is indicated (see Figure 19).

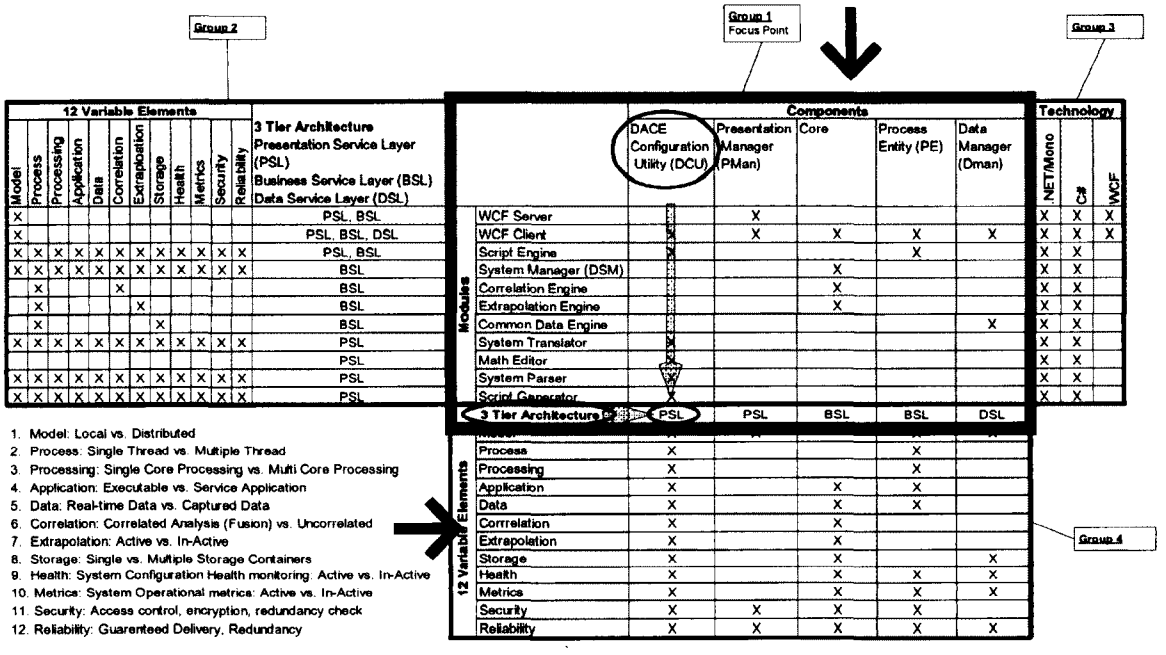
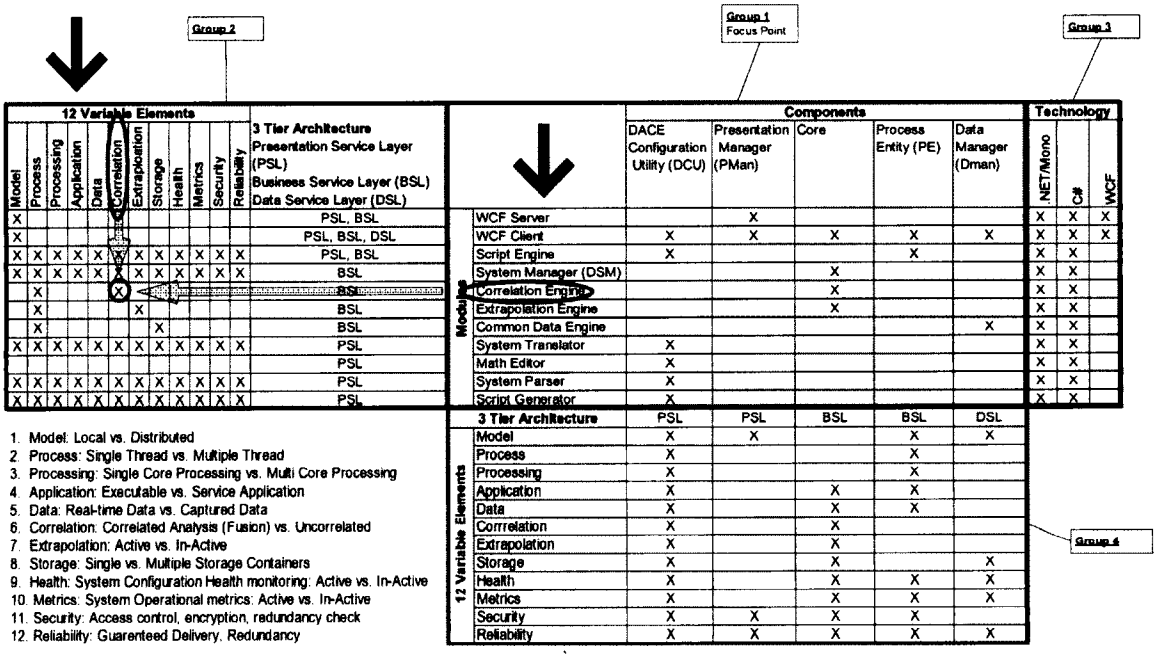



**Figure 19. Infrastructure Relationship Matrix, Three-Tier vs. Components**

When looking at the relationships from the 12 variable elements to either the design Modules or the Components, the matrix must be evaluated in the same fashion as the Three-Tier Architecture methods, either horizontally or vertically independently, by using pairs of groups. For example, to determine if there is a relationship between the 12 variable elements and a Module, Groups 1 and 2 are utilized. Tracing from the 12 variable elements in Group 2 vertically and the Module items in Group 1 horizontally, one can see that if there is an identified relationship between these items within the design framework, an "X" is indicated (see Figure 20).




| Model | Process | Processing | Application | Data | Correlation | Extrapolation | Storage | Health | Metrics | Security | Reliability | 3 Tier Architecture |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X | | | | | | | | | | | | PSL, BSL |
| X | | | | | | | | | | | | PSL, BSL, DSL |
| X | X | X | X | X | X | X | X | X | X | X | X | PSL, BSL |
| X | X | X | X | X | X | X | X | X | X | X | X | BSL |
| | X | | | | X | | | | | | | BSL |
| | X | | | | | X | | | | | | BSL |
| | X | | | | | | X | | | | | BSL |
| X | X | X | X | X | X | X | X | X | X | X | X | PSL |
| | | | | | | | | | | | | PSL |
| X | X | X | X | X | X | X | X | X | X | X | X | PSL |
| X | X | X | X | X | X | X | X | X | X | X | X | PSL |

| Modules | Components | DACE Configuration Utility (DCU) | Presentation Manager (PMan) | Core | Process Entity (PE) | Data Manager (Dman) | Technology .NET/Mono | C# | WCF |
|---|---|---|---|---|---|---|---|---|---|
| | WCF Server | | X | | | | X | X | X |
| | WCF Client | X | X | X | X | X | X | X | X |
| | Script Engine | X | | | X | | X | X | |
| | System Manager (DSM) | | | X | | | X | X | |
| | Correlation Engine | | | X | | | X | X | |
| | Extrapolation Engine | | | X | | | X | X | |
| | Common Data Engine | | | | | X | X | X | |
| | System Translator | X | | | | | X | X | |
| | Math Editor | X | | | | | X | X | |
| | System Parser | X | | | | | X | X | |
| | Script Generator | X | | | | | X | X | |

| 12 Variable Elements | 3 Tier Architecture | PSL | PSL | BSL | BSL | DSL |
|---|---|---|---|---|---|---|
| | Model | X | X | | X | X |
| | Process | X | | | X | |
| | Processing | X | | | X | |
| | Application | X | | X | X | |
| | Data | X | | X | X | |
| | Correlation | X | | X | | |
| | Extrapolation | X | | X | | |
| | Storage | X | | X | | X |
| | Health | X | | X | X | X |
| | Metrics | X | | X | X | X |
| | Security | X | X | X | X | |
| | Reliability | X | X | X | X | X |

**Figure 20. Infrastructure Relationship Matrix, 12 Variable Elements vs. Modules**

When looking at the relationships between the 12 variable elements to a Component, the matrix must be evaluated using Groups 1 and 4. For example, by tracing from the 12 variable elements in Group 4 horizontally and the Component items in Group 1 vertically, one can see if there is an identified relationship within the design framework, an "X" is indicated (see Figure 21).

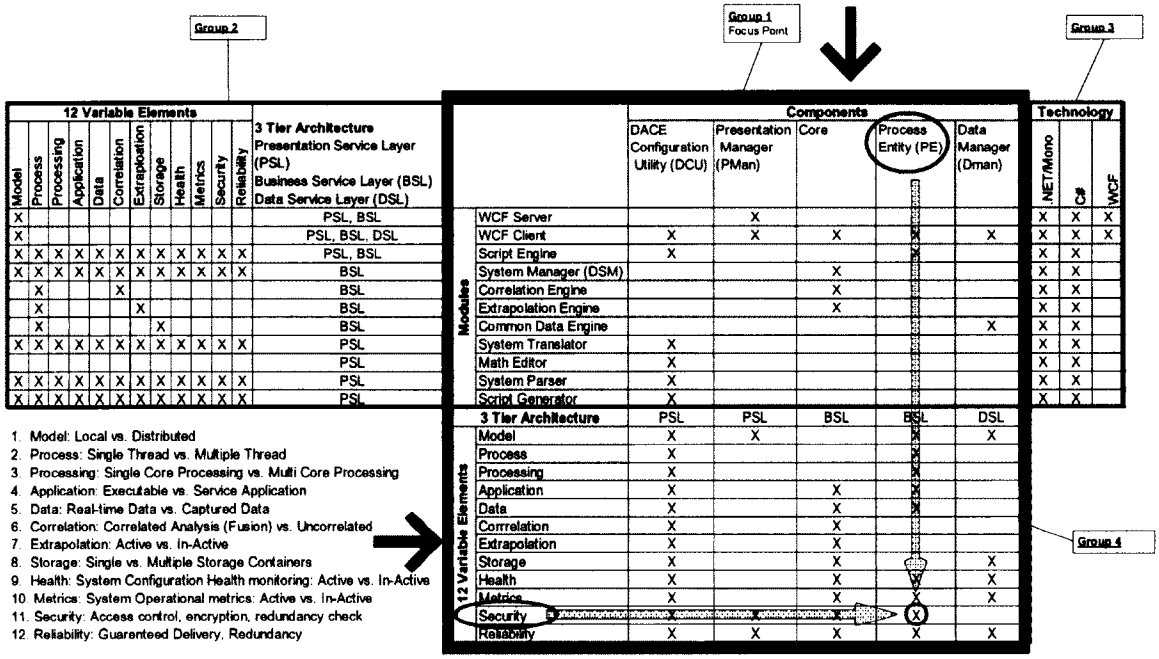



**Figure 21. Infrastructure Relationship Matrix, 12 Variable Elements vs. Components**

In the evaluation of Technology to the items in the focus point, there is only a single process. This is due to the fact that all items listed under Technology relate to all items under the Components. This leaves the evaluation of Technology to the Modules. This is accomplished by tracing from the Technology items in Group 3 vertically and the Modules items in Group 1 horizontally, one can see that if there is an identified relationship within the design framework, an "X" is indicated (see Figure 22).

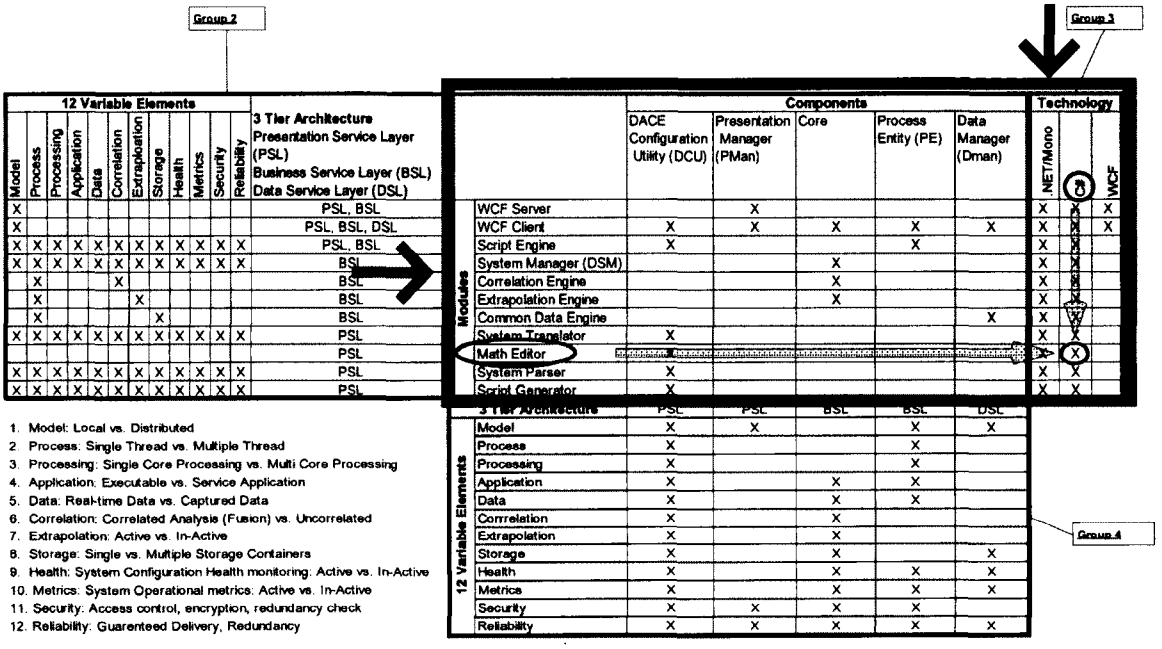



**Figure 22. Infrastructure Relationship Matrix, Technology vs. Modules**

# CHAPTER 3

## SCOPE OF PROJECT

The overall DACE framework was highly complex and is a very large task to accomplish. Due to the scope of the complete framework which entails the Presentation Service Layer, the Business Service Layer, and the Data Service Layer, a bounded subset (see Figure 23) of the framework's design for this project has been defined which is to be considered Phase I of the application's full design.

The Phase I focus is on the Business Service Layer design with a subset of the Data Service Layer and Presentation Service Layer (see Figure 23). The Presentation Service Layer is not intended to be the primary focus of this effort. This was due to the large task of implementing the graphically intensive interface, a DACE Configuration Utility or DCU, with limited functionality was developed to configure and update the developed system.

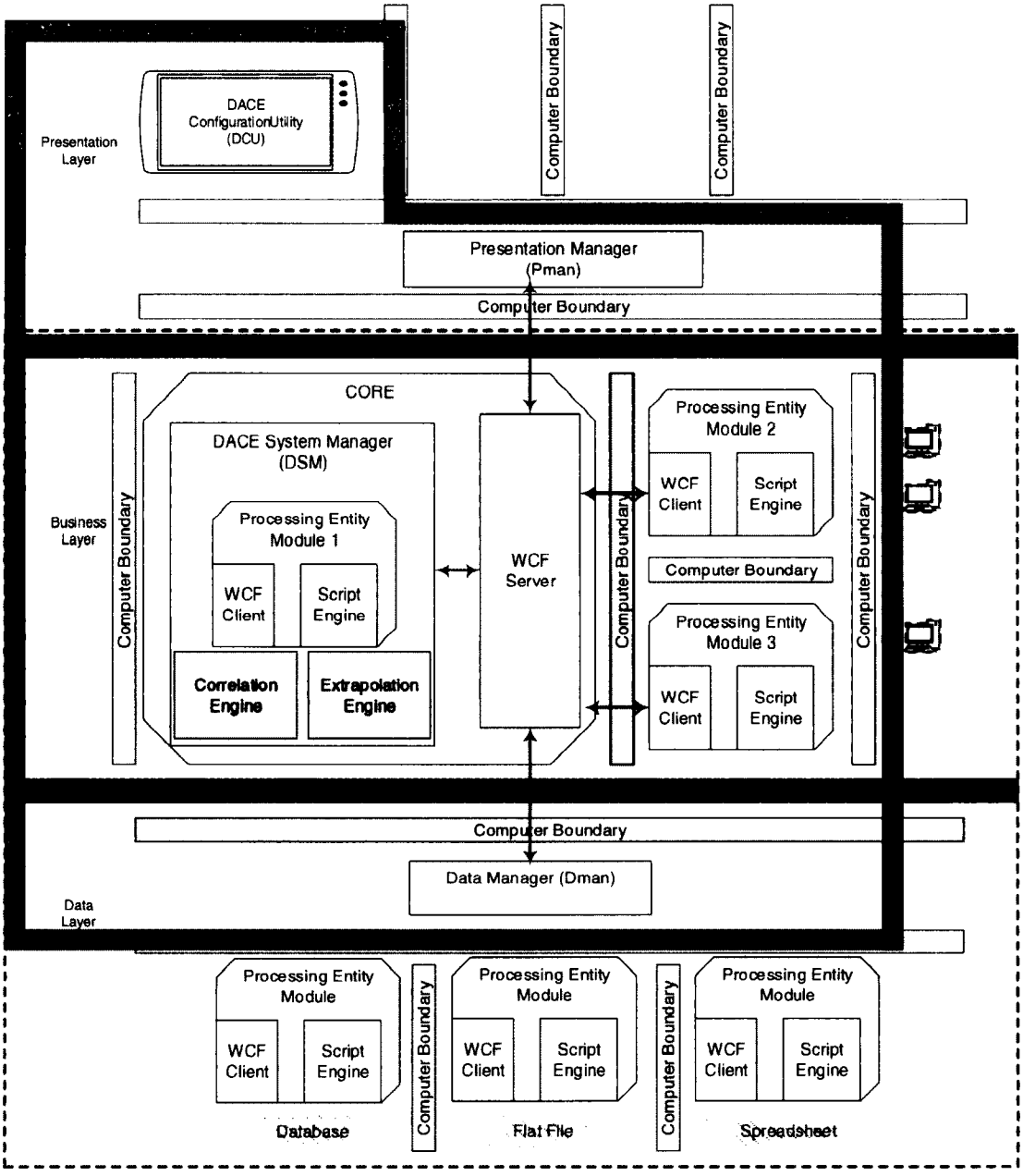



**Figure 23. Project Scope**

To implement the DACE system, three methods were chosen. The first method was the utilization of an open source Commercial Off-The-Shelf (COTS) component. Since the design of DACE required a scripting engine and given the size and scope of developing a fully functional engine was not the primary goal of this project, a Common

Language Runtime (CLR) scripting engine called CS-Script was chosen to be utilized. The second method of module implementation was the actual coding of each module in C#. The third method was script generation. System specific scripts are auto generated by DACE upon the development of system configurations. Table 2 shows the modules and the methods that were utilized to implement DACE.

**Table 2. DACE Module Implementation**

| *Phase I Level of Effort Module Implementations* | | **Implementation Method** | | | **Applicable Layers** Presentation Service Layer (PSL) Business Service Layer (BSL) Data Service Layer (DSL) |
|---|---|---|---|---|---|
| | | C# Code Developed | Components Off-The-Shelf (COTS) | Generated Scripts from "Script Generator" | |
| **Modules** | WCF Server | X | | | PSL, BSL |
| | WCF Client | X | | | PSL, BSL, DSL |
| | Script Engine (CS-Script) | | X | | PSL, BSL |
| | System Manager (DSM) | X | | | BSL |
| | Correlation Engine | X | | | BSL |
| | Extrapolation Engine | X | | | BSL |
| | Common Data Engine | X | | | BSL |
| | System Translator | X | | | PSL |
| | Math Editor | X | | | PSL |
| | System Parser | X | | | PSL |
| | Script Generator | X | | | PSL |
| | Application Functionality | | | X | BSL |

The COTS CS-Script engine is a MIT Licensed product that can be used within other applications without restriction. The DACE system utilizes this engine to provide

the functionality to load and execute custom C# code. To utilize this engine, additional integration code needed to be developed.

The code development using the C# language was the majority of the systems implementation effort. There were 15 modules needed for system implementation, classified into two separate tiers based on functional application level within the design. Table 3 shows the low level modules and their supporting roles to the higher level modules that are in Figure 23.

The script generation method is the dynamic process of the system generating run-time scripts based on a system configuration. These scripts can then be loaded upon startup of a defined systems implementation.

**Table 3. Development Level of Effort**

| Modules | |
|---|---|
| Level 1 | Level 0 - Support Modules |
| DCU | WCF Client |
| | Script Engine Integration |
| | System Translator |
| | Math Editor |
| | System Parser |
| | Script Generator |
| | Function & Integration Code |
| | |
| Pman | WCF Server |
| | WCF Client |
| | Function & Integration Code |
| | |
| Dman | WCF Server |
| | WCF Client |
| | Function & Integration Code |
| | |
| PEMs | WCF Client |
| | Script Engine Integration |
| | Function & Integration Code |
| | |
| DSM | PEMs |
| | Correlation Engine |
| | Extrapolation Engine |
| | Function & Integration Code |
| | |
| Core | DSM |
| | WCF Server |
| | Function & Integration Code |
| | |

### 3.1 Capabilities

The Phase I implementation has the ability to configure, build, run, and monitor a system definition through utilization of the DCU. Since the graphical configuration module has not been implemented in this phase, all system definition operations are manually performed using the DACE Configuration template. The resultant configuration file is then loaded and built using the DCU. The generated files are stored in a specific folder directory which can be accessed within the DCU. These files can then be transferred to the appropriate computer. If this is a new build of the system configuration, then the files need to be manually moved. If the system structure already exists on the computers with no physical changes to where the components are or the quantity of computers, and the only thing that changed was workflow logic, then the DCU can be used to transfer the updated items.

The Business Service Layer has been fully implemented. This allows for a variety of complex configurations to test system functionality and performance. There is no limit on the number of PEMs which can be configured within the system. All of the system configuration categories Single Processing, Single Processing Multi-Threaded, Single Processing Mixed-Threaded Multi-PEMs, and a Mixed Processing Mixed Threaded Multi-PEMs can be achieved within the current system state.

The Data Service Layer has only been partially implemented. This layer was not a focus point for this phase but due to the similarities of the Dman and the Pman, the Dman has been partially developed. The communication and workflow logic has been developed. The common data engine module, which provides the common data warehousing interface mechanism, is slated for Phase II development.

## 3.2 Development Methodology

The DACE framework was developed using the Agile development process integrated into a system engineering process. Agile development uses iterations and continuous feedback to refine and deliver a software implementation. The Dynamic Systems Development Method (DSDM) Atern (Arctic Tern), which is a development method of the Agile development process, was used (see Figure 24). This method was designed to focus on rapid application development. It tries to solidify the development efforts time, cost, and quality at the beginning of a project by using identifiers such as musts, shoulds, coulds, and won't haves to meet a set deliverable.

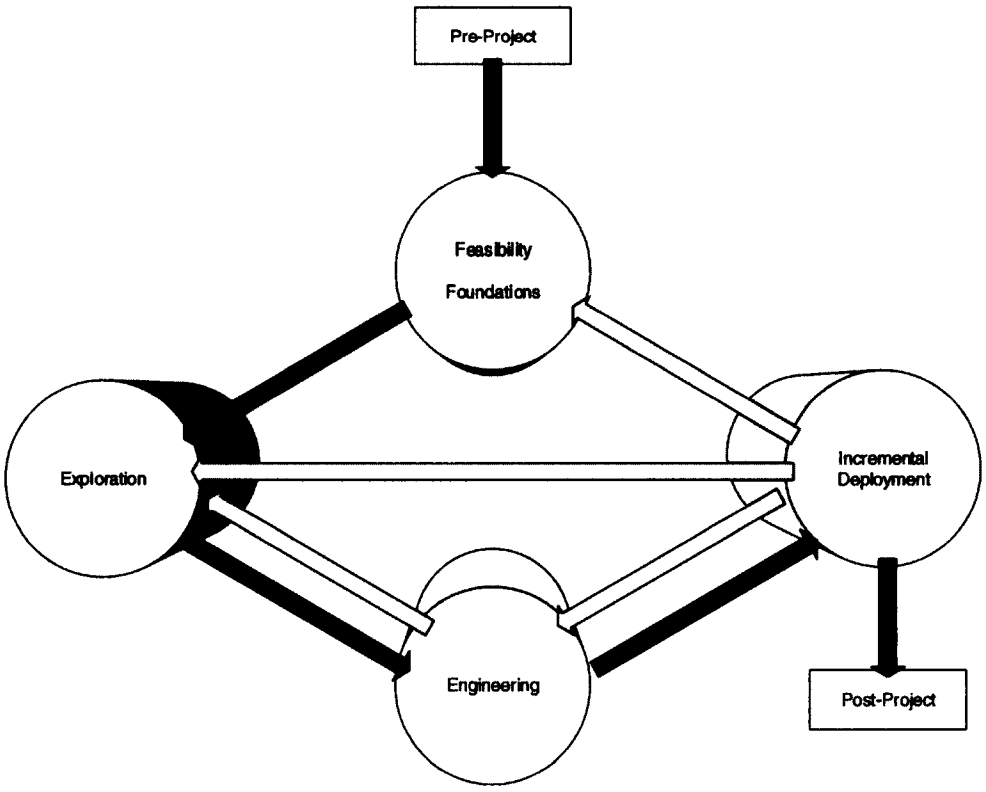



**Figure 24. Dynamic Systems Development Method (DSDM) Atern (Arctic Tern)**

## 3.3 Metrics

In the design and development of the DACE framework, several metrics were selected to quantify the system. To measure the architecture and quality of the design, the Design Structure Quality Index (DSQI) models were utilized. To quantify the implementation the following measurements were selected: number of lines of code, program execution time, program load time, and size.

### 3.3.1 Design Structure Quality Index

The Design Structure Quality Index (DSQI) is metric for architectural designs implemented using object oriented design. This method was developed by the United States Air Force. It is used to evaluate computer program efficiency relative to its code modules. This correlates to the quality of the systems design. Results from this method, range from 0 (lower quality) to 1 (higher quality) in range. There are seven variables evaluated, these are:

1. Total number of modules in the architecture.
2. Number of modules that depend on data input or produce data to be produced in another module.
3. Number of modules who depend on prior processing.
4. Number of database items.
5. Total number of unique database items.
6. Number of database segments.
7. Number of modules with only 1 entry and exit point within the module.

### 3.3.2 Implementation Characteristics

The system implementation characteristics selected are an attempt to provide quantifiable data to a system configuration for the DACE framework design. The system design shown in Figure 23 is the physical layout of the configuration with all of the baseline modules included. The implementation characteristics defined are:

- Number of lines of code.
- Program execution time.
- Program load time.
- Program size.

## 3.4 Schedule

The Gantt chart (see Figure 25) shows the Phase I development schedule which defines four phases. This schedule was a developed as a result of the architecture design. The four phases defined in the schedule are the System Requirements Development, the Full System Design, the Phase I Development, and the Experimental Evaluation. The sections that follow provide an overview of phases relative to the schedule in Figure 25.

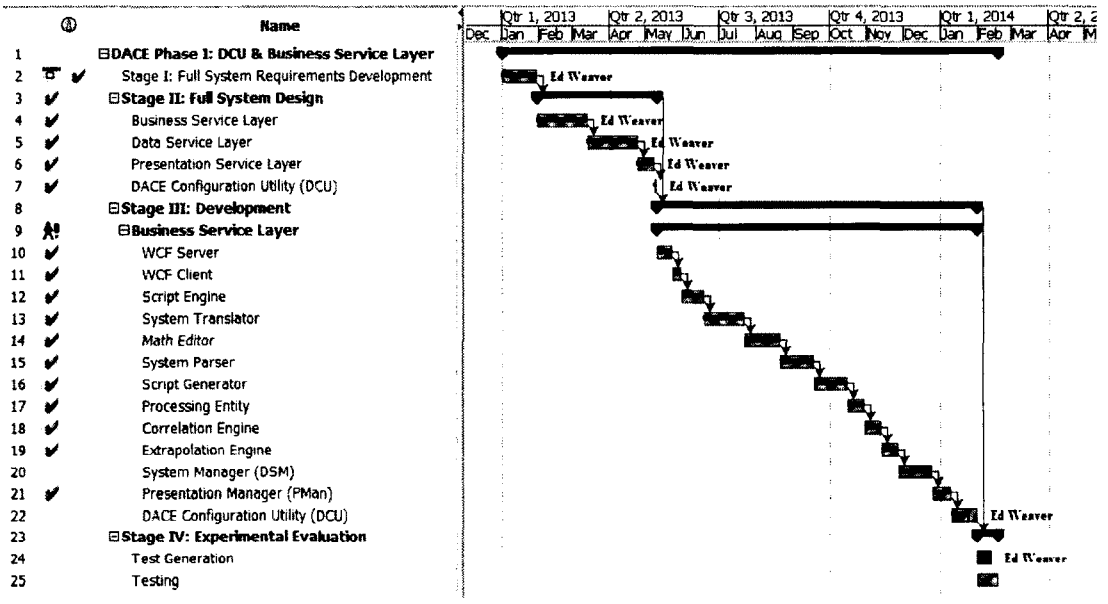



**Figure 25. DACE Full Schedule**

### 3.4.1 Stage I - System Requirements

The first stage is where the system requirements definition occurred. This entailed the definition of the proposed system requirements to be identified and categorized. The categorization process defined the 12 variable elements and the full system design phases of development. Although the high level system requirements were adhered too, the process of lower lever requirements refinement occurred throughout the project's life cycle.

### 3.4.2 Stage II - Design

The second stage defined the full system design. The full system design needed to be developed from the beginning; that allowed an appropriate software architecture to be defined. The choice was narrowed down to three options: peer-to-peer, layered architecture, and the Three Tier Architecture. The decision for the Three Tier

Architecture design was driven by the requirements and the definition of the 12 variable elements.

The DACE framework is built around a client-server architecture, but does deploy some additional functionality from other architectural methodologies. Within the Business Service Layer, the components have some peer-to-peer architecture qualities due to the fact that all components can act as both consumers and suppliers. A consumer is an object that receives or consumes messages or data and performs an action. Suppliers are objects that produce and transmit a message or data to a consumer object. This layer also can be configured and perform very similar to the Layered Architecture where the different PEMs can provide layered operations that are passed down or up to the next PEMs in the configuration. This scenario is similar in nature to the network stack or Java virtual machine functionality. Once the architecture was selected, the details of the three service layers where developed to align back to the 12 variable elements. This information was then used to generate and DACE Infrastructure Relationship Matrix in Appendix C.

### 3.4.3 Stage III - Development

The purpose of this section is provide an overview of how the system was developed and relates to the DACE schedule in Figure 25. The system development was broken out into the two focus layers of the architecture, the Business and Presentation Layer. Table 4 provides information relative to the level of effort of the DACE development. The table shows module lines of code, man hours expended, and the effective man-month equivalent.

**Table 4. DACE Development Level of Effort**

| Modules | | Lines of Code | Man Hours | Man Months |
|---|---|---|---|---|
| Level 1 | Level 0 - Support Modules | | | |
| DCU | WCF Client | 680 | 218 | 1.36 |
| | Script Engine Integration | 121 | 39 | 0.24 |
| | System Translator | 694 | 222 | 1.39 |
| | Math Editor | 2,143 | 686 | 4.29 |
| | System Parser | 948 | 303 | 1.90 |
| | Script Generator | 1,432 | 458 | 2.86 |
| | Function & Integration Code | 378 | 151 | 0.12 |
| | **Tally:** | **6,396** | **2077** | **12.16** |
| Pman | WCF Server | 183 | 59 | 0.37 |
| | WCF Client | 680 | 218 | 1.36 |
| | Function & Integration Code | 83 | 27 | 0.17 |
| | **Tally:** | **946** | **303** | **1.89** |
| Dman | WCF Server | 183 | 59 | 0.37 |
| | WCF Client | 680 | 218 | 1.36 |
| | Function & Integration Code | 83 | 27 | 0.17 |
| | **Tally:** | **946** | **303** | **1.89** |
| PEMs | WCF Client | 680 | 218 | 1.36 |
| | Script Engine Integration | 121 | 39 | 0.24 |
| | Function & Integration Code | 234 | 75 | 0.47 |
| | **Tally:** | **1,035** | **331** | **2.07** |
| DSM | PEMs | 1,035 | 331 | 2.07 |
| | Correlation Engine | 756 | 242 | 1.51 |
| | Extrapolation Engine | 509 | 163 | 1.02 |
| | Function & Integration Code | 413 | 132 | 0.83 |
| | **Tally:** | **2,713** | **868** | **5.43** |
| Core | DSM | 2,713 | 868 | 5.43 |
| | WCF Server | 183 | 59 | 0.37 |
| | Function & Integration Code | 178 | 57 | 0.36 |
| | **Tally:** | **3,074** | **984** | **6.15** |
| **DACE Phase I Totals:** | | **9,553** | **3,087** | **18.47** |

#### 3.4.3.1 Presentation Service Layer

The Presentation Service Layer was the first section to be developed. This consisted of, in development order, the System Parser, Script Generator, Math Editor, and System Translator modules. The development of this layer was done based on the order of operational need. The System Parser and Math Editor modules were done in parallel, which lead into the development the Script Generator.

The System Parser which essentially takes in the system configuration parameters from either an XML or Excel file (see Figure 26) and evaluates them to determine the system boundaries. This means that is parses the system out into computer configurations and their corresponding information for each computer in the system. System configuration parameters can be found in Appendix C: DCU Configuration Parameters.

Since this module was the main interface to implementing the proposed system, care was taking in the implementation to make sure to catch any load errors. This module not only organizes all the system components and aligns their functional requirements; it also does system configuration validation. This was necessary to make the overall architecture more reliable and provide a feedback mechanism to end users when there is a system design issue.

| Number of Computers: | 3 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Configuration ID: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Computer ID: | 1 | 2 | 2 | | | | | | | |
| Computer IP: | 192.168.1.10 | 192.168.1.11 | 192.168.1.11 | | | | | | | |
| Computer OS: | Win7 | OpenSuse | OpenSuse | | | | | | | |
| CPU Cores: | 1 | 1 | 1 | | | | | | | |
| Threads: | 1 | 1 | 1 | | | | | | | |
| System Component ID: | 4001 | 4002 | 4003 | | | | | | | |
| Computer Components | DCU | PEMs | PEMs | | | | | | | |
| | Pman | DCU | | | | | | | | |
| | DSM | | | | | | | | | |
| | WCFS | | | | | | | | | |
| | Dman | | | | | | | | | |
| | PEMs | | | | | | | | | |

**Figure 26. DACE Configuration Worksheet**

The System Translator is the module that takes mathematical formulas that have been provided in Excel and creates a mechanism to convert the formula into C# code for utilization in the DACE system. This module essentially matches all standard mathematic operations in the same form as Excel. This mechanism provided a quick way to perform mathematical functions and do one-to-one mappings in C# code.

The Script Generator was developed next. Upon initial development of this module, it came clear that it needed to support two main purposes. The first purpose was to take the application logic that was cached from the System Translator operation and convert this into executable C# code. The second purpose was to create the installation packages for the machines that components would be installed on.

The process of creating installation packages had a few challenges that were faced along the way. The biggest challenge encountered was making sure to group the required

technologies into the build environment. This first phase effort essentially supported only Windows and openSUSE installations. These configuration package parameters were hard coded, but a more flexible and sustainable method would be to provide OS specific information files and information that the module could dynamically load. This information would then be utilized to create the installation package as required.

Since this phase was not intended to provide a high end graphical user interface, the System Translator and the Math Editor were not developed to their fullest extent. Both modules were developed to allow data input to provide the correct format of information to the System Parser.

A DCU user interface was developed to wrap the modules into one application. Due to the scope of the project, the graphical configuration mechanism was not implemented. A generic dashboard was created that automates the build process and shows the status. Table 5 provides information to the development level of effort for the DCU and supporting modules that it uses. The table shows module lines of code, man hours expended, and the effective man-months to each development effort.

**Table 5. DCU Development Effort**

| Modules | | Lines of Code | Man Hours | Man Months |
|---|---|---|---|---|
| Level 1 | Level 0 - Support Modules | | | |
| DCU | WCF Client | 680 | 218 | 1.36 |
| | Script Engine Integration | 121 | 39 | 0.24 |
| | System Translator | 694 | 222 | 1.39 |
| | Math Editor | 2,143 | 686 | 4.29 |
| | System Parser | 948 | 303 | 1.90 |
| | Script Generator | 1,432 | 458 | 2.86 |
| | Function & Integration Code | 378 | 151 | 0.12 |
| | **Tally:** | **6,396** | **2077** | **12.16** |

3.4.3.2 Business Service Layer

The Business Service Layer was the next section to be developed. This consisted of, in development order, the Scripting Engine, the WCF Server, the WCF Client, the Processing Entity Module, the Core, the Correlation Engine, and the Extrapolation Engine. The development of this layer was done based on the bottom up approach. Since the WCF Server is the backbone of this layer, it was developed first. The corresponding WCF Client module was then developed and tested for all modes of communications.

The WCF server and client were developed in C# utilizing the .NET framework. Two projects were created that would utilize both Named Pipes and the TCP communication mechanisms. Since both Windows 7 and openSUSE supported TCP, this was developed first. Using .NET WCF both the server and client modules were created. Based on the concept of general usability, a data distribution mechanism needed to be identified to be able to pass any type of data within the system.

The PEMs was the next component to be developed but the biggest challenge was the Scripting Engine module. A base scripting engine was developed but the first

generation scripting engine lacked some of the dynamic capabilities that were needed for the design. After researching scripting engines and evaluating the level of effort to implement the full capabilities required in the DACE framework, an open source C# scripting engine called CS-Script was selected for this project. This saved time while providing the module functionality required in the development and evaluating the proposed framework. The PEMs module was then developed to integrate the component control logic with the WCF client and the scripting engine.

The Core was the next component to receive focus, to align the layer's components to the functional requirements. Since both the PEMs and the WCF Server were already developed, the focus was on the administrative system logic, the Correlation Engine, and the Extrapolation Engine. The general administrative logic consisted of implementing status operations and system component heartbeat mechanisms. This allows the DSM to have a fundamental concept of component health. Although this is not a holistic feature by any means, it does provide low administrative monitoring capability of the system components. Future development will provide a Health Monitoring messaging structure to be handled by system components.

The Correlation Engine and the Extrapolation Engine models were then developed. Both of these modules turned out to be similar in functional requirements. Both needed to have a base capability where information collected needed a definition or identification of a relationship to another piece of information. For example, to correlate information one needs to either know what the rules are (procedure or function) or what information to watch based on other information identified as input variables, and what information to watch as output variables. So a common model was developed to support

a simple model that required variable inputs function and output designation. The Correlation Engine would then essentially use a signal to noise methodology to wiggle input variables and map the output variance looking for strong and weak bonds. For the Extrapolation Engine, the same mind set was utilized. Either time specific identifiers could be used to extrapolate a given data set or the identification of specific information could be used. This structure provided flexibility to add more advanced methods while providing a base capability within the system.

# CHAPTER 4

# EXPERIMENTAL EVALUATION

The DACE framework contains 12 variable elements; this produces 4096 test combinations for every possible system definition that could be defined. To evaluate the Phase I systems framework development, the system deployment in Figure 27 was implemented.

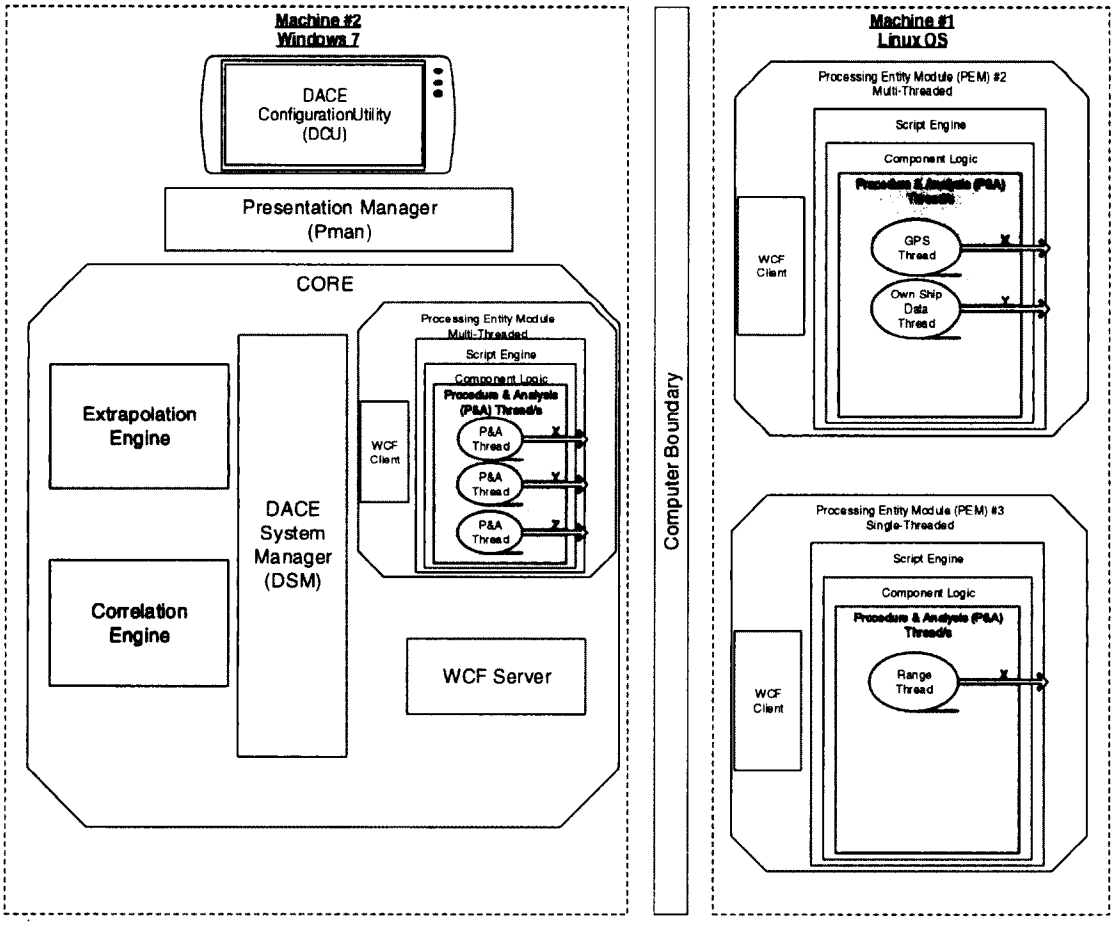



**Figure 27. Project Scope**

## 4.1 Implementation

The evaluation scenario which aligns with the system configuration chosen is based off of a general data acquisition design. The system requirement is detailed in the following sections.

### 4.1.1 System Problem – Target Interrogation

A customer has two sensors that he wants to use to increase his situational awareness on a boat but wants to provide an automated solution. The first system is a highly accurate legacy GPS. The second system is a target acquisition camera that utilizes laser range finder to provide situational awareness. This camera, once fixed on a target, tracks the target automatically and uses the laser to determine the range to the target. Both sensors are connected to a computer with a Linux operating system and the customer wants the data displayed on a second computer that has a Windows 7 operating system. The customer also wants to determine their position and does not have radars on his small crafts. He does not want to pay and wait for a custom application to be developed and would like to have a prototype.

### 4.1.2 DACE Component Configuration Requirements

Based on the problem the requirements have been documented and mapped to the DACE system components (see Table 6). There are many ways to implement this scenario and the one depicted is not considered the optimal. It does show the flexibility in design considerations. The system could have been defined using one PEMs and all components except the DCU could have resided on the Linux computer.

**Table 6. DACE Configuration Requirements**

| ID | Requirement | Dace Component | OS |
|---|---|---|---|
| 1 | Acquire GPS data on Computer with Linux OS | PEMs #2 | Linux Computer #1 |
| 2 | Calculate approximate own ship speed | PEMs #2 | |
| 3 | Calculate approximate own ship heading | PEMs #2 | |
| 4 | Acquire range information from camera on Linux OS Computer | PEMs #3 | |
| 5 | Calculate targets current GPS location | Correlation Engine | Window 7 Computer #2 |
| 6 | Calculate targets speed | Correlation Engine | |
| 7 | Calculate targets heading | Correlation Engine | |
| 8 | Show Target Information on 2 computer with Windows 7 OS | DCU | |

The test case that was implemented within DACE, was generated using the DCU. The code was separated into two separate modules. The first module was generated was for Computer #1. The second module was generated for Computer #2 with a Windows 7 operating system. Table 7 shows a cross mapping of the components utilized in the test case against the technologies utilized in the DACE design.

**Table 7. Test Scenario Technology Map**

| | | Components | | Technology | | | |
|---|---|---|---|---|---|---|---|
| | | Computer #1 openSUSE | Computer #2 Windows 7 | .NET | Mono | C# | WCF |
| Modules | DCU | | X | X | | X | |
| | Pman | | X | | X | X | |
| | System Manager (DSM) | | X | X | | X | |
| | Correlation Engine | | X | | X | X | |
| | Extrapolation Engine | | X | | X | X | |
| | WCF Server | | X | X | | X | X |
| | WCF Client | X | | X | X | X | X |
| | PEMs | X | X | X | X | X | |

The Linux machine has two CPU's and two PEMs. PEMs two, focuses on the acquisition of GPS data and calculation of own ship's speed and heading. PEMs number three, focused on the acquisition of target range data from the camera system. Since receiving time critical information resides on the same machine, each PEMs will be configured to run on a separate CPU.

On the Windows 7 machine, even though it is a dual core CPU, the DCU and the Pman will execute on the same core. Table 8 shows additional details of the 12 Variable Elements configuration for this scenario.

**Table 8. Test Scenario Variable Elements Options**

| Component | Computer | OS (W=Win 7, L=Linux) | Model (D=Distributed, L=Local) | Process (Threads) (S=Single, M=Multi) | Processing (Cores) (S=Single, M=Multi) | Application (E=Executable, S=Service) | Data (L=Live or C=Captured) | Correlation (X = Active) | Extrapolation (X = Active) | Storage | Health (X = Active) | Metrics (X = Active) | Security (A=Access Control, E=Encryption, R=Redundancy Check) | Reliability (M=Message Delivery, R=Redundancy) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DCU | 2 | W | D | S | S | E | | | | | | | | M |
| Pman | 2 | W | D | M | S | S | | | | | X | X | A | M |
| Core | 2 | L | L | M | M | E | | | | | X | X | | M |
| WCF Server | 2 | L | L | M | M | E | | | | | X | X | A | M |
| PEMs #2 | 1 | L | L | M | M | E | C | X | | | X | X | | M |
| PEMs #3 | 1 | L | L | S | M | E | C | X | X | | X | X | | M |

4.1.3 Test System Configuration

The actual test was implemented using a Dell Precision T1500 computer. The CPU was an Intel i3 Dual Core, 3.07 GHz processor with 2.99 Gigabits of Random Access Memory (RAM) and a 32 bit Windows 7 Professional Operating System. The test was performed while the following applications, in Table 9, were running.

**Table 9. Applications running during test**

| | |
|---|---|
| McAfee Total Protection | FireFox v25.0.1 |
| Microsoft Visual Studio 2010 | Internet Explorer 10 |
| Microsoft Word 2010 | Microsoft Excel |
| Microsoft Visio 2013 | iTunes |
| Microsoft Visual Sourcesafe 6.0 | Oracle VM VirtualBox |

The test scenario defined Windows components, were executed directly on a Windows 7 based machine. The test scenario Linux components, were executed on an openSUSE Linux Virtual Machine (VM) that was running on the same test computer. In this configuration, it is understood that there would be a performance penalty of the configuration under the test due to the performance hit of running on a VM and the VM residing on the same machine. Figure 28 shows a graphical representation of the test scenario implementation.

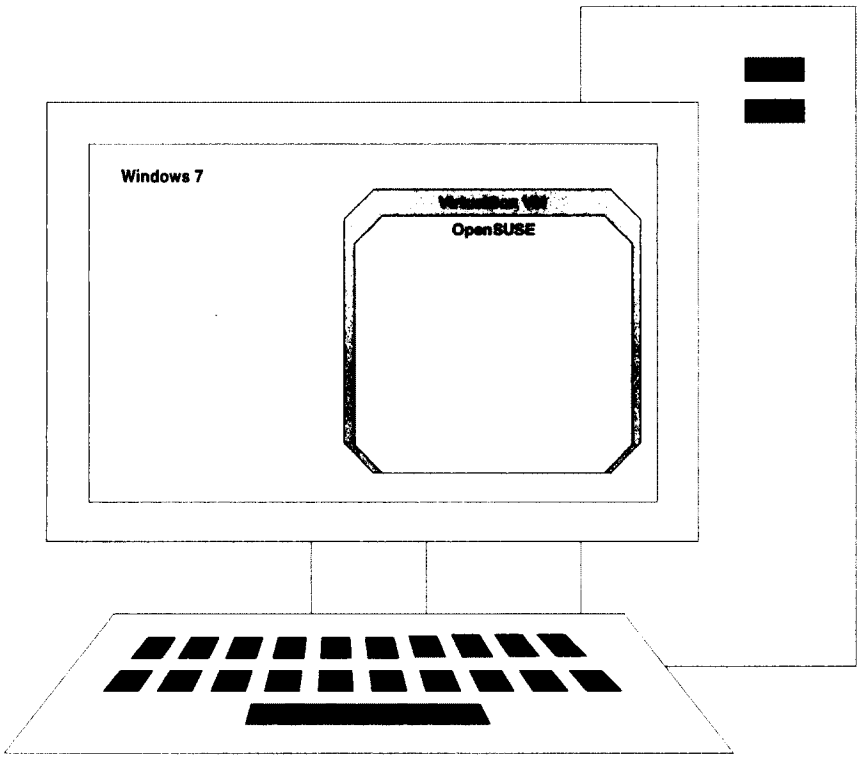



**Figure 28. Test System Configuration**

## 4.2 Experimental Evaluation

The "Target Interrogation" test scenario was implemented (see Figure 27) and tested based on the selections and configuration covered in Table 8. The test information computed is based the Design Structure Quality Index, DACE Application and Implementation Characteristics, and System Performance Characteristics. This information was then evaluated against the same problem developed using a typical development methodology.

### 4.2.1 System Configuration Metrics

#### 4.2.1.1 DACE Application and Implementation Characteristics

The test case that was implemented within DACE, was generated using the DCU and showed acceptable performance characteristics, depicted in Table 10. The test scenario design produced 463 lines of application code. The test scenario, upon execution, started up in 5.66 seconds (s) and the actual functional execution of the test case 66 milliseconds (ms) into the DACE engine. The test case then took approximately 129.6 ms to fully execute its process.

**Table 10. Test Design Implementation Characteristics**

| Measurement | Computed |
|---|---|
| Total Lines of Code Utilized | 463 |
| Execution Time (ms) | 129.6 |
| Total Application Load Time(s) | 5.66 |
| Script Load Time (ms) | 66 |
| System Size (KB) | 1111 |

The DACE test case system consisted of over 14,700 lines of code (see Table 11). The test case specific functional code produced equaled 183 lines of code. If this design was written in a monolithic application and assuming that there is a 40% gain in efficiency by reworking some of the DACE modules to be application specific, the total lines of code utilized to implement the same system would be approximately 8,830 lines of code. This level of effort of development effort would have impacts to both schedule and cost.

**Table 11. Test Scenario Level of Effort**

| Modules | Computer #1 openSUSE Lines of Code | Computer #2 Windows 7 | System Code Tally: |
|---|---|---|---|
| DCU | | 6,396 | 6,396 |
| Pman | | 946 | 946 |
| System Manager (DSM) | | 2,713 | 2,713 |
| Correlation Engine | | 756 | 756 |
| Extrapolation Engine | | 509 | 509 |
| WCF Server | | 183 | 183 |
| WCF Client | | 680 | 680 |
| PEMs | 2,070 | | 2,070 |
| Test Scenario | 463 | | 463 |
| **Test Scenario Totals:** | **2,533** | **12,183** | **14,716** |
| 40% Efficiency Gain | 1520 | 7310 | 8830 |

4.2.1.2 Design Structure Quality Index (DSQI)

To evaluate the efficiency and structure of the test scenario design, the Design Structure Quality Index method was used. Table 12 provides the parameters that were used to calculate the DSQI. The input variables that were of focus based on the test scenario and the DACE architecture were the total number of modules in the architecture,

the number of modules that depend on data input and the number of modules with only one entry and exit point. Since the focus was on the application structure and module independence, a higher weighted value was used. Module entrance and exit values play a strong role in the DSQI. The higher the value for this module entrance and exit characteristics means that the system is more venerable to cyber-attacks. Although this is of concern, it is not the intention of this phase to harden the system against these types of threats. Based on the program architecture and the values and weights entered, module independence induced the most DSQI variance resultant.

**Table 12. DACE Phase I Design Structure Quality Index**

| Variables | Values | Measurement | Weights |
|---|---|---|---|
| Total number of modules in the architecture | 15 | Program Structure | 30 |
| Number of modules that depend on data input or produce data to be produced in another module | 4 | Module Independence | 30 |
| Number of modules who depend on prior processing | 2 | Modules not dependent on prior processing | 30 |
| Number of database items | 0 | Database size | 0 |
| Total number of unique database items | 0 | Database compartmentalization | 0 |
| Number of database segments | 0 | Module entrance and exit characteristics | 10 |
| Number of modules with only 1 entry and exit point within the module | 7 | | |
| | | | 100.0 |
| | | DSQI | 0.87333 |

### 4.2.2 System Test Results

The Target Interrogator test scenario showed the flexibility within the Phase I DACE system design. The test, although not exhaustive of the full frameworks capabilities, executed as expected with no issues for over 3 hours. During test execution, there were no indicators of any system performance issues. The computers' performance stayed reasonably within the values displayed in Figure 29. The CPU utilization stayed within a 10% margin. Upon investigation on the fluctuation using the operating systems Resource Monitor application, both FireFox and Internet Explorer were the main drivers to the fluctuation.

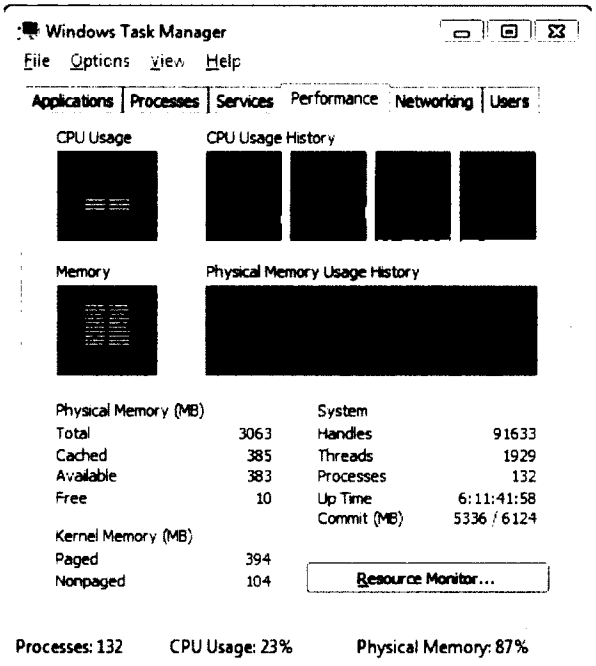



**Figure 29. Computer Performance Prior to DACE Execution**

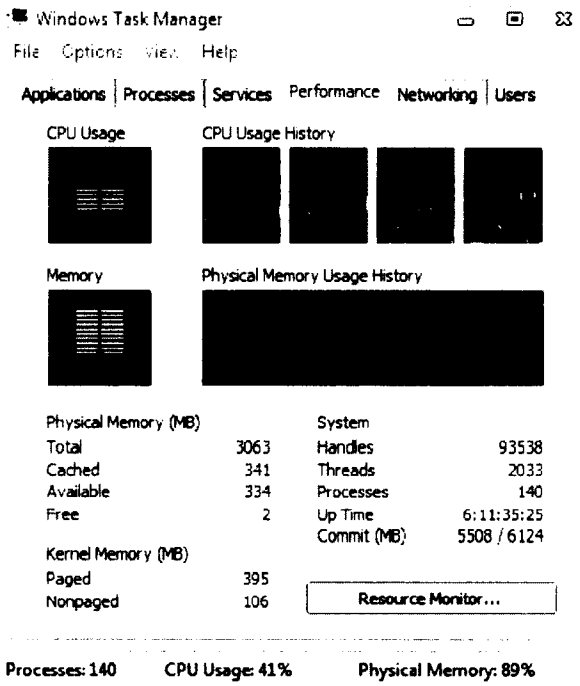



**Figure 30. Computer Performance during DACE Execution**

Upon execution, the solution took 5.66 s to load. This means that all modules were up and processing. Researching both .NET and Mono, showed that the frameworks have an overhead startup time of anywhere from 4 to 9 seconds on a typical software implementation. This is without any startup optimization techniques. During the test system start up, there was a system resource surge that occurred. This aligns with the .NET and Mono's CLR execution.

The performance of the DACE test scenario was compared to a Research & Development (R&D) project that contained the same functionality, see Table 13. This project was developed using mostly the ANSI C and C++ programming languages. If we are to take the stance that DACE tool has been developed and that the application specific code is the level of effort to implement a defined functionality then the DACE

implementation code was significantly less. Since the DACE tool was already available, there was no need to develop an architecture to support this effort.

**Table 13. System Implementation Characteristics**

| Measurement | R&D Project | DACE |
|---|---|---|
| Total Lines of Code Utilized | 2019 | 463 |
| Execution Time (ms) | 119.4 | 129.6 |
| Total Application Load Time(s) | 0.68 | 5.66 |
| Script Load Time (ms) | N/A | 66 |
| System Size (KB) | 2033.13 | 1111 |

The system execution time between the two solutions shows that the R&D project solution actually executed the specific functionality faster than the DACE solution. This outcome is partially due to the languages used and most likely the sensor data simulation execution time. The original project's development language is considered a lower level language which typically contains less overhead and executes faster than higher level languages. The DACE tool was developed in the C# language and runs in a virtual software machine which will have more execution management overhead.

The test sensor data were housed in a comma separated file. These data were accessed through the file and published or made available in fixed intervals. File access operations are known to have access overhead to them which may have affected the data availability to the system.

The application load time between the two systems was significant. The load time of the DACE system was much greater than on the original implementation. The increased load time is due to the application startup and running within essentially a

software execution virtual machine in Mono. There are optimization techniques that can be implemented to gain efficiencies but they have not been addressed at this point in time.

The final item of comparison is the system size or size of the code that was generated and utilized. The original project was over 80% larger in size then the DACE implementation. This was mostly due to the Windows User Interface that was developed to present the data. Graphical interfaces are typically larger in size then a generic shell window like what was utilized in the DACE test scenario.

This test scenario provided a view of the flexibility of the DACE architecture. The test also showed, given the defined architecture and conceptual usage possibilities, the DACE initial design was successful. The current limitation that did surface was the significant application load time and differences on the various operating system platforms. Additional testing and optimization techniques need to be evaluated to gain efficiencies in this area.

### 4.2.3 System Implementation Comparison and Evaluation

To evaluate the efficiency and magnitude of the test scenario design, the DACE test scenario was compared to a similar effort that was implemented using a typical R&D project. The overall life cycle of the project lasted 63 day or 11.8 weeks shown in Table 14. The projects life cycle consisted of requirements analysis, research, design, development, and system testing.

**Table 14. R&D Project Effort Data**

| Phase | Time Frame (Working Days) | Weeks | Approximate Cost |
|---|---|---|---|
| Requirements Analysis | 3 | 0.6 | $4.5K |
| Research | 5 | 1 | $7.6K |
| Design | 8 | 0.8 | $12K |
| Development | 20 | 4 | $30.5K |
| Testing | 15 | 3 | $22.8K |
| Documentation | 10 | 2 | $14K |
| Close Out | 2 | 0.4 | $3K |
| **Totals:** | **63** | **11.8** | **$94.4K** |

To compare this effort with the DACE test scenario and provide a one to one development analysis, it was determined that the major items effected based on the current state of the design, where the design and development phases of the original effort. The general evaluation approach was to assume the current version of the DACE tool was available to be utilized for the R&D project effort. This means that the typical project phases and administration would still apply.

The implementation of the DACE scenario required a determination of the 12 Variable Elements of configuration that would align to the required design requirements. Since the work could be distributed to multiple PEMs in many configurations, the choice to provide a three PEMs approach, aligned with the R&D project implementation as far as physical configuration. This decision was determined in two hours and aligns with the Design phase in the R&D projects life cycle. This phase took the team eight working days due to the evaluation of the architectures that could have been deployed.

The next major impact area for the evaluation of the DACE design against the R&D project was the Development phase. This phase took the design team a total of 20 working days to complete. This was due to the creation of the architecture for the system

which contained modules for both Linux and Windows. This was needed before even the scenario specific algorithms could be addressed. This design was implemented by two software engineers. The DACE test scenario "Development" effort consisted only of the development of the scenario specific algorithms which consisted of 463 lines of code. This took a total of six hours to develop, test, and debug.

Table 15 shows the differences between the R&D project and the DACE test. It provides insight to where the impacts would have occurred if the DACE tool could have been utilized within the project instead of the typical system development methodology. Comparisons of the two efforts shows a large efficiency gain in the level of effort needed to implement the system. This then translates to a schedule savings of 27 working days and a cost savings of approximately $41 thousand dollars. This provides a huge value to a project and customers success.

**Table 15. R&D Project Implementation Comparison**

| | **R&D Project** | | | **DACE** | | |
|---|---|---|---|---|---|---|
| **Phase** | **Time Frame (Working Days)** | **Weeks** | **Approximate Cost** | **Time Frame (Working Days)** | **Weeks** | **Approximate Cost** |
| Requirements Analysis | 3 | 0.6 | $4.5K | 3 | 0.6 | $4.5K |
| Research | 5 | 1 | $7.6K | 5 | 1 | $7.6K |
| Design | **8** | **0.8** | **$12K** | **0.25** | **0.00625** | **$0.4K** |
| Development | **20** | **4** | **$30.5K** | **0.75** | **0.01875** | **$1.1K** |
| Testing | 15 | 3 | $22.8K | 15 | 3 | $22.8K |
| Documentation | 10 | 2 | $14K | 10 | 2 | $14K |
| Close Out | 2 | 0.4 | $3K | 2 | 0.4 | $3K |
| Totals | **63** | **11.8** | **$94.4K** | **36** | **7.025** | **$53.4K** |
| Savings | | | | **27** | **4.775** | **$41K** |

Upon further maturity of the DACE framework, both the Documentation and Testing phases could also be impacted as well. Based on the idea that the DACE general architecture documentation will be available, only the specific generated system and the application specific functionality would need developed for the design. A documentation standard template could also be developed that could provide system implementers a head start on their projects documentation efforts.

System testing is always a critical phase of a projects life cycle, whether talking about unit, integration, verification or even final validation testing. An efficiency gain could be gained in this area as well, based on the fact that the DACE framework would have already been and continues to be tested during each development effort. This means that every possible customer benefits on testing and bug fixes based on a common tool mentality. As the system matures, based on the level of testing and implementations fielded, a higher level of quality assurance will be achieved. This translates to lower levels of life cycle costs and system failures due to implementation errors.

# CHAPTER 5

# CONCLUSIONS

This project's focus was to develop a framework that could be used to develop a Common Knowledge Engineering Framework for Data Assimilation, Correlation, and Extrapolation application that would allow end users to define or generate their preferred methodology for analysis. The applicability of such a design provides a very good base for analyzing complex situations that are driven by multiple criteria.

## 5.1 Conclusions

The purpose of this project was to provide a common software framework to address data assimilation, correlation, and extrapolation. The intent was for this tool to become a foundation application for complex system design and integration efforts. Based on the current design state of this phase and the limited testing that has been done, the implementation of this framework was successful in aligning with the objectives. The comparison between a R&D project and the DACE test scenario showed significant efficiencies in a project schedule and cost. This provides significant value to customers which allows them to address complex problems in a cost effective manner. It also provides a strong mechanism to get approval and funding from sponsors on technical problems that may seem extremely complex or push the current feasibility envelope.

The framework would have significant impact to system development efforts. The framework was developed in a modular and configurable architecture, and able to support the three main focus areas of data assimilation, correlation, and extrapolation in one

cohesive application. This ability allows the framework to be deployed in a variety of situations.

When it comes to complex system designs, there is always a gap that needs to be addressed for component integration. Often components have different types of interfaces and communication mechanisms. The DACE framework would be able to be the "glue" or translation element for these components. The translation element would be achievable without writing custom code, which is essentially the current process, by defining the components communication schema and the information that needs to be translated. This is important because the system developer may not have the ability to modify the components within the system without providing Non-Recurring Engineering (NRE) funding to the original manufacturer.

There is often a need to have the ability to analyze data, whether it is live or historical in nature, utilizing multiple methods and performing cross correlation to the results of each analysis method to provide a final output model. The DACE framework would be designed to support this type of analysis fusion. This provides value to academia and industry, by providing a common mechanism where analysis methodology or methodologies is defined by the end user for a specific focus. For example, a system needs to be analyzed for an IRAD project to determine the impact to a company's portfolio. It has been determined that the following methods would be utilized for this effort: Failure Modes and Effects Analysis (Reliability), Fault Tree Analysis (Reliability), Life-cycle Analysis (AKA Life-Cycle Assessment), and Value Chain Analysis (Firm Level). Within the DACE framework each of these methods could be analyzed in their own separate process. The results of each analysis would then, based on the users defined

cross relationship rules, be correlated together to provide a final solution set to the defined scenario.

These situational application areas provide value to both the engineering management and systems engineering disciplines. With the DACE framework, significant efficiencies could be gained in both schedule and cost of a projects effort.

Engineering Management, from a federal market perspective, has a heavy focus on the cost of an overall product development effort. The utilization of a common application to integrate a system provides a significant cost savings relative to reduced schedule and/or to resource allocation costs of developing a custom application to interface components. The cost savings gained could even be across multiple projects. This savings could translate to quicker break-even scenarios allowing companies to start making profits on their designs, faster. If the project is able to leverage a reduction in schedule based on not having to develop custom interfaces, this also could allow for faster time to market.

System engineers will also gain significant benefit on projects that require quick prototypes, feasibility studies, complex analysis methodology, and custom analysis methods. This could possibly provide them additional justification to get buy-in for internal research and development projects based on a common tool which does not require a long programming cycle or purchase of a different tool for each IRAD exercise.

To address "System of System" or highly complex problems, a tool with this type of capability could be used. Even if the design does not predict or solve every issue, it should at least provide a mechanism for moving forward on additional tool set capabilities to address these types of situations.

## 5.2 Future Direction

This project, Phase I, focused on the Business Service Layer and interfacing modules which are the heart of the system. Additional testing and two more phases need to occur to complete the full DACE framework. Since there are over 4,096 quantifiable test cases based on the 12 variable elements defined, a fairly defined subset of tests has been defined (see Figure 31). Both phases provide a significant development challenge. Both will have to support generic interfacing and expandability.

| | Test | OS (Core) | Components | Threading | Execution Type | Exe Mode | Model | Metrics | Data | Correlation | Extrapolation | Storage |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Single machine, just the Core | 1 | Windows | single | single | Executable | Debug | local | Enabled | Captured | Enabled | Enabled | Single |
| | 2 | Windows | single | multiple | Executable | Debug | local | Enabled | Captured | Enabled | Enabled | Single |
| | 3 | Windows | single | single | Executable | Release | local | Enabled | Captured | Enabled | Enabled | Single |
| | 4 | Windows | single | multiple | Executable | Release | local | Enabled | Captured | Enabled | Enabled | Single |
| | 5 | Linux | single | single | Executable | Debug | local | Enabled | Captured | Enabled | Enabled | Single |
| | 6 | Linux | single | multiple | Executable | Debug | local | Enabled | Captured | Enabled | Enabled | Single |
| | 7 | Linux | single | single | Executable | Release | local | Enabled | Captured | Enabled | Enabled | Single |
| | 8 | Linux | single | multiple | Executable | Release | local | Enabled | Captured | Enabled | Enabled | Single |
| | 9 | Windows | single | single | Service/Daemon | Debug | local | Enabled | Captured | Enabled | Enabled | Single |
| | 10 | Windows | single | multiple | Service/Daemon | Debug | local | Enabled | Captured | Enabled | Enabled | Single |
| | 11 | Windows | single | single | Service/Daemon | Release | local | Enabled | Captured | Enabled | Enabled | Single |
| | 12 | Windows | single | multiple | Service/Daemon | Release | local | Enabled | Captured | Enabled | Enabled | Single |
| | 13 | Linux | single | single | Service/Daemon | Debug | local | Enabled | Captured | Enabled | Enabled | Single |
| | 14 | Linux | single | multiple | Service/Daemon | Debug | local | Enabled | Captured | Enabled | Enabled | Single |
| | 15 | Linux | single | single | Service/Daemon | Release | local | Enabled | Captured | Enabled | Enabled | Single |
| | 16 | Linux | single | multiple | Service/Daemon | Release | local | Enabled | Captured | Enabled | Enabled | Single |
| 3 machines, Core and 2 PEMs 2 PEMs Windows 2 PEMs Linux a. Windows PEMs are on the same machine b. Linux PEMS are on on the same machine | 17 | Windows | single | single | Executable | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 18 | Windows | single | multiple | Executable | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 19 | Windows | single | single | Executable | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 20 | Windows | single | multiple | Executable | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 21 | Linux | single | single | Executable | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 22 | Linux | single | multiple | Executable | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 23 | Linux | single | single | Executable | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 24 | Linux | single | multiple | Executable | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 25 | Windows | single | single | Service/Daemon | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 26 | Windows | single | multiple | Service/Daemon | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 27 | Windows | single | single | Service/Daemon | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 28 | Windows | single | multiple | Service/Daemon | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 29 | Linux | single | single | Service/Daemon | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 30 | Linux | single | multiple | Service/Daemon | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 31 | Linux | single | single | Service/Daemon | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 32 | Linux | single | multiple | Service/Daemon | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| 5 machines, Core and 2 PEMs 2 PEMs Windows 2 PEMs Linux a. Windows PEMs are on the different machine b. Linux PEMS are on on the different machine | 33 | Windows | single | single | Executable | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 34 | Windows | single | multiple | Executable | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 35 | Windows | single | single | Executable | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 36 | Windows | single | multiple | Executable | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 37 | Linux | single | single | Executable | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 38 | Linux | single | multiple | Executable | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 39 | Linux | single | single | Executable | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 40 | Linux | single | multiple | Executable | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 41 | Windows | single | single | Service/Daemon | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 42 | Windows | single | multiple | Service/Daemon | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 43 | Windows | single | single | Service/Daemon | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 44 | Windows | single | multiple | Service/Daemon | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 45 | Linux | single | single | Service/Daemon | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 46 | Linux | single | multiple | Service/Daemon | Debug | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 47 | Linux | single | single | Service/Daemon | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |
| | 48 | Linux | single | multiple | Service/Daemon | Release | distributed | Enabled | Captured | Enabled | Enabled | Single |

**Figure 31. Additional System Test Cases**

The next phase of the development will focus on the Common Data Engine within the Data Service Layer. This will provide a generic interface into multiple types of data warehouses, which can be extendable. It will also require the ability to duplicate, share, and synchronize across multiple types of data warehouses. This phase will also have to address redundancy and fail safes for the Core, the WCF server, the Pman, and the Dman. Additionally, based on the premise of DACE, which is to provide a framework where any system design restrictions would be on the end users ability to implement the system not the frameworks inability to support, some additions and changes will be implemented. These are:

1. Providing a Business Layer interface that does not require Mono into the Pman.
2. Move the Extrapolation and Correlation engines into the Processing Entity component. This will allow for each PEMs to perform extrapolation and correlations prior to posting its data back to the Core Module. It will also allow for them to be more flexible on where they can reside within the physical system.

The third and last phase will focus on the DCU's graphical interface. It will provide a graphical system design canvas. The canvas will allow users to drag and drop system components from a toolbox and organize them in a fashion that suits the application. The result would look similar to the system diagrams provided in this paper. The end user will be able to either double click or right click on the object and configure the module as the design requires. This includes specifying component settings and even

defining the module's Procedures. Procedure development will have several options. These include:

1. Using a flow chart symbols.
2. Using a software development module that will be provided that allows programmers to develop their procedures in C#.
3. Editing the DACE Configuration Template directly within the DCU.

### 5.3 Lessons Learned

This project's major focus was to provide an analytic tool that provided various degrees of flexibility while not limiting the end user in its implementation. To do this required a lot of design considerations. The use of Agile development methods for iterative design considerations was necessary to align the design back to its requirements. This method was used during the design and development phases which resulted in modules evolving through multiple iterative changes. These changes led to schedule slippages and documentation rework, ultimately making the scope of the project much larger than anticipated. The full scope of this project was initially estimated to be equivalent to four man years. After looking back on the project and what still needs to be completed, the project resource allocation is estimated to be equivalent to eight man years of software engineering and testing.

Additional Lessons:

1. The level of application complexity increases as the level of application flexibility/usability increases.

2. Current technologies have significant capabilities; the key to developing something new with significant value is to keep up with the technologies and look for synergies to use them together to develop something that fulfills a need or gap.

# REFERENCES

Burgelman, R., Christensen, C. M., Wheelwright, S. C. (2004). *Strategic Management of technology and innovation.* New York, NY: The McGraw-Hill Companies, Inc.

Clemen. R. T (1996). *Making Hard Decisions: An Introduction to Decision Analysis.* Pacific Grove, CA: Brooks/Cole Publishing Company

Eisner, H. (2002). *Essentials of Project and Systems Engineering Management.* New York, NY: John Wiley & Sons, Inc.

Gheorghe, A. (2005). *Integrated Risk and Vulnerability Management Assisted by Decision Support Systems: Relevance and Impact on Governance.* Dordrecht, Netherlands: Springer

Kerzner, H. (2006). *Project Management: A Systems Approach to Planning, Scheduling, and Controlling.* Hoboken, NJ: John Wiley & Sons, Inc.

Phadke, M. S. (1989). *Quality Engineering Using Robust Design.* Upper Saddler River, NJ: P T R Prentice-Hall, Inc.

Royer, P. S. (2002). *Project Risk Management: A Proactive Approach.* Vienna, VA: Management Concepts, Inc.

Shilo, O. (2014). CS-Script (Version 3.8.2) [software]. Retrieved from http://www.csscript.net/

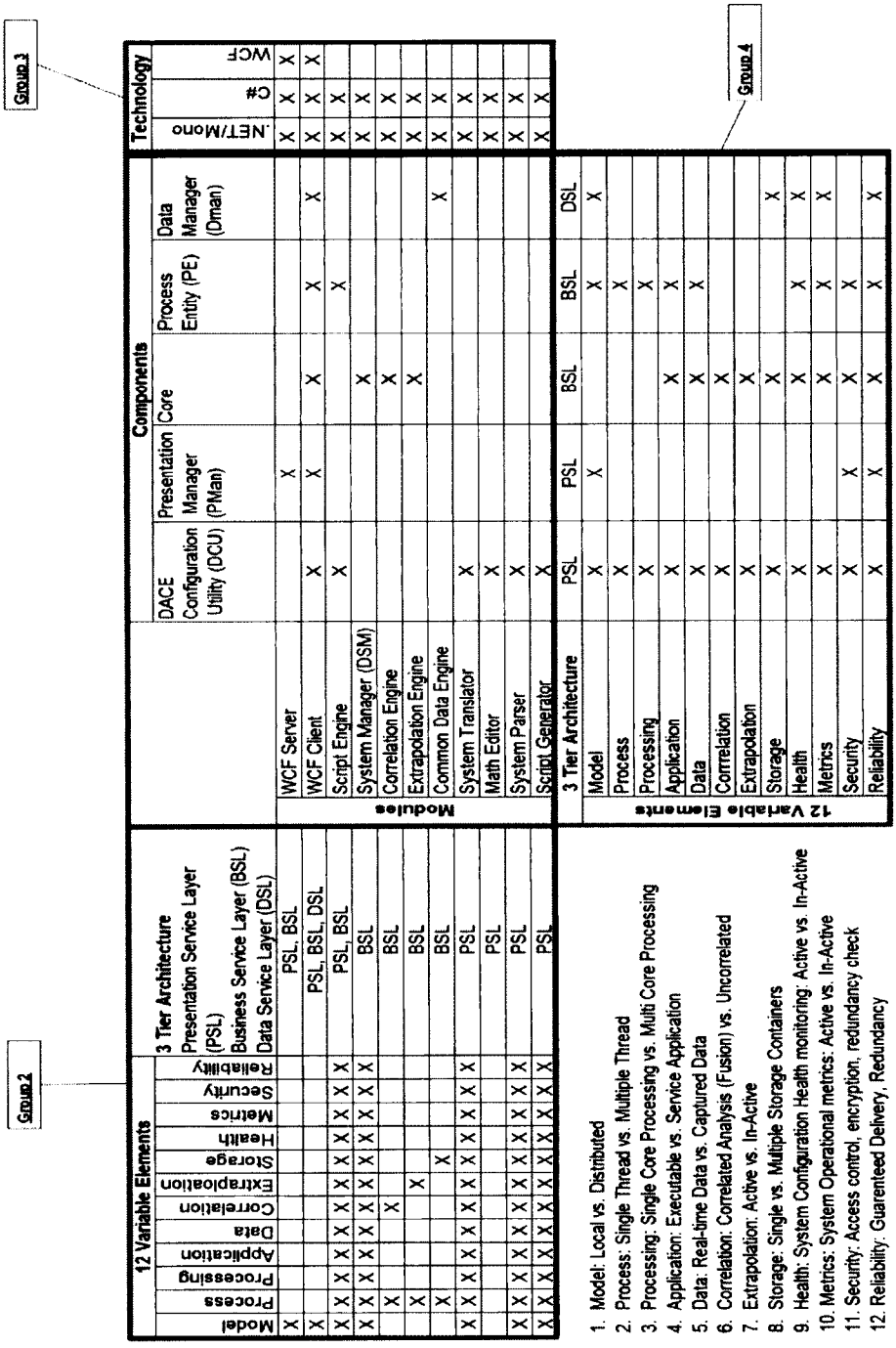
# APPENDICES

## APPENDIX A: DACE REQUIREMENTS TRACEABILITY MATRIX

**Table 16. DACE Requirements Traceability Matrix**

| *ID* | *Functional Requirement* | *Status* | *Software Component(s) or Module (s)* | **Implemented In** |
|---|---|---|---|---|
| 001 | Cross platform compatibility | | ALL | ALL |
| 002 | Ability for work flow logic to be single or multiple threaded | | PEMs | PEMs |
| 003 | Ability to perform data Correlation | | Correlation Engine | Correlation Engine |
| 004 | Ability to perform data Extrapolation | | Extrapolation Engine | Extrapolation Engine |
| 005 | Ability to store data in various forms | | Dman | Dman |
| 006 | Ability to provide system health status | | DSM | DSM |
| 007 | Ability to provide performance metrics | | ALL | ALL |
| 008 | Ability to provide access level authorization | | Pman | Pman |
| 009 | Ability to encrypt data | | WCF server and client | WCF server and client |
| 010 | Ability to transfer data | | WCF server and client | WCF server and client |
| 011 | Ability to provide reliable messaging | | WCF server and client | WCF server and client |
| 012 | Ability to interact with real time data | | PEMs | PEMs |
| 013 | Ability to utilize stored or captured data | | PEMs, | PEMs, |
| 014 | Ability to execute flow logic on a single or multiple cores | | PEMs | PEMs |
| 015 | Ability to support distributed configurations | | ALL | ALL |
| 016 | Ability to dynamically configure and update a system | | DCU | DCU |
| 017 | Ability to be expandable | | Pman, Dman, PEMs | Pman, Dman, PEMs |
| 018 | Ability to create executable applications | | DCU | DCU |
| 019 | Ability to create service applications | | DCU | |
| 020 | Ability to run in a debug mode | | PEMs | |

# APPENDIX B: DACE INFRASTRUCTURE RELATIONSHIP MATRIX




1. Model: Local vs. Distributed
2. Process: Single Thread vs. Multiple Thread
3. Processing: Single Core Processing vs. Multi Core Processing
4. Application: Executable vs. Service Application
5. Data: Real-time Data vs. Captured Data
6. Correlation: Correlated Analysis (Fusion) vs. Uncorrelated
7. Extrapolation: Active vs. In-Active
8. Storage: Single vs. Multiple Storage Containers
9. Health: System Configuration Health monitoring: Active vs. In-Active
10. Metrics: System Operational metrics: Active vs. In-Active
11. Security: Access control, encryption, redundancy check
12. Reliability: Guarenteed Delivery, Redundancy

**Figure 32. DACE Infrastructure Relationship**

# APPENDIX C: DCU CONFIGURATION PARAMETERS

1. Definition of client access
    a. Number of clients
        i. Each Client
            1. Definition of Client ID
            2. Definition of Client IP
            3. Definition of Client Operating System
            4. Definition of Client Access Level
            5. Definition of Data Encryption
            6. Definition of Data Cyclic Redundancy Check (CRC)
            7. Definition of Performance Metrics Reporting
2. Definition of number of components (Each component)
    a. Each Component
        i. Definition of Component ID
        ii. Definition of Components IP
        iii. Definition of Components Operating System
        iv. Definition of Execution Process (Threads)
        v. Definition of Processing (CPU Cores)
        vi. Definition of components operations
            1. Definition of input data
                a. Definition of data source
                b. Definition of data format

c. Definition of data types

2. Definition of operations on data

a. Definition of process

b. Definition of internal variables

3. Definition of output data

a. Definition of distribution rights

b. Definition of encryption option

c. Definition of storage options

4. Definition of Performance Metrics

3. Definition of Correlation Operations

a. Definition of input data

i. Definition of data source

ii. Definition of data format

iii. Definition of data types

b. Definition of operations on data

i. Definition of process

ii. Definition of internal variables

c. Definition of output data

i. Definition of distribution rights

ii. Definition of encryption option

iii. Definition of storage options

d. Definition of Performance Metrics

4. Definition of Extrapolation Operations

a. Definition of input data
    i. Definition of data source
    ii. Definition of data format
    iii. Definition of data types
b. Definition of operations on data
    i. Definition of process
    ii. Definition of internal variables
c. Definition of output data
    i. Definition of distribution rights
    ii. Definition of encryption option
    iii. Definition of storage options
d. Definition of Performance Metrics

# APPENDIX D: DACE ALIGNMENT WITH LEVELS OF INFORMATION SYSTEMS INTEROPERABILITY (LISI)

In 1993, it was recognized that there were different levels of technical interoperability within military departments and the systems did not interact well. The Levels of Information Systems Interoperability (LISI) was developed to provide a maturity model and process for determining overall joint interoperability requirements, assessing information systems against those requirements, and providing guidance for solutions and transition paths to meet those requirements. LISI is comprised of seven elements; the LISI Interoperability Maturity Model, LISI Reference Model, LISI Capabilities Model, LISI Implementation Options Tables, Interoperability Profile, LISI Metric, and LISI Products.

The LISI Reference Model is the pinnacle of the LISI process. It is designed to provide guidance towards compliance to technical characteristics supported by the Department of Defense (DoD). The reference model is essentially a lookup table where the rows refer to the five interoperability levels and the columns refer to defined attributes (see Table 17). The five levels of interoperability are Level 0: Isolated, Level 1: Connected, Level 2: Functional, Level 3: Domain, and Level 4: Enterprise. The four attributes are Procedural (P), Applications (A), Infrastructure (I), and Data (D).

**Table 17. LISI Reference Model**

| Description | Computing Environment | Level | P | A | I | D |
|---|---|---|---|---|---|---|
| Enterprise | Universal | 4 | Enterprise Level | Interactive | Multi-Dimensional Topologies | Enterprise Model |
| Domain | Integrated | 3 | Domain Level | Groupware | World-wide Network | Domain Model |
| Functional | Distributed | 2 | Program Level | Desktop Automation | Local Networks | Program Model |
| Connected | Peer-to-Peer | 1 | Local/Site Level | Standard System Drivers | Simple Connection | Local |
| Isolated | Manual | 0 | Access Control | N/A | Independent | Private |

Over the last four years, a major focus for me has been designing Unmanned Surface Vessels (USVs) for the Navy. The major challenge besides designing algorithms for machines to operate in various conditions as a human would operate the craft, is the ability to integrate systems. In this project, we are constantly evaluating new products. This often means a custom development effort to be able to integrate and utilize the system to test, collect and analyze the systems. There are essentially two types of possible solutions that we test, Government off the shelf (GOTS) and commercial designs. Both options require time and money. For GOTS systems, agencies and their contractors need to get involved to determine the feasibility, schedule, and cost to make the changes to integrate which costs significant money and time. For commercial systems, the biggest challenge is to get a company to agree to make changes for testing since there is no guarantee of future sales. One would deduce that the goal would be to create a plug and play system which is based on a mature standard that is global in nature. Even with this, which was done, it has not been enough. There are too many vendors outside of the

normal military focused market that have very good products that do not comply with standards within the maritime and robotic industry.

The DACE concept was designed to include the ability to function as a translator and interconnection agent between components, systems, and even system of systems designs. The construct was to address the common issue of interoperability between systems down at a mission critical warfighter level. This goal was not disjoint to the goal of providing a Common Knowledge Engineering Framework for Data Assimilation, Correlation, and Extrapolation. The architecture construct had already encapsulated the process distribution capability for data assimilation. The DACE model, due to the configuration flexibility, aligns to three levels with the LISI model and possibly provides value to a fourth level for non-compliant systems to operate in.

To be considered interoperable to any level within the LISI Reference Model, an application needs to be evaluated against the four attributes: Procedural (P), Applications (A), Infrastructure (I), and Data (D) at each level. The next sections will evaluate DACE concept against these four attributes at all levels. Within each level, one attribute is considered the key attribute for that level.

Level 0 consists of isolated systems that employ manual data transfer process. This includes end user copying data to CDs and other memory devices and physically carrying them to another computer for utilization. The primary enabler for the key interoperability category for this level is the "Procedures" attribute within the LISI Reference Model.

At this level, the "Procedures" attribute is described to focus on access control features. Systems access needs to be defined and documented to have procedure clarity.

This includes system login, security, data movement, and data disposal. The DACE model can be configured to operate on a local machine with no external connections. This assumes that all data to be utilized resides on the computer system in question, either in memory or in the unit's peripherals.

The "Applications" attribute, at this level, plays no role at this level. The transfer of data is controlled by manual operation and is independent of applications.

The "Infrastructure" attribute, at this level, is independent. Since there is no connection between systems, then there is no common infrastructure required. Again, the DACE model can be configured to run in a stand-alone configuration provided its data set is local. This means that there are simple data exchanges with independent databases. The DACE model allows for various types of information exchanges. General interactions with sensors and embedded systems comply with this level's data distribution attribute.

The last attribute is the "Data" attribute, at this level, the focus is on local data models. This means that data schemas are independent and give little consideration to interoperability. The DACE concept planned to provide generic data storage operations with a translation layer for interfacing within the DACE architecture. This construct aligns with the levels "Data" attribute. This does bring up a good point on future data interoperability. The Data Service Layer will need a customizable data extraction mechanism, which allows the end user to define what format/presentation data extraction will be extracted too.

Level 1 consists of peer-to-peer connectivity within the environment. This means that there is a communication link for device to device. This could be serial (RS-232, RS-422, RS-485), Ethernet (Telnet, FTP, etc.) or even buses (I2C, CANBUS, etc.). The data

exchange is typically small amounts of data like, small sensor messages up to text files. The connections are local such as on a Local Area Network (LAN) and the primary enabler for the key interoperability category is "Infrastructure" within LISI Reference Model table.

At this level, the "Procedures" attribute is described to focus on local and sight level procedures. The DACE model can be configured to connect locally. This means that the application can connect to a LAN, RS-232, or even a USB. There is a direct dependence on the computers I/O capabilities but the model provides no limitation to the LISI Level 1 specification in this configuration.

The "Applications" attribute, at this level, is characterized to focus on simple data exchanges electronically. The DACE model can be configured to exchange, or to gather, data from sensors, embedded devices, and other components for utilization within the DACE framework. DACE could then do higher level operations to correlate and extrapolate information to provide additional situational awareness or understanding.

The "Infrastructure" attribute, at this level, is focused on electronic connections among components. Specific focus is on peer-to-peer wired connections with common protocols. The DACE model allows for various types of communications which include Ethernet and serial. Within the Ethernet method are mechanisms such as Telnet, TCP/IP, UDP, and FTP, all of which align with this level's infrastructure focus.

The last attribute is the "Data" attribute, at this level, is focused on local data models. This means that there are simple data exchanges with independent databases. The DACE model allows for various types of information exchanges. General interactions with sensors and embedded systems comply with this level's data distribution attribute.

Level 2 consists of distributed interoperability within the environment. The ability to provide and access web-based data is a key component. The ability of independent applications to interact and process complex information in both a direct and a distributed fashion is the key feature for this level. Information can consist of simple data to audio, video, and picture images. The primary enabler for the key interoperability category is "Applications" within LISI Reference Model table.

At this level, the "Procedures" attribute is described to focus on program type procedures. This means that systems, at this level, should have similar procedures such as planning, training, and staffing, which align to a common operating environment. The DACE model general configuration capabilities can be configured to operate in this level. This is based on the concept of moving complex data in a distributed environment. The DACE model does have a limiting factor that has not been considered to function at this level. Applications that run at this level should comply with Department of Defense (DoD) 8320 data standards or have a migration plan to comply to this standard. Assuming this is still the primary standard for DoD, there are two possible solutions to meet this requirement. The first is to plan to comply and build to this standard. A review of this standard is needed to determine the impact to the development time frame. It may provide setbacks and expand the scope of this project effort. The second option is to create a plan to comply. This would be determined after a review of the standard and the impact to the project.

The "Applications" attribute, at this level, is characterized by the increased complexity of applications and their ability to have a common comprehension of the data set. The DACE model concept has not specifically called out any government

applications on which to interact. The concept is to provide a generic means to interact with applications, not bound the design to certain fixed applications. If the end user wishes to interact with another application and they understand the communication mechanisms for that application, then the end user has to define a component within the framework and its communication characteristics.

The "Infrastructure" attribute, at this level, is focused on electronic connections among many components on LANs. Specific focus is on the ability to establish communications with other systems without the need to change hardware. The DACE model allows for various types of communications which includes network communications. The Ethernet method contains mechanisms that align with this level such as Transmission Control Protocol (TCP)/Internet Protocol (IP), Hypertext Transfer Protocol (HTTP), and Real Time Streaming Protocol (RTSP); all align with this levels infrastructure.

The last attribute is the "Data" attribute, at this level, is focused program-wide, independent and sometimes duplicate databases. This means that there are complex data exchanges with independent databases and there are common tools like data dictionaries. The DACE model allows for various types of information exchanges. As long as the end user knows the information to interact with a system, it can be configured to interoperate.

Level 3 consists of interoperability within an integrated environment. Where there are domain level data models and procedures for interaction and sharing. In this level there is multiple application to application interactions but only have a local understanding of the domain. At this level, the primary enabler is the data attribute.

At this level, the "Procedures" attribute is described to focus on an application's ability to conform to the doctrine of the domain. This is difficult due to each service containing their own doctrine for development of systems for Joint operations. The DACE model could be operated within this level. The framework would not be the limiting factor. The limiting factor would again be the ability of the end user to acquire the doctrine and configure DACE to meet the domain's requirements. The DACE concept is not a limiter, it is an enabler. The model provides a generic capability that allows end users the ability to configure it to their situational needs.

The "Applications" attribute, at this level, is characterized to focus on the ability to cross discipline or organization boundaries. The utilization of higher level development languages that possess Object Oriented capabilities is strong. The DACE model is planned to be developed in C# for Business and Data Service Layers, and ASP for its Presentation Service Layer. Both languages are considered higher order languages with advanced capabilities. The DACE model has no designed restrictions to organizational boundaries. If there was to be a boundary issue when utilizing the DACE model, this would be in the organizational IT restrictions. The only way to resolve that issue is for the DACE model to get IA certified for Nonsecure Internet Protocol Router Network (NIPRNet) and Secret Internet Protocol Router Network (SIPRNet).

The "Infrastructure" attribute, at this level, is focused on a wide area network (WAN) capability. Specific focus is on the ability to interact over a broader domain, consisting of many LANs. The DACE model has no designed restrictions to organizational boundaries.

The last attribute is the "Data" attribute, at this level, the focus allows for direct database interactions. This includes having domain support artifacts such as data dictionaries and standard data elements. The DACE model's only restrictions under this attribute would again be the compliance to DoD standard 8320.

Level 4 consists of interoperability from an enterprise level in a universal environment. It is comprised of enterprise level data modules and procedures for interaction. At this level, applications share data and interact in a universal, integrated manner. This level is considered the ultimate goal of interoperability.

At this level, the "Procedures" attribute is described to focus on how well a system complies with the enterprise doctrine. Its enterprise system directly meets enterprise requirements and provided cross domain functions. The DACE model has no designed restrictions to enterprise operations. The biggest concern would be the end user's ability to obtain and define the enterprise doctrine within DACE.

The "Applications" attribute, at this level, is characterized to focus on multiple or redundant applications. In this context the DACE model would most likely fit as either a data assimilation and correlation engine, or a means to pull system data into this level that are not compliant at this level. This capability provides significant value to level capabilities. If the ultimate goal is to provide a level where all data interaction complies to an enterprise doctrine, then DACE could provide an interface at this level to allow sub compliant systems the ability to have their data sets present. It is import to specify data, because this would not make the application itself capable of running at this level.

The "Infrastructure" attribute, at this level, is focused on multi-dimensional networks. This can be in the form of location, security, or even virtual networks. A major

characteristic is its ability to replicate capabilities at lower levels. As in the previous statement, the ability to provide lower level functionalities to a higher level is a capability within DACE.

The last attribute is the "Data" attribute, at this level, the focus allows for universal data models and supporting artifacts. This level is indicative of a fully interoperable data environment with shared databases and servers. The DACE model complies with interoperability and the ability to correlate and extrapolate information for advanced analysis.

The DACE model, when evaluated against the LISI Reference Model held up fairly well. It has been shown that in general, the architecture could operate in theory within Levels 1 through 4. There are three deficiencies that have been identified, compliance to DoD 8320, IA compliance, and a possible issue of end users acquisition of the system doctrine and procedures to utilize DACE to interact with other applications within a DoD domain. Out of these deficiencies only the DoD 8320 can be addressed initially. The IA compliance item cannot be addressed until the full application is complete.

# VITA

Edward P Weaver
Department of Engineering Management and Systems Engineering
Old Dominion University
5155 Hampton Boulevard
Norfolk, VA 23529

Mr. Weaver has more than 20 years of engineering experience in design, development, and testing of complex systems. He has extensible experience in both software and electrical engineering disciplines and has provided engineering expertise for both the military and commercial applications in: unmanned surface vessels and intelligent systems; industrial sensors; modeling and simulation; embedded systems; audio and video, analog and digital high speed designs.

## EDUCATION

Master of Science Degree in Engineering Management
George Washington University, May 2007

Bachelor in Computer Engineering
Old Dominion University, May 2001

## PROFESSIONAL EXPERIENCE

October 2008 to Present
Project Manager/Principal Systems Engineer, W. R. Systems, Ltd.

May 2007 to October 2008
Project Manager/Senior Engineer, MTS Technologies, Inc.

December 2003 to May 2007
Design Engineer/Product Manager, Teledyne Hastings Instruments

September 2000 to December 2003
Design Engineer, Leitch, Inc.

## PATENTS

#: 8,184,296 B2, Emissions Monitoring Apparatus, System, and Method

## AWARDS & CERTIFICATIONS

Certified Project Management Professional (PMP)
Certified Software Development Processional (CSDP)
Reliability Centered Maintenance (RCM), Level 2 Certification