# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# DISSERTATION

**DEVELOPING DEPENDABLE SOFTWARE FOR A SYSTEM-OF-SYSTEMS**

by

Dale Scott Caffall

March 2005

Dissertation Supervisor: James Bret Michael

| | |
|---|---|
| **REPORT DOCUMENTATION PAGE** | *Form Approved OMB No. 0704-0188* |

| | | |
|---|---|---|
| **1. AGENCY USE ONLY** ***(Leave blank)*** | **2. REPORT DATE** March 2005 | **3. REPORT TYPE AND DATES COVERED** Dissertation |

| | |
|---|---|
| **4. TITLE AND SUBTITLE**: Developing Dependable Software for a System-of-Systems | **5. FUNDING NUMBERS** |
| **6. AUTHOR(S)** Dale Scott Caffall | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Naval Postgraduate School Monterey, CA 93943-5000 | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)** Missile Defense Agency 7100 Defense Pentagon, Washington, D.C. 20301-7100 | **10. SPONSORING / MONITORING AGENCY REPORT NUMBER** |

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

| | |
|---|---|
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT** Approved for public release; distribution is unlimited | **12b. DISTRIBUTION CODE** |

**13. ABSTRACT** ***(maximum 200 words)***

Capturing and realizing the desired system-of-systems behavior in the traditional natural language development documents is a complex issue given that the legacy systems in a system-of-systems exhibit independent behaviors. As a result of a development strategy of interconnecting systems, the emergent behavior of the system-of-systems cannot be predicted. In our consideration of dependable software for a system-of-systems, we used our case study of the Ballistic Missile Defense System to study the development of architectural views, distributed-system and real-time design considerations, components, contract interfaces, and the application of formal methods in system-of-systems specifications. We developed a prototype of a battle manager and demonstrated a slice of the formal model of the battle manager. Given the technical contributions of this research, we conclude that it is possible to develop an architecture from which we can reason about the controlling software for a system-of-systems. Furthermore, we can realize the controlling software for a system-of-systems through the concepts of component-based software engineering. Finally, we can apply formal methods in the design and development of the controlling software for a system-of-systems by specifying the requirements for the software components with assertions and employing a runtime-verification tool to verify the desired behavior as specified by the assertions.

| | |
|---|---|
| **14. SUBJECT TERMS** System-of-systems, dependable, trustworthy, architecture, distributed system, real-time system, component-based software engineering, kernel, formal methods, assertions, model checking | **15. NUMBER OF PAGES** 264 |
| | **16. PRICE CODE** |

| | | | |
|---|---|---|---|
| **17. SECURITY CLASSIFICATION OF REPORT** Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified | **20. LIMITATION OF ABSTRACT** UL |

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

# DEVELOPING DEPENDABLE SOFTWARE FOR A SYSTEM-OF-SYSTEMS

Dale Scott Caffall
Civilian, Missile Defense Agency, Washington, D.C.
B.S. in Electrical Engineering, University of Arizona, 1986
M.S. in Software Engineering, Naval Postgraduate School, 2003

Submitted in partial fulfillment of the
requirements for the degree of

**DOCTOR OF PHILOSOPHY IN SOFTWARE ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL**
**March 2005**

Author: ______________________________
Dale Scott Caffall

Approved by:

| | |
|---|---|
| James Bret Michael<br>Professor of Computer Science<br>Dissertation Supervisor<br>Committee Chairman | Man-tak Shing<br>Professor of Computer Science |
| Doron Drusinsky<br>Professor of Computer Science | Dan Boger<br>Chairman of Information Sciences |
| Kevin Greaney<br>Senior Software Engineer<br>DB Data Systems | |

Approved by: ______________________________
Peter J. Denning, Chairman, Department of Computer Science

Approved by: ______________________________
Julie Filizetti, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

Capturing and realizing the desired system-of-systems behavior in the traditional natural language development documents is a complex issue given that the legacy systems in a system-of-systems exhibit independent behaviors. As a result of a development strategy of interconnecting systems, the emergent behavior of the system-of-systems cannot be predicted. In our consideration of dependable software for a system-of-systems, we used our case study of the Ballistic Missile Defense System to study the development of architectural views, distributed-system and real-time design considerations, components, contract interfaces, and the application of formal methods in system-of-systems specifications. We developed a prototype of a battle manager and demonstrated a slice of the formal model of the battle manager.

Given the technical contributions of this research, we conclude that it is possible to develop an architecture from which we can reason about the controlling software for a system-of-systems. Furthermore, we can realize the controlling software for a system-of-systems through the concepts of component-based software engineering. Finally, we can apply formal methods in the design and development of the controlling software for a system-of-systems by specifying the requirements for the software components with assertions and employing a runtime-verification tool to verify the desired behavior as specified by the assertions.

THIS PAGE INTENTIONALLY LEFT BLANK

# TABLE OF CONTENTS

# LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

# ACKNOWLEDGMENTS

I would like to thank Professor Bret Michael, Professor Man-Tak Shing, Professor Doron Drusinsky, Professor Dan Boger, and Dr. Kevin Greaney for serving on my dissertation committee and providing me with wise counsel during the past four years. They worked tirelessly to help increase my understanding of software engineering. I would like to offer additional gratitude and admiration to Professor Michael who helped shape my understanding of software development, engaged me in valuable discussions, and provided sage guidance. He is a gifted educator and a talented researcher, and without peer in dedication and devotion to the responsibility of guiding students to greater academic achievements. I would like to thank Dr. Kevin Greaney for introducing me to the oustanding software engineering program at the Naval Postgraduate School and encouraging me throughout my journey.

I would like to thank Lieutenant Colonel Tom Cook who offered me numerous reviews of my dissertation material and keen insight into computer science. I would like to thank General Robert Dehnert and Mr. Richard Ritter for their encouragement and support during the past two years of my work and research. I would like to offer my thanks to Mr. Gerald Durbin who recruited me into Federal Service and mentored me throughout my career. He continues to be a great inspiration to me.

I would like to thank my mother and father who provided me with the inspiration to seek higher-level education through personal example and sacrifice. I offer thanks to my daughter Kim and our two granddaughters – Kailie and Alayna. They motivated me far beyond what they might imagine.

Most of all, I would like to thank the greatest wife in the world to whom I am very happily married. As is her giving and supporting nature, Trudy went “above and beyond” to ensure that the all was well on the home front, and made possible this doctoral journey. While this research and the challenges of my job motivate me to reach for higher goals, she inspires me to love life and to be a better man. I love her dearly.

THIS PAGE INTENTIONALLY LEFT BLANK

# EXECUTIVE SUMMARY

Many of the systems that comprise a system-of-systems most likely existed first as legacy systems that operated as stand-alone capabilities in the operational world. These legacy systems were developed with specific sets of requirements and with specific system functionality in mind. Additionally, just as we developed the legacy systems as independent developments, we are typically developing new systems that will become members of a system-of-systems under similar conditions. That is, we are developing these systems as stand-alone capabilities with specific sets of requirements and with specific system functionality in mind.

Typically, developers will connect these systems through some communication medium in the hope of achieving greater functionality, although this long-standing acquisition strategy does not necessarily result in the intended synergistic effect. One can identify the systems that will form the system-of-systems, and then set out to bend, fold, spindle, and mutilate these systems in the fevered hope of producing a functional composition: it is difficult to think about the system-of-systems as a single entity, which may explain why system developers sometimes mistakenly focus on modifying individual systems with little deliberation and consideration for the system as a whole.

The control of a system-of-systems presents a tremendous challenge to software developers. As developers interconnect a number of independent systems to form a system-of-systems, they should address the emergent properties for the control of a system-of-systems that cannot be predicted by analyzing each independent system.

The following is a summary of the technical contributions of this research:

1. Identification of distributed-system attributes for controlling software in a system-of-systems

2. Identification of real-time attributes for real-time controlling software in a reactive system-of-systems

3. Development of system-of-systems architecture views from system-of-systems view to component view in controlling software

4. Use of kernel in controlling software for system-of-systems to shape dependable behavior of system-of-systems

5. Reduction of software complexity from an exponential factor for a monolithic software program to a component-based construct in which the active components are decoupled by data stores

6. Development of assertions from collaboration diagrams

7. Adaptation of CBSE by advanced use of assertions in interface contracts between components to assert protocols surrounding the components in reactive systems

8. Demonstration that formal methods can be applied to large, complex system-of-systems developments

The technical contributions of this research offer evidence that lead us to conclude the following about the questions posed for this research:

1. It is possible to develop a system-of-systems architecture from which we can reason about the controlling software for a system-of-systems.

2. We can realize the controlling software from a system-of-systems architecture through the concepts of component-based software engineering.

3. We can apply formal methods in the design and development of the controlling software for a system-of-systems by specifying the requirements for the software components with assertions and employing a runtime verification tool to verify the desired behavior specified in the assertions.

Additionally, this research addresses David Parnas' challenges back in 1985 to the Department of Defense on the Strategic Defense Initiative (SDI). Parnas' six issues are summarized as follows:

1. Discrimination of the threat objects from decoys and debris is a significant challenge.

2. Software developers cannot predict the behavior of the battle-management software with confidence given the actual configuration of weapons, sensors, and battle managers are not known until the moment of battle.

3. Software developers cannot test the battle-management software under realistic conditions.

4. The duration of the defense engagement will be short. It will not allow for either human intervention or debugging the software to overcome software faults at runtime.

5. Battle-management software will have absolute real-time deadlines.

6. Battle-management software must integrate numerous dynamic software systems to the extent that has never before been achieved.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

## A. BACKGROUND

The annals of human conflict are replete with the terrible results of the traditional war strategy of attrition in which opposing forces attempt to inflict more casualties on the enemy than the enemy can sustain and maintain a viable military force. This "mass-on-mass" strategy resulted in staggering losses of life in countless wars. For example, 623,026 soldiers lost their lives in the four years of the U.S. Civil War. At Antietam, the combined casualties of Union and Confederate forces totaled 26,134 soldiers on a single day of battle. [48] The war of attrition concept was a costly strategy in terms of human life.

During the past decade, the Department of Defense (DoD) shifted military tactics from the traditional war of attrition to a transformational concept of full-spectrum dominance: the ability of US forces, operating unilaterally or in combination with multinational and interagency partners, to defeat any adversary and control any situation across the full range of military operations. In Joint Vision 2020 (JV 2020), the Chairman, Joint Chiefs of Staff included the following operational concepts that will support the achievement of full-spectrum dominance [23]:

1. Dominant maneuver is the ability of joint forces to gain positional advantage with decisive speed and overwhelming operational tempo in the achievement of assigned military tasks.

2. Focused logistics is the ability to provide the joint force with the right personnel, equipment, and supplies in the right place, at the right time, and in the right quantity, across the full range of military operations.

3. Full dimensional protection is the ability of the joint force to protect its personnel and other assets required to decisively execute assigned tasks.

4. Precision engagement is the ability of joint forces to locate, surveil, discern, and track objectives or targets; select, organize, and use the correct systems;

generate desired effects, assess results; and reengage with decisive speed and overwhelming operational tempo as required, throughout the full range of military operations.

## B. SYSTEM-OF-SYSTEMS ENVIRONMENT

In recent times, systems-of-systems have exploded into the battlespace of the joint and coalition warfighters to meet the challenges that the Chairman identified in his vision of future warfare. (N.B.: For this research, we define a system-of-systems as an amalgamation of legacy systems and developing systems that provide an enhanced military capability greater than that of any of the individual systems within the system-of-systems.) The development community's response in the U.S. Department of Defense to the rabid craving for more accurate information and more lethal functionality has been a less than sterling hobbling of various legacy systems and ongoing system developments through tightly coupled and lowly cohesive communication shackles.

Many of the systems that comprise a system-of-systems most likely existed first as legacy systems that operated as stand-alone capabilities in the operational world. These legacy systems were developed with specific sets of requirements and with specific system functionality in mind. Just as we developed the legacy systems as independent developments, we are typically developing new systems that will become members of a system-of-systems under similar conditions. That is, we are developing these systems as stand-alone capabilities with specific sets of requirements and with specific system functionality in mind.

Typically, developers will connect these systems through some communication medium in the hope of achieving greater functionality, although this long-standing development strategy does not necessarily result in the intended synergistic effect. One can identify the systems that will form the system-of-systems, and then set out to bend, fold, spindle, and mutilate these systems in the fevered hope of producing a functional composition: it is difficult to think about the system-of-systems as a single entity, which may explain why system developers sometimes mistakenly focus on modifying individual systems with limited deliberation and consideration for the system as a whole.

Our tools for reasoning about a system-of-systems typically consist of little more than a “sticks-and-circles” diagram. The “circles” represent the various systems that comprise the system-of-systems while the “sticks” are means of information transfer, a messaging protocol, and, perhaps, a translator box to translate the messaging format from one system to another. Armed with this sophomoric view of the system-of-systems, we attempt to analyze and design the system-of-systems through a trivial picture of the various systems as connected by a convoluted labyrinth of lines. Unfortunately, sticks-and-circles diagrams lack both a formal semantics and the richness needed to express the many dimensions of system behavior.

Are the circles meant to represent systems, subsystems, modules, classes, objects, functions, hardware, or some other entity? Are the sticks meant to represent data flow, triggers, synchronization, calls, inheritance, or something else?

Far too frequently, we initiate detailed design and coding from reasoning about the sticks-and-circles diagrams. During the development, we add new layers of features and functional enhancements to the system software without clear insight into the organization of the system software. Inevitably, the basic organization of the software that seemed so reasonable at the beginning of the development process begins to break apart under the weight of the revisions made to the system software. Sadly, the software development becomes another casualty to report in future studies as to why software developments are not successful.

Traditionally, this methodology failed to achieve an interoperable and integrated system-of-systems with predictable, dependable behavior. With each new failure, the system engineers attempted to “tighten up” the protocol standard; however, the system-of-systems cannot be made to exhibit predictable, dependable behavior by increasing the level of detail in the interconnectivity standards. As a result of the traditional system-of-systems development effort, the end-state is a collection of systems that have a high degree of coupling with a realized protocol standard that only serves to significantly increase the system-of-systems software complexity.

As we have witnessed time and again, system-software critical interactions increase as the complexity of highly interconnected systems increases. In the complex system-of-systems, these possible combinations are practically limitless. System "unravelings" seem to have an intelligence of their own as they expose hidden connections, neutralize redundancies, and exploit chance circumstances for which no system engineer might plan. A software fault at runtime in one module of the system-software may coincide with the software fault of an entirely different module of the system-software. This unforeseeable combination can cause cascading failures within the system-of-systems.

## C. CONTROLLING SOFTWARE IN A SYSTEM-OF-SYSTEMS[1]

The control of a system-of-systems presents a tremendous challenge to software developers. As developers interconnect a number of independent systems to form a system-of-systems, they should address the emergent properties for the control of a system-of-systems that cannot be predicted by analyzing each independent system. [100]

In [101] and [102], the authors maintain that the "…basic paradigm of control has not found its place as a first-class concept in software engineering." They offer a new paradigm that considers the software system as a plant and includes a "controller subsystem" for controlling the plant. As suggested in [15], we offer the concept of developing the controlling software as a distinct control application for a system-of-systems.

We suggest that the battle-management software in the Ballistic Missile Defense System (BMDS) exemplifies the aforementioned difficulties and challenges of controlling software in a system-of-systems. (N.B.: The BMDS is a collection of independent missile defense systems that will be integrated into a system-of-systems. We describe the BMDS in detail in Chapter II as we use the BMDS as our case study in this research.)

In his book "Software Fundamentals: Collected Papers by David L. Parnas," Parnas outlines six major characteristics of the battle-management software in the

[1] The text of the statement of the problem is largely an extract from [10].

Strategic Defense Initiative (SDI) program (known today as the BMDS). [61] The following issues are as relevant today as during the time when Parnas published his observations:

1. The battle-management software must identify, track, and direct weapons towards targets whose characteristics may not be known with certainty until the moment of battle. The battle-management software must discriminate the threat objects from decoys and debris.

2. The battle-management computing will be accomplished through a network of computers that are connected to sensors and weapons as well as other battle-management computers. The behavior of the battle-management software cannot be predicted with confidence given the actual configuration of weapons, sensors, and battle managers at the moment of battle.

3. Developers cannot test the battle-management software under realistic conditions prior to actual use of the software.

4. The duration of the defense engagement will be short: it will not allow for either human intervention or debugging the software to overcome software faults at runtime.

5. The battle-management software will have absolute real-time deadlines for the computation that will consist of periodic processes to include detecting and identifying potential threat missiles, assigning a weapon to engage the threat missile, and providing an assessment of the interceptor-threat missile engagement. Because of the unpredictability of the computational requirements of each process, developers cannot predict the required resources for computation.

6. The missile defense system will include a large variety of sensors, weapons, and battle-management components for which all will be large, complex software systems. The suite of weapons and sensors will increase in number as the development progresses. The characteristics of these future weapons and sensors are not well defined and will likely remain fluid for many years. Additionally, all weapons and sensors will be subject to change independently of each other. As such, the battle-

management software must integrate numerous dynamic software systems to the extent that has never before been achieved.

## D. CONTRIBUTIONS OF THIS RESEARCH

The following is a summary of the technical contributions of this research:

1. Identification of distributed-system attributes for controlling software in a system-of-systems

2. Identification of real-time attributes for real-time controlling software in a reactive system-of-systems

3. Development of system-of-systems architecture views from system-of-systems view to component view in controlling software

4. Use of kernel in controlling software for system-of-systems to shape dependable behavior of system-of-systems

5. Reduction of software complexity from an exponential factor for a monolithic software program to a component-based construct in which the active components are decoupled by data stores

6. Development of assertions from collaboration diagrams

7. Adaptation of CBSE by advanced use of assertions in interface contracts between components to assert protocols surrounding the components in reactive systems

8. Demonstration that formal methods can be applied to large, complex system-of-systems developments

The technical contributions of this research offer evidence that lead us to conclude the following about the questions posed for this research:

1. It is possible to develop a system-of-systems architecture from which we can reason about the controlling software for a system-of-systems.

2. We can realize the controlling software from a system-of-systems architecture through the concepts of component-based software engineering.

3. We can apply formal methods in the design and development of the controlling software for a system-of-systems by specifying the requirements for the software components with assertions and employing a runtime verification tool to verify the desired behavior specified in the assertions.

Additionally, this research addresses each of the aforementioned challenges highlighted by Parnas regarding the Strategic Defense Initiative (SDI). (N.B.: President Ronald Reagan tasked the SDI Organization (SDIO) to develop the SDI which was colloquially known as Star Wars. SDIO later become the Ballistic Missile Defense Organization (BMDO). In January 2002, Secretary of Defense Donald Rumsfeld elevated BMDO to agency status and renamed BMDO as the Missile Defense Agency (MDA). SDI is currently known as the BMDS which contains both the former National Missile Defense (NMD) system and the Theater Ballistic Missile Defense (TBMD) systems.)

## E. OVERVIEW OF THE DISSERTATION

In Chapter II, we reference Parnas' six issues and discuss the difficulty involved in the design and development of a system-of-systems. Parnas' six issues are summarized as follows:

1. Discrimination of the threat objects from decoys and debris is a significant challenge.
2. Software developers cannot predict the behavior of the battle-management software with confidence given the actual configuration of weapons, sensors, and battle managers are not known until the moment of battle.
3. Software developers cannot test the battle-management software under realistic conditions.
4. The duration of the defense engagement will be short. It will not allow for either human intervention or debugging the software to overcome software faults at runtime.
5. Battle-management software will have absolute real-time deadlines.
6. Battle-management software must integrate numerous dynamic software systems to the extent that has never before been achieved.

In Chapter III, we provide examples of system-of-systems that failed in operations – two of the examples reflect the loss of life as a result of unknown emergent behavior in the system-of-systems.

In Chapter IV, we assess the previous research and developments of systems-of-systems. Based upon our assessment of previous research and developments of system-of-systems, the key topics to be addressed are as follows:

1. Although there would seem to be a global trend towards composing systems-of-systems, there appears to be a significant void in both research activities and the engineering practices for the development of a system-of-systems.

2. The architecture of the system-of-systems seems to be a critical issue for the developer of a system-of-systems. A suitable architecture may be a valuable first step in the successful design and development of a system-of-systems.

3. A component-based structure could be one characteristic of successful system-of-system developments. With components, we might consider techniques in which we can isolate the behavior of a given component from all others with the objective of isolating faults within the component.

4. Formal methods and model checking could be useful in the design and development of a system-of-systems. It seems as if previous research considered the entire system-of-systems rather than decomposing the system-of-systems into components for which formal methods and model checking could provide value.

In Chapter V, we pose the following research questions:

1. Is it possible to develop a system-of-systems architecture from which we can reason about the controlling software for a system-of-systems?
2. Can we realize the controlling software from a system-of-systems architecture through the concepts of component-based software engineering?
3. Can we apply formal methods in the design and development of the controlling software for a system-of-systems by specifying the requirements for the software components with assertions and employing a runtime verification tool to verify the desired behavior specified in the assertions?

In Chapter VI, we define a dependable system and a trustworthy system as follows:

A **dependable system** is one that provides the appropriate levels of correctness and robustness in accomplishing its mission while demonstrating the appropriate levels of availability, consistency, reliability, safety, and recoverability.

A **trustworthy system** is one that provides the appropriate levels of correctness and robustness in accomplishing its mission while demonstrating the appropriate levels of availability, consistency, reliability, safety, and recoverability *to the degree that justifies a user's confidence that the system will behave as expected.*

We discuss specific dependability issues in the development of specifications, interfaces, battle-management kernel (BMK), distributed systems, and real-time systems.

In Chapter VII, we briefly discuss the distributed-system issues of the battle manager; however, we propose that this be a future research topic.

In Chapter VIII, we briefly discuss the real-time issues of the battle manager; however, we propose that this be a future research topic.

In Chapter IX, we offer several architectural views of the battle manager and propose the following architectural principles for the battle manager:

1. With the knowledge of the short timelines to conduct battle management for missile defense, it is not prudent to realize the battle-manager capability in a centralized fashion; that is, it is not reasonable to expect such a system to be positioned within the United States and require the system to direct the engagements of all possible ballistic missiles from all parts of the globe. As such, we will consider a distributed-system construct for the BMDS Battle Manager. The distributed battle manager must be able to communicate with all the sensors and all the weapons systems in the BMDS; however, the distributed battle manager should be transparent from the perspective of the sensors and weapons connected to it.

2. We anticipate that the BMDS Battle Manager will continually experience modifications and upgrades to its applications. As such, it would be useful to isolate the

software that will change little over time from the software that will change more frequently. We will employ the concept of component-based engineering to design and develop the BMDS Battle Manager.

3. We will develop the software that realizes the basic functions of the battle manager as a kernel given that this software should experience limited modifications over time. We will add a software application to the kernel that controls the distributed processing in the battle manager.

4. We have defined an architecture for the battle manager that could provide flexibility for decisions in the detailed design. The architecture that we have defined for the battle manager could allow the incorporation of design details for dependable software, distributed systems, real-time software, software kernels, and software components.

In Chapter X, we propose the use of a BMK that consists of the set of software components that are necessary to provide correct real-time execution of battle-management tasks in a system-of-systems context, both in nominal and degraded modes of system operation. We propose that the BMK should exhibit the following characteristics:

1. The BMK has absolute priority. That is, no other component can interrupt the kernel from accomplishing its work.

2. System parameters and external events are measurable and observable by the BMK. When presented with measurements for a given set of parameters and external events, the BMK will exhibit correct system behavior. (N.B.: We define correct as the reaching of the desired state given the previous state is presented with a given set of inputs.)

3. Detection of errors will be through the use of assertions. The BMK will direct non-kernel software components for the recovery of observed errors such as violation of pre-conditions, post-conditions, and invariants.

In Chapter XI, we propose that we develop the software that contains the basic functions of battle management as a set of components in the BMK given that this software should experience limited modifications over time. We propose to develop other component software that contains the algorithms required to perform the computations of the BMDS Battle Manager.

In Chapter XII, we assert that our current development techniques are failing to support system developers in producing systems with predictable behavior. Almost exclusively, acquisition organizations rely on exhaustive testing prior to fielding the completed product to assess system behavior. Rather than discovering system behavior at the end of the development phase, developers might apply techniques that support the design and realization of desired system behavior from the earliest phases of concept development and requirements development.

The application of formal methods for specification and verification is a technique for consideration by developers of system-of-systems. Formal methods can complement traditional techniques such as testing and can help developers improve the degree of trustworthiness in defense acquisitions.

For the battle manager, we will use assertions to specify the desired behavior. While assertions alone will not ensure dependable software, the use of assertions can increase the level of dependability of a system.

In Chapter XIII, we develop a prototype that contains natural language assertions to specify specific behavior for the track processing and weapon-system assignment in the BMK. Additionally, we develop safety policies for the battle manager and proposed natural language assertions to monitor the safety policies in the BMK. Finally, we identify specific distributed behavior and offer natural language assertions to specify the desired distributed behavior among BMKs in the system-of-systems.

In Chapter XIV, we demonstrate a slice of the BMK as we transform several natural language assertions to temporal logic assertions and run those assertions through a model checker to determine whether we achieved the desired behavior.

In Chapter XV, we outline the technical contributions of this research and present a matrix of the contributions against Parnas' six issues for SDI.

In Chapter XVI, we offer future research topics for the subject of systems-of-systems.

# II. STATEMENT OF THE PROBLEM

## A. INTRODUCTION

In the crusade for acquisition reform, the tendency in specifications is to document the “thou shalts” of specific system functions, design to the “thou shalts” with modifications to accommodate the development, and field a system that little resembles the collective “thou shalts” and, more importantly contains limited user utility. The Department of Defense has realized limited success in attempting to capture the desired system behavior in natural language requirements documentation (e.g., Operational Requirements Documents (ORDs), System Requirements Specifications (SRSs), and Interface Requirements Specifications (IRSs)) and achieve the documented behavior in the fielded systems. [49] In [50], Leffingwell and Widrig discuss the impacts of insufficient and incomplete system specifications.

The system-of-systems problem is one of designing and implementing the degree of desired system behavior by somehow connecting legacy systems. (N.B.: For this research, we define system behavior as the collective responses of a system as it reacts to stimuli such as sensory information, a clock, or a received transaction.) Capturing the desired system-of-systems behavior in the traditional natural language documents is a complex issue given that the legacy systems in the system-of-systems have a combination of existing known and unknown system behaviors. Typically, the system-of-systems specification is reduced to a table of information exchange requirements (IERs) that define the messaging that passes from one system to another.

As a result of a system-of-systems development strategy of interconnecting systems while concentrating the development efforts on messaging and protocols, the operational system-of-systems frequently demonstrate undesired system behaviors. Although the system-of-systems may require safety considerations, system architects and system designers can experience a significant degree of difficulty in testing for the safety-critical features and certifying a system-of-systems as being safe. (N.B.: For this research, we define safety as the property of avoiding a catastrophic outcome given a system fails to operate correctly.) Additionally, the user is frequently irritated at the undesirable distributed-system behaviors that a system-of-systems may exhibit such as halted processes without recovery and disparate versions of same-source data.

While a primary concern of a system-of-systems is the exhibited system behavior, the financial costs for incorrectly and insufficiently specified systems is staggering. System-specification errors will be the source of seventy percent of the system rework costs. Given that rework costs are typically 30% to 50% of a program budget, the correction of system-specification errors can cost 25% to 40% of an entire program budget. [50]

Increasingly, software concerns are overtaking hardware concerns in systems engineering. From numerous lessons-learned from failed developments, it would appear that the following statements are becoming development truths [50]:

*Software – not hardware – determines the degree of achieved success in the fielded capabilities of our systems.*

*Software – not hardware – consumes the majority of costs for system development.*

*Software – not hardware – is on the critical path of every development and ultimately determines when and if a system is fielded.*

*Software – not hardware – is the entity changed most often in the acquisition lifecycle of a system to meet the changing needs of the warfighters.*

Our journey to achieve the desired system behavior in a system-of-systems has no end in sight. The development community continues to fail in delivering the critical functionality required by the warfighters in our system-of-systems. The development community continues to fail in capturing and achieving the desired system behavior, and the development community continues to invest a significant amount of the program budget in system rework to correct specification deficiencies.

Unfortunately, we seem to accept these facts and are resigned to repeat the errors of our predecessors. We continue on with the same bad practices of the past. As the old adage goes: we do not plan to fail - we just fail to plan.

## B. CASE STUDY: BALLISTIC MISSILE DEFENSE

To help us understand the proposed concepts for developing dependable software for a system-of-systems environment, we will use the BMDS as a case study in this research. We offer that the BMDS exemplifies other systems-of-systems as the BMDS is an amalgamation of numerous independent systems. Although we will describe the research with respect to the BMDS, the statement of the problem, proposed concepts, and conclusions of this research is representative of other system-of-systems environments for which a dependable system-of-systems-level software is required.

Within a sensor-to-shooter system-of-systems, battle managers hold the controlling software. The legacy systems in the system-of-systems include sensors that detect and track a threat object, and weapon systems that compute firing solutions and engage the threat object.

We find the importance of battle management within the concept of precision engagement. (N.B.: For this research, we define battle management as the decisions and actions executed in direct response to the activities of enemy forces in support of the Joint Chiefs of Staff's concept of precision engagement. [18]) Battle managers must rapidly make decisions to counter both enemy actions and force movements. Battle managers must correctly cope with the fog-of-war conditions that are ever-present during the prosecution of the war. The success or failure of the battle-management functions will determine the success or failure of joint forces with respect to the achievement of their assigned objectives. [30]

### 1. Ballistic Missile Threats

With respect to ballistic missile threats, the current trend is increased distance for ballistic missile flight which requires greater velocities of future threats. Additionally, the proliferation of ballistic missiles is worldwide as noted by the Director of Central Intelligence in his testimony before the Senate Select Committee on Intelligence. [76] Excerpts from his testimony are as presented below:

"North Korea also continues to advance its missile programs. North Korea is nearly self-sufficient in ballistic missiles, and has continued procurement of raw materials and components for its extensive ballistic missile programs from various foreign sources.

The North also has demonstrated a willingness to sell complete systems and components that have enabled other states to acquire longer-range capabilities and a basis for domestic development efforts earlier than would otherwise have been possible."

"North Korea has maintained a unilateral long-range missile launch moratorium since 1999, but could end that with little or no warning. The multiple-stage Taepo Dong-2—capable of reaching the United States with a nuclear weapon-sized payload—may be ready for flight-testing."

"Finally, Iran's missile program is both a regional threat and a proliferation concern. Iran's ballistic missile inventory is among the largest in the Middle East and includes the 1300-km range Shahab-3 medium-range ballistic missile (MRBM) as well as a few hundred short-range ballistic missiles (SRBMs). Iran has announced production of the Shahab-3 and publicly acknowledged development of follow-on versions. During 2003, Iran continued R&D on its longer-range ballistic missile programs, and publicly reiterated its intention to develop space launch vehicles (SLVs)—and SLVs contain most of the key building blocks for an intercontinental ballistic missile (ICBM). Iran could begin flight-testing these systems in the mid- to latter-part of the decade."

"Iran also appears willing to supply missile-related technology to countries of concern and publicly advertises its artillery rockets and related technologies, including guidance instruments and missile propellants."

"China continues an aggressive missile modernization program that will improve its ability to conduct a wide range of military options against Taiwan supported by both cruise and ballistic missiles. Expected technical improvements will give Beijing a more accurate and lethal missile force. China is also moving on with its first generation of mobile strategic missiles."

"Although Beijing has taken steps to improve ballistic missile related export controls, Chinese firms continue to be a leading source of relevant technology and continue to work with other countries on ballistic missile-related projects."

"South Asian ballistic missile development continues apace. Both India and Pakistan are pressing ahead with development and testing of longer-range ballistic missiles and are inducting additional SRBMs into missile units. Both countries are testing missiles that will enable them to deliver nuclear warheads to greater distances."

"Last year Syria continued to seek help from abroad to establish a solid-propellant rocket motor development and production capability. Syria's liquid-propellant ballistic missile program continued to depend on essential foreign equipment and assistance, primarily from North Korean entities. Syria is developing longer-range missile programs, such as a Scud D and possibly other variants, with assistance from North Korea and Iran."

"Many countries remain interested in developing or acquiring land-attack cruise missiles, which are almost always significantly more accurate than ballistic missiles and complicate missile defense systems. Unmanned aerial vehicles are also of growing concern."

"To conclude my comments on proliferation, I'll briefly run through some WMD programs I have not yet discussed, beginning with Syria."

"Syria is an NPT signatory with full-scope IAEA safeguards and has a nuclear research center at Dayr Al Hajar. Russia and Syria have continued their long-standing agreements on cooperation regarding nuclear energy, although specific assistance has not yet materialized. Broader access to foreign expertise provides opportunities to expand its indigenous capabilities and we are closely monitoring Syrian nuclear intentions. Meanwhile, Damascus has an active CW development and testing program that relies on foreign suppliers for key controlled chemicals suitable for producing CW."

"Finally, we remain alert to the vulnerability of Russian WMD materials and technology to theft or diversion. We are also concerned by the continued eagerness of Russia's cash-strapped defense, biotechnology, chemical, aerospace, and nuclear industries to raise funds via exports and transfers—which makes Russian expertise an attractive target for countries and groups seeking WMD and missile-related assistance."

**2. Description of the Ballistic Missile Trajectory**

All ballistic missiles share a common, fundamental element - they follow a ballistic trajectory that includes three phases. These phases are the boost phase, the midcourse phase, and the terminal phase.

The boost phase is the portion of a missile's flight in which it is thrusting to gain the acceleration needed to reach its target. This phase usually lasts 50 seconds for a 1000 kilometer ballistic missile threat and 75 seconds for a 3000 kilometer ballistic missile threat. During the boost phase the rocket is climbing against the earth's gravity and either exiting the earth's atmosphere, or in the case of shorter-range missiles, only reaching the fringes of outer space.

Once the missile has completed firing its propulsion system, it begins the longest part of its flight, which is known as the mid-course phase. During this phase the missile is coasting, or freefalling towards it target. This phase can be as short as 6 minutes for a 1000 kilometer ballistic missile threat and as long as 11 minutes for a 3000 kilometer ballistic missile threat. Most missiles that leave the atmosphere shed their rocket motors by this time in order to increase the range that the missile's weapon (i.e., warhead) can travel. For medium and long-range missiles this phase occurs outside the earth's atmosphere.

The final phase of a missile's flight is the terminal phase which is flight space in which the ballistic missile threat reenters the Earth's atmosphere during which the atmosphere begins to measurably impact the velocity and flight characteristics of the ballistic missile threat. For this research, we will consider 32 kilometers above sea level as the point of reentry into Earth's atmosphere. During this phase the ballistic missile's warhead reenters the Earth's atmosphere at incredible speeds. For example, a 3000 kilometer ballistic missile threat has a velocity of 3.85 kilometers per second at an altitude of 10 kilometers. This phase last approximately 16 seconds for a 1000 kilometer ballistic missile threat and approximately 10 seconds for a 3000 kilometer ballistic missile threat. [55] (N.B.: We offer the flight times for missiles in the three phases as a point of reference rather than absolute values. [55] offers a detailed analysis of flight times for various missile types and atmospheric conditions.)

### 3. Ballistic Missile Defense System

DoD plans to acquire a layered ballistic missile defense to defend the forces and territories of the United States, its Allies, and friends against all classes of ballistic missile threats. The Ballistic Missile Defense (BMD) program will pursue a broad range of activities in order to develop and evaluate technologies for the integration of land, sea, air, and space-based platforms to counter ballistic missiles in all phases of their flight. In parallel, sensor suites and battle management and command and control will be developed to form the backbone of the BMDS. The Missile Defense Agency[2] (MDA) will accomplish this mission by developing a layered defense that employs complementary sensors and weapons to engage threat targets in the boost, midcourse, and terminal phases of flight, and incrementally deploying that capability.

There are advantages and challenges to set up engagement opportunities against a threat missile in each of the three phases of flight. The capability to defend against an attacking missile in each of these phases is called a layered defense, and it may be expected to increase the chances that the missile and its payload will be destroyed. By attacking the missile in all phases of flight, we exploit opportunities that could increase the advantage of the defense. A capability to intercept a missile in the boost phase, for example, can destroy a missile regardless of its range or intended aim-point and provide a global coverage capability. A midcourse intercept capability can provide wide coverage of a region or regions, while a terminal defense reduces the protection coverage considerably to a localized area. As we add shot opportunities in the midcourse and terminal phases of flight to boost phase opportunities, we increase the probability for a successful intercept of the ballistic missile threat.

Improving the odds of interception becomes critical when ballistic missiles carry weapons of mass destruction. When possible, for the global coverage and protection against lethal payloads, a capability to intercept a missile near its launch point is always

[2] MDA is the US agency within the Department of Defene that is responsible for leading the design and development of the BMDS. MDA evolved from the former Ballistic Missile Defense Organization which evolved from the former Strategic Defense Initiative Organization.

preferable to attempting to intercept that same missile closer to its target. To minimize the negative fallout effects from weapons of mass destruction, we expect terminal defense systems to engage a ballistic missile threat at a minimum altitude of 15 kilometers above a defended asset. For those mid-course systems that engage ballistic missile threats beyond the discernable impacts of Earth's atmosphere, we expect mid-course systems to engage a ballistic missile threat at a minimum altitude of 83 kilometers above a defended asset. [55]

#### 4. Battle Manager

The battle managers must direct the activities in the battlespace. Typically, multiple engagements occur concurrently in the battlespace. Oftentimes, the activities for killing a threat object at such a high operations tempo (OPTEMPO) that humans can experience great difficulty in maintaining situational awareness of the entire battlespace. (N.B.: For this research, we define operations tempo as **t**he rate of military actions or missions. Additionally, we define situational awareness as the perception of available facts, comprehension of the facts in relation to the individual's expert knowledge, and projecting how the situation is likely to develop in the future.)

The challenge will be to develop the battle manager as a dependable system within the capabilities and constraints of the system-of-systems. (N.B.: For this research, we define a dependable system as one that demonstrates the appropriate levels of availability, reliability, safety, and recoverability to the degree that justifies a user's confidence that the system will behave as expected.)

## C. PROBLEM STATEMENT

The six issues identified by Parnas are not unique to the BMDS Battle Manager. With limited tailoring, these six issues could extend to controlling software in any system-of-systems.

Referencing Parnas' six issues for the battle manager, we propose that a problem in the acquisition community is defining, developing, and building a controller in a system-of-systems environment that is available for operations at any time, operates correctly at all times, traps system faults and returns to operations without impacting the mission of the BMDS, and performs its missions in such a way that no unintended harm to people and protected assets will come from its operations.

This is the problem that we will address in this research.

THIS PAGE INTENTIONALLY LEFT BLANK

# III. SIGNIFICANCE OF THE PROBLEM

## A. BACKGROUND

A system-of-systems development raises a multitude of issues that are beyond the development of a single system such as the system behavior of the system-of-systems as well as each system within the system-of-systems. In defense acquisitions, the control of a system-of-systems is typically left to humans via voice or assigned sector responsibilities in the battlespace (i.e., a predetermined, physical division of the battlespace in the military operations for each system in the system-of-systems). As examples to the potential impact of our current inability to confidently predict system behavior and our continued failings in system-of-systems developments, the following anecdotes are offered:

### 1. Insufficient Requirements Specification and Verification

US Central Command (CENTCOM) forces deployed six PATRIOT batteries in the Dhahran area of operations during the Persian Gulf War of 1991. Of those six PATRIOT batteries, CENTCOM forces assigned Alpha Battery the mission of protecting the Dhahran air base. Alpha Battery had been in continuous operations for over one hundred hours on February 25, 1991. Iraqi forces launched a Scud missile at the Dhahran air base that Alpha Battery failed to track and intercept. The Scud missile impacted at an US Army barracks and killed twenty-eight US soldiers. Subsequent investigations into this catastrophe revealed that PATRIOT could not perform sustained operations beyond twenty continuous hours as potential targets would fall outside the range gate – an electronic detection system within the PATRIOT radar that calculates the area in the field of regard where PATRIOT should next look for the threat missile. At one hundred hours of continuous operation, the shift in the range gate would be 687 meters so the PATRIOT could not detect, track, and destroy incoming ballistic missiles. [35][62]

### 2. System-of-Systems Integration

In December of 2001, a 2000-pound, Joint Direct Attach Munitions (JDAM) bomb killed three U.S. Special Forces airmen and five Afghan soldiers, and wounded nineteen other military personnel. The root cause of this friendly-fire incident was the inadvertent passing of the coordinates of the US air controller's own position to the bomber. While the air controller had correctly passed the latitude and longitude target coordinates to a US Navy F/A-18 minutes prior to the friendly-fire incident, the US Air Force B-52 bomber crew required a second calculation in "degrees decimal." The air controller complied with the request and stored the coordinates in the GPS receiver. At that moment, the GPS' battery expired and the air controller replaced it with a fresh battery. Apparently unbeknownst to the air controller, the GPS receiver re-initialized and displayed its coordinates (as programmed into the GPS software) which the air controller passed to the bomber crew. The JDAM struck at those coordinates with deadly precision. [53] [77] Included with the ground-aided precision strike conclusions and recommendations in [77], the author recommended an interface between relay-coordinate systems and weapons vice the human handling of target coordinates. What was not mentioned was the insufficient integration of independently-acquired systems into a larger system-of-systems.

### 3. Specifications and Error-Handling Verification

From a study of 387 software errors discovered during the integration and testing phase of the Voyager and Galileo spacecraft, Robyn Lutz observed that the safety-related, functional faults exhibited by Voyager could be categorized as follows: 50% as behavioral faults, 31% as conditional faults, and 19% as operating faults. For Galileo, the safety-related, functional faults could be categorized as follows: 38% as behavioral faults, 18% as conditional faults, and 44% as operating faults. (N.B.: The author offered the following definitions: *behavioral faults* – "incorrect behavior, not conforming to requirements", *conditional faults* – "incorrect condition or limit value", and *operating faults* – "omission or unnecessary operations.") The author concluded that the primary cause of safety-related, functional faults (62% on Voyager and 79% on Galileo) was due to requirements that had not been identified by the developers. Included with the author's six recommendations to software developers were the use of formal methods in the specification of requirements and the inclusion of appropriate software responses to unexpected conditions and values. [54]

#### 4. Logic Errors

Delores Wallace and Richard Kuhn analyzed software faults from 342 medical systems and determined that 43% of the software faults were logic-related errors such as incorrect logic in requirement specification, unexpected behavior of multiple conditions occurring simultaneously, and improper limits. Additionally, Wallace and Kuhn attributed 24% of the software faults to calculation errors to include incorrect limits and ranges as well as implementation of mathematical expressions. [81] Among the recommendations, the authors suggested that software engineers should consider formal methods for highly complex systems with emphasis on pre- and post-conditions as well as the interaction of system functions and unforeseeable combinations that can cause cascading failures within the system-of-systems.

## B. BATTLE-MANAGEMENT ISSUES

### 1. Predictable System Behavior

Given that the interconnected battle-management solutions in the system-of-systems environment are separately designed and developed on various operating platforms in different languages, predicting battle-management behavior of the system-of-systems is not possible. As a rule, battle management is still executed at the system level rather than the desired system-of-systems level.

Another factor that contributes to the challenge involved in predicting battle-management behavior is the development practices currently employed in DoD. The increased pressure to rapidly move product into the operational battlespace tends to channel program managers into focusing on achieving functionality as quickly as possible. As such, the development community responds with a hurried and oftentimes inadequate design phase, and follows with an intense period of coding. In the rush to rapidly develop a product, one can fall into the trap of exclusively seeking some level of achieved capability while ignoring the behavior of the software.

Defense acquisition organizations typically develop a system-of-systems by interconnecting multiple processors with a defined communications medium. Rather than focusing on shaping the behavior of the system-of-systems, the focus of the interconnectivity scheme is the exchange of messages that follow a prescribed protocol standard. The interpretation of the messages and the resultant actions are left to the individual acquirers of the various systems; however, the protocol standards oftentimes include limited behavior rules such as the rule for determining which system should report a track in a tactical data-link network. Also, the protocol standards do not include requirements for handling runtime faults other than error-checking of transmitted messages (e.g., parity checks, Hamming codes, cyclic redundancy checks). Thus, handling runtime faults is left to the developers of the individual systems in the system-of-systems.

Because each system in the system-of-systems develops the implementation of the interconnectivity standard independent of the other systems, there is no guarantee that all the implementations will result in consistent behavior by the system-of-systems. As such, the warfighters cannot predict the reactive behavior of the system-of-systems to external events.

Given the short duration of the BMD fight, the warfighters cannot accept the risk of inconsistent behavior and undesired outcomes. The serious nature of the consequences of missed engagements drives the requirement for correct, predictable behavior in the system-of-systems as well as the ability to handle all system faults during runtime. Additionally, the system-of-systems must be able react immediately to external events so it must be available for use twenty-four hours of every day.

### 2. Distributed System Environment

It is true that any fool can interconnect a set of computers by following given standards for messaging protocols and physical connections; however, it takes deliberate, knowledgeable engineering to achieve the desired system behavior in a system-of-systems. Typically, Defense acquisition organizations concentrate on the functionality of individual system applications while giving limited attention to the principles of distributed system theory. As a result, the developers may find that the system-of-systems has a fragile dependency on absolute timing synchronization for system-of-systems computations. Additionally, the developers may learn that the heterogeneity of the systems in the system-of-systems results in a significant level of difficulty in integration with respect to continued consistent behavior through independent system software releases.

In operations, the warfighters may experience a high level of confusion and frustration as a result of the inconsistent outcomes of various battle-management computations. Without deliberate engineering efforts to handle out-of-tolerance latencies or non-functioning communication mediums, the warfighters will not be able to distinguish the continued wait for the termination of a computation from the result of a slow process or that of a failed processor. Subsequently, the defense of the battlespace may suffer the consequences of missed engagement opportunities.

### 3. Real-Time System Considerations

C2 systems are usually non-real-time systems. As such, C2 systems typically depend on synchronous messaging schemes for operations. Traditionally, weapon systems are real-time systems. Thus, required message synchronization between a non-real-time C2 system and a real-time weapon might cause an inadvertent interrupt of a real-time process that could result in deadlock and race conditions. If interrupts are disabled during critical-section processing, then the interrupt for message synchronization will be missed and the message could be lost. Lost messages could result in missed engagement opportunities in the battlespace.

A computation that is late may result in late or missed engagement opportunities. The kill chain has definitive deadlines which must be met by the BMDS or the warfighters cannot successfully defeat ballistic missile attacks. If we are to match the performance of the weapon systems and avoid the negative impact of forcing synchronization of the battle manager with the weapon system for messaging, then we should develop the battle manager as real-time software.

### 4. Software Architecture

The software architecture can impact one's ability to understand and modify the software. [31] If the software maintenance team does not understand the design and behavior of a system, then adding additional features or improving the characteristics (e.g., performance, correctness, robustness) of the software may be difficult. If the executable code was compiled as a single, monolithic software program, then the software maintenance team may experience a significant level of effort in providing new features in the software as the team may require that the operational system be taken offline so that the old software version can be downloaded and the new software version can be uploaded.

### 5. Safety Considerations

One safety consideration is the interconnectivity construct in the system-of-systems. Because the development focus is frequently on correct message transmission and receipt, there is no viable means of ensuring safety at the system-of-systems level. As such, an erroneous output in a given system can be quickly propagated throughout the system-of-systems with potentially catastrophic results. For example, the inadvertent typing of a manned space vehicle as a ballistic missile threat by a sensor could result in the inappropriate destruction of the manned space vehicle by a weapon in the BMDS.

A second safety consideration is the potential situation in which two processes have simultaneous access to a critical section during runtime. (N.B.: We define the critical section to be a shared resource in which multiple processes may access during runtime.) The software of the critical section must execute without interruption; otherwise, the system software could experience deadlock and race conditions. A *deadlock condition* can occur if a process waits indefinitely for conditions that will never be satisfied. For deadlock to occur, all of the following four conditions must be true: (1) processes claim exclusive control of shared resources, (2) processes hold shared resources while waiting for other shared resources to be released, (3) processes cannot be directed to release shared resources, and (4) a circular waiting condition exists for the release of shared resources. [30] A *race condition* can occur if the final result of a computation that requires access to a critical section is executed by two or more processes, and the final result of the computation depends on the order in which those processes execute. For example, if two processes ($P_A$ and $P_B$) write different values $V_A$ and $V_B$ to the same variable in a critical section, then the final value of the variable is determined by the order in which $P_A$ and $P_B$ execute.

# IV. ASSESSMENT OF PREVIOUS WORK

## A. BACKGROUND

So far we have argued the case that developers have experienced limited success in designing, developing, and delivering a dependable system-of-systems. We presented two examples in which military system-of-systems exhibited unknown behaviors which resulted in the loss of American and allied lives. At this point in the dissertation, we review the literature on system-of-systems development to provide some insight into the root causes of systems-of-systems development failures. We would like to assess the proposed solutions and potentially offer solutions that could address the shortcomings in a system-of-systems development.

## B. STATE OF SYSTEM-OF-SYSTEMS RESEARCH AND DEVELOPMENT

In July 2004, the Potomac Institute for Policy Studies hosted a series of discussions of issues concerning systems-of-systems. [83] The group noted that traditional system engineering is based on the assumption that engineers can build a system if provided a complete set of requirements. They note that the defining of requirements for a system-of-systems may be an intractable problem because the complete operational concepts on how a system-of-systems might be employed can never be completely known at the requirements elicitation phase. They state that the Federal Aviation Administration attempted five times to build a completely new system from the ground up and was not successful in any of the attempts. The group asserted that systems-of-systems are open systems in the sense that a system-of-systems does not have fixed and stable boundaries. The group noted that emergent behavior in a system-of-systems is a critical concern and further noted that tools do not exist for developers to deal with the emergent behavior of the components within an individual system and the system-of-systems as a single entity. The group offered the Unified Modeling Language (UML) as a possible tool for developers to handle the emergent behavior of a system-of-systems.

Mark Greaves, Victoria Stavridou-Coleman, and Robert Laddaga noted in [99] the following: "It is well known that building dependable software systems for dynamic environments is difficult. It is also well known that building large-scale distributed software systems is difficult. The relatively few attempts to combine these two tasks confirm that successfully building large-scale distributed systems with predictable properties is exceptionally difficult."

The authors observe that the increasing demand for systems-of-systems will require engineers and computer scientists to design, develop, and deliver highly flexible and dependable systems-of-systems that exhibit highly predictable behavior. Furthermore, they note that the traditional techniques for designing and developing dependable software will be "difficult or impossible to employ" in a system-of-systems environment.

In [84], Dennis Smith, Edwin Morris, and David Carney discuss the issues associated with the development of a system-of-systems that include incomplete requirements, unexpected interactions, and unshared assumptions. The authors claim that "strict specification of standards" is not sufficient for achieving the desired level of interoperability in a system-of-systems as the various developers of the individual systems can interpret specifications differently. The authors further note that achieving and maintaining interoperability among the systems is difficult due to the inherent complexity of the individual systems, and the number of potential interactions between and among systems. According to the authors, the technical innovations in software engineering have not fruitfully addressed the system-of-systems development problems. They cite an example in the United States automobile supply chain in which inadequate interoperability among complex systems costs one billion dollars each year.

Smith, Morris, and Carney claim that traditional software engineer state-of-the-practice assumes a complete and precise understanding of the system in development. They state that integrators of the independent systems hold different assumptions and beliefs about the system-of-systems than that of the developers of the individual systems. As such, system-of-systems developments can fail due to conflicting and incomplete information. They cite emergent properties of the system-of-systems pose the greatest challenge to developers for predicting dependable system-of-systems.

The authors offer that developers require new approaches to establish and maintain interoperability in a system-of-systems. These new approaches include: (1) assess proposed requirements and architectural changes to the system-of-systems as well as each individual system, (2) develop a system-of-systems architecture that minimizes the impact of change, and (3) verify proposed interoperability solutions prior to fielding the system-of-systems.

Robert Schaefer offers in [100] that developers of systems-of-systems face significantly more complex integration issues than developers of single systems. He asserts that "[i]mproving the development process is not enough" and "[l]anguage and tools are not enough." According to Schaefer, the current set of system integration tools "… do not yet fill the needs for debugging large systems." He claims that placing additional controls over the development process for a system-of-systems may not prove to be more cost effective than developing a system-of-systems architecture. Schaefer notes that independent systems may exhibit the desired behavior; however, when the independent systems are integrated into a system-of-systems, system faults can occur "…irrespective of good architecture and design practices." He contends that middleware is "…both a breach in the [system interface] firewall that can propagate faults and a critical weak link when the middleware software itself fails." Furthermore, he states that a system build from a number of components will be only as dependable is its weakest serial component. That is, the component with the "poorest fault handling capabilities" will determine the level of dependability in the system-of-systems. Schaefer holds that advances in systems architecture may be the key to successful integration of the independent systems in a system-of-systems.

Cliff Jones and Brian Randell wrote in [85] that it is "…easy to bemoan the fact that computer systems are less dependable than one might want, but it is essential to understand that large networked computer systems are among the most complex things that the human race has created." They further noted that "…present trends and predictions indicate that huge, even globally-distributed, networked computer systems, perhaps involving everything from super-computers and large server ‘farms’ to myriads of small mobile computers and tiny embedded devices, are likely to become highly pervasive.”

The authors cited the following as examples of the costs of undependable systems:

1. “The average cost per hour of computer system downtime across 30 domains such as banking, manufacturing, retail, health insurances, securities, reservations, etc. has recently been estimated at nearly $950,000 by the US Association of Contingency Planners.

2. The French Insurer’s Association has estimated that the yearly cost of computer failures is [approximately $2,000,000,000] of which slightly more than half is due to deliberately induced faults, for example, by hackers and corrupt insiders.

3. The Standish Group’s ‘Chaos Chronicles’ report for 2003 analyzed over 13,000 IT projects and estimated that nearly 70 percent either failed completely or were ‘challenged’ that is although completed and operational, exceeded their budget and time estimates and had less functionality than originally specified. This led to their estimate that in 2002 the US ‘wasted’ $55 billion in cancelled and over-run IT projects compared with a total IT spend of $255 billion.”

Jones and Randell report that the European Commission’s Accompanying Measure on System Dependability Overall Dependability Roadmap 2003 estimates that “…for large and complex computer systems, namely those involving 1-100 [million] lines of code, current development techniques…can at best produce systems that achieve a level of reliability in the range of 10 to 100 failures per year.”

The authors claim that current formal methods and tools are not “readily applicable” to large and complex systems. They do state that formal methods can play a significant role in the development of large systems; however, they believe that the state-of-the-practice may limit formal methods to “small, highly-critical areas.”

John Knight discussed computing system dependability in [86], in which he states "It is important that computer engineers, software engineers, project managers, and users understand the major elements of current technology in the field of dependability, yet this material tends to be unfamiliar to researchers and practitioners alike." According to Knight, the "…trick to dependable design is to make sure that failed components within a system do not lead to system failure." He points to the specification of system requirements as the Achilles' heel of software developments. He notes that while verification techniques such as formal methods and model checking are available, software developers depend on testing as the dominant approach for verification. Knight observes that testing "remains problematic" in that "…it is impossible to execute a sufficient number of tests to permit a statistical assessment of extreme dependability…."

A wealth of literature exists that suggests the use of formal methods can benefit software developers in the verification of requirements. Likewise, there is a significant amount of literature in which attempts are made to debunk the purported myths surrounding the use of formal methods. [5][42][60][79] Bob Lang from the Software Engineering Institute (SEI) offers the following observation:

"Formal methods have long offered the promise of ensuring high quality software using mathematical rigor. The director's message in the Spring 2001 issue of news@sei points to one article suggesting that 40 to 50 percent of programs contain nontrivial defects. Formal methods represent a clear attempt to address such concerns. However, applying traditional formal methods to a complete system design requires a significant investment – from learning a difficult technology to applying it in all phases of the development effort." [47]

As a result, while there have been successful developments that employed formal methods, software developers have not embraced and employed formal methods in the development of large, complex systems. In [8], the authors describe several research projects in which researchers working for the National Aeronautics and Space Administration (NASA) are applying formal methods.

Charles Keating et al. noted in [87] that the challenge to engineers of developing and integrating large, complex systems. They observe that system-of-systems engineering is "…emerging as an attempt to address integrating complex meta-systems. However, [system-of-systems engineering] is in the embryonic stages of development and lacks consistent focus." Furthermore, the authors claim that system-of-systems engineering is being largely addressed as an information technology issue with the general objective of "getting everything to work together." Additionally, they assert that the current state of system-of-systems literature that could support practicing engineers is a "…fragmented collection of seemingly disparate perspectives on the associated phenomena."

Goran Mustapic et al. list seven system-of-systems developments that were successful albeit perhaps not on the scale of large, global system-of-systems developments. [88] The authors focused on the software architecture aspects of these developments. These systems will all experience system evolution as requirements are expressed and new technologies are introduced. What seemed to be common in the successful programs was a component-based structure in a layered architecture. Additionally, the trend in the architecture seemed to be isolating the controller from other parts of the system. Finally, another trend that seemed to be a factor in the success of these efforts is that the development and integration of the system-of-systems was within a single company. This is not the conventional situation for a system-of-systems in which the independent systems are typically designed and developed independently of the system-of-systems. The authors noted that potential difficulties could arise if the architecture can no longer support new requirements and technologies; however, they believe that the architecture techniques described could be applicable to large, complex systems-of-systems.

Mark Maier observes in [89] that system-of-systems should be a different class of development than a single system due to the independence of the individual systems in a system-of-systems and the emergent behavior of a system-of-systems. Maier states that a system-of-systems is defined by its communication standards. He cites Integrated Air Defense as a system-of-systems as well as the Internet and intelligent transportion systems (e.g., advanced traveler information services and advanced traffic control systems). He believes that the greatest opportunity to develop a system-of-systems is at the interfaces which are where he sees the greatest dangers in the development of a system-of-systems. According to Maier, if the individual systems in a system-of-systems are operationally and managerially independent, then the work of the architect is at the system-of-systems interfaces. For example, if an architect separates the sensors and weapons as independent systems, then the integrated air defense system is the battle management network. He offers that communications is the primary enabling technology for a system-of-systems. Maier states his belief that successful information exchange is the key to a successfully functioning system-of-systems. He supports the concept of better communication standards to improve the degree of success in system-of-systems development.

David Fisher and Dennis Smith stated the following in [90]: "Most systems of systems use their component systems in ways that were neither intended nor anticipated. Assumptions that were reasonable and appropriate for individual component systems become sources of errors and malfunction within system-of-systems." They state that the effect of emergent properties in a system-of-systems is potentially the greatest development risk in a system-of-systems given that developers cannot predict the dependability of the system-of-systems before fielding.

Fisher and Smith further note that unbounded system-of-systems are the typical products of the military and commercial applications with challenging requirements. They offer that unbounded systems exhibit emergent behavior that cannot be predicted in advance by developers. Furthermore, they state that the techniques used to develop dependability in closed, tightly coupled, and completely defined systems will not produce the same level of dependability in unbounded systems.

The authors note that developers have typically employed such interoperability methods as enforcing a single, central control on the system-of-systems, imposing stronger standards, and demanding increased coordination mechanisms. Fisher and Smith state that these methods become less effective as the complexity and degree of distributed computing increases in a system-of-systems. As a result, a system-of-systems can experience an increase of system faults and user errors. Furthermore, they state that the frequency of accidents in systems-of-systems increases with the degree of coupling of the components, degree of a central control, overly specified requirements, and “broadly imposed” interface standards. They recommend that loose coupling of components should be an objective of the developers of systems-of-systems.

Donna Rhodes and Daniel Hastings state in [91] that “…classical Systems Engineering is not well suited to dealing with the global and social-technical aspects of the 21st century engineering systems….” They believe that classical system engineering practices should be adapted and expanded to address the engineering and development of highly complex systems. Furthermore, Rhodes and Hastings believe that Systems Engineering will experience a “significant evolution” given the increasing complexity of technology globalization of users as well as emerging systems models such as network-centric structures and system-of-systems developments.

In [92], Pin Chen and Jun Han state that immaturity of the development practices and increasing complexity of a system-of-systems drive the need to develop new approaches for developing such systems. The authors claim that the individual systems must be compatible within the system-of-systems architecture to realize an effective system-of-systems. They offer that developers must find solutions for identified architectural gaps which signify non-compatible systems.

Ivy Hooks proposes in [93] that basic requirements principles are more essential for the development of a system-of-systems than for a single system. The basic requirements principles include developing the operational concept for the entire life-cycle of the system-of-systems and identify verification methods for each requirement. She further states that standards may be the key to developing an effective system-of-systems. Finally, she offers that each system in the system-of-systems must develop defensive and self-healing requirements to protect itself from undesired behavior at its interface to the system-of-systems.

Andrew Sage claims in [94] that risk and conflict thrive in the system-of-systems development environment due to the conflict of individual system goals and system-of-systems goals. He states that developers can observe the behavior of a system-of-systems; however, the interplay of individual systems cannot be considered by the study of individual systems. Furthermore, Sage claims that emergent behavior evolves from the interaction of many systems but it cannot be predicted from the knowledge of the individual systems.

In [95], Joseph Kasser offers that the development of a system-of-systems is not as complex as others may believe. His premise is that a system-of-systems development is ad hoc and uncontrolled. He offers that centralizing the program management activities of all the systems within the system-of-systems will lead to a successful development.

In May of 2003, the Schools of Engineering at Purdue University identified the concept of system-of-systems as a focused research area. [96] The objective of this focused research was to develop new techniques to support the development of a system-of-systems. In the paper that describes this effort, William Crossley noted that the ability of the individual systems to operate independently within a system-of-systems increases complexity above the level of complexity of a single system. He notes that developers have a significant challenge in optimizing performance of a system-of-systems without creating computational bottlenecks and eliminating conflicting commands from the controller of the system-of-systems. Crossley states that approaches required for predicting dependability in a system-of-systems may not be available to developers. He further notes that the optimized control of a system-of-systems may compete for resources against individual systems in the system-of-systems. He concludes his paper with the recommendation for a significant amount of research to solve the system-of-systems development problems.

In May 2004, Eliot Christian, who was representing the Federal Geographic Data Committee of the United States Geological Survey, presented a briefing to the Industry Workshop on Global Earth Observation System of Systems (GEOSS) that described the proposed architecture of GEOSS. [97] The architectural team proposed a distributed system-of-systems that would include an observation component, data processing and archiving component, and a data exchange and dissemination component. The architectural team proposed that interface specifications focus upon how the components interoperated with each other. They believed the means to achieve this level of interoperability was a set of open, international standards. The architectural team identified UML, XML, Web Services Definition language (WSDL), and Common Object Request Broker Architecture (CORBA) as means to realize the interfaces of components. The architectural team proposes that GEOSS will be assured of verifiable, scalable, and interoperable interfaces by imposing standard service definitions on interface interoperability specifications on components within a complex system or discrete systems within the system-of-systems.

Craig Stoudt offers in [98] that the air traffic control system will transform "…from a ground-based, centralized, loosely integrated system to a space-based, decentralized, tightly integrated system-of-systems." According to the author, decentralized systems tend to be "inefficient and lead to conflict." He points out that the advantages of decentralized systems are increased dependability as compared to a centralized system. As other authors stated, the emergent behavior of a system-of-systems can reflect the desired behavior as well as unpredicted and undesired behavior.

## C. FINDINGS

1. The increasing demand for systems-of-systems will require engineers and computer scientists to design, develop, and deliver highly flexible and dependable systems-of-systems that exhibit highly predictable behavior.

2. Building large, complex system-of-systems that exhibits predictable behavior and is dependable is among the most complex endeavors of the human race.

3. Classical systems engineering techniques may not be well suited to dealing with the design and development of system-of-systems.

4. The body of knowledge on developing a system-of-systems seems to be limited.

5. System-of-systems development seems to be ad hoc and unstructured due to programmatic shortcomings as well as architecture and verification issues in a system-of-systems.

6. Developers require new engineering skills and tools to support the development of a dependable system-of-systems that exhibits predictable behavior.

7. It is important that computer engineers, software engineers, project managers, and users understand the major elements of current technology in the field of dependability, yet this material tends to be unfamiliar to researchers and practitioners alike.

8. Developers require new approaches to establish and maintain interoperability in a system-of-systems. These new approaches should include: (1) assess proposed requirements and architectural changes to the system-of-systems as well as each individual system, (2) develop a system-of-systems architecture that minimizes the impact of change, and (3) verify proposed interoperability solutions prior to fielding the system-of-systems.

9. The emergent properties in a system-of-systems may be the greatest development risk given that developers cannot predict the dependability of the system-of-systems before delivering it to the users.

10. The ability of the individual systems to operate independently within a system-of-systems increases complexity above the level of complexity of a single system. System-of-systems developers have a significant challenge in optimizing performance of a system-of-systems without creating computational bottlenecks and eliminating conflicting commands from the controller of the system-of-systems.

11. The increasing detail in the specification of standards will not be sufficient for achieving the desired level of interoperability in a system-of-systems as the various developers of the individual systems can interpret specifications differently. Achieving and maintaining interoperability among the systems in a system-of-systems is difficult due to the inherent complexity of the individual systems, and the number of potential interactions between systems.

12. The specification and verification of system requirements in a system-of-systems may be the Achilles' heel of software developments. Although verification

techniques such as formal methods and model checking are available, software developers depend on testing as the dominant approach for verification.

13. A component-based structure in a layered architecture could be a successful technique in the development of a system-of-systems.

14. Formal methods may have a significant role in the design and development of a system-of-systems; however, current formal methods and tools may need to be extended to support the development of large, complex system-of-systems.

## D. KEY TOPICS TO BE ADDRESSED

Although there would seem to be a global trend towards composing systems-of-systems, there appears to be a significant void in both research activities and the engineering practices for the development of a system-of-systems. Esteemed researchers from the Massachusetts Institute for Technology and the Purdue University have expressed this observation. The Purdue University has established a focused research area in the engineering of a system-of-systems with the objective of filling the void.

The architecture of the system-of-systems seems to be a critical issue for the developer of a system-of-systems. A suitable architecture may be a valuable first step in the successful design and development of a system-of-systems.

A component-based structure could be one characteristic of successful system-of-systems developments. With components, we might consider techniques in which we can isolate the behavior of a given component from all others with the objective of isolating faults within the component.

Formal methods could be useful in the design and development of a system-of-systems. It seems as if previous research considered the entire system-of-systems rather than decomposing the system-of-systems into components for which formal methods and model checking could provide value. We might consider techniques in which we can isolate state behavior to reduce the number of reachable states in the components.

# V. RESEARCH

## A. RESEARCH QUESTIONS

We believe that it is possible to develop globally distributed, real-time controlling software for a system-of-systems that exhibits highly predictable system-software behavior. We make the assumption that it is impractical to realize a significant level of changes in legacy software of the independent systems within a system-of-systems. To a lesser degree, this assumption could hold true for systems that are currently under development. If these assumptions hold true, software engineers can design and develop controlling software for a system-of-systems that exhibits a high level of trustworthiness.

We pose the following questions that we will address in this research:

1. Is it possible to develop a system-of-systems architecture from which we can reason about the controlling software for a system-of-systems?

2. Can we realize the controlling software from a system-of-systems architecture through the concepts of component-based software engineering?

3. Can we apply formal methods in the design and development of the controlling software for a system-of-systems by specifying the requirements for the software components with assertions and employing a runtime verification tool to verify the desired behavior specified in the assertions?

## B. RESEARCH STRATEGY

In the assessment of previous work in the area of system-of-systems, there is a recognized void in the software engineering body of knowledge for the design and development of a system-of-systems. This research extends the body of knowledge in the design and development of systems-of-systems by proposing innovative architectural and software development practices.

In the key topics to be addressed, we highlighted several suggestions that others proposed could increase the effectiveness and dependability of fielded systems-of-systems. We developed an architectural framework that ranged from the summary view of our case study for a system-of-systems to the component framework in the BMK. We offered a systematic method to develop natural language assertions from a collaboration diagram for the specification of the BMK components. We demonstrated the method for a slice of the BMK by transforming natural language assertions into temporal assertions that we demonstrated on a runtime verification tool.

To address research question V.A.1, we expanded the key topic area from Chapter IV in which the proposal was made that a suitable architecture could be a valuable first step in the successful design and development of a system-of-systems. We developed architectural views of the system-of-systems to include a framework for the controlling software in the BMDS. We treated weapon systems as being comprised of components rather than a single entity. In addition, we treated sensors as a component. We identified a controlling component for this system-of-systems that we refer to as the battle manager.

In the battle-manager framework, we identified a BMK that is the controlling software in the battle manager. The BMK connects to software components used for calculations in battle-management as well as the interfaces to external components of systems such as sensors, C2, and weapons. The objective of this framework is to show a design of a battle manager as an integration of various components rather than a single software application. [9]

While the design of the system-of-systems might require other views and additional details, we demonstrated that it is possible to create a set of architectural views that represents the independent systems in a system-of-systems and identifies controlling software for the system-of-systems. Furthermore, we established that it is possible to create a set of architectural views for the controlling software that identify the functionality of the controlling software in terms of its components.

Using our case study of the BMDS, we identified desired dependability properties for the system-of-systems. Additionally, we identified design considerations for the distributed properties of a system-of-systems. Finally, we identified design considerations for the real-time nature of the BMDS.

To address research question V.A.2, we expanded the key topic area that a component-based structure could be one characteristic of successful system-of-system developments. We separated computational work from behavioral work through the specification of passive and active components, with data stores employed between active components to isolate state behavior and potential software faults in any given active component from all other active components. We provided examples of the work as well as the thought process to develop the work.

To address research question V.A.3, we expanded the key topic that formal methods could be useful in the design and development of a system-of-systems. Recall that previous research seemed to consider applying formal methods across the entire system-of-systems as a single entity rather than decomposing the system-of-systems into components for which formal methods and model checking could provide value.

In this research, we decoupled each active component from all other active components through the use of data stores. The objective of this design technique was to isolate the effects of state behavior to a single active component. As a result, we reduced the number of reachable states in the controlling software as compared to realizing the battle-manager functionality in a monolithic program in which each component might have direct, synchronous messaging among other components.

We developed natural language assertions to define the desired behavior of the active components in the battle manager. We assessed the role of natural language assertions to define the dependability properties identified for this research: availability, correctness, consistency, reliability, robustness, safety, and recoverability. To demonstrate the feasibility of applying formal methods to a system-of-systems as outlined in this research, we developed a working model of a slice of the BMK by transforming the natural language assertions into temporal assertions and exercising the temporal assertions in a runtime verification tool; this demonstrated how formal methods can be applied to the design and development of controlling software for a system-of-systems.

## C. SCOPE

Our research addresses Parnas' six issues with the exception of the discrimination problem. We extended the suggestions for the design and development of a system-of-systems by the development of architectural views and a framework for the controlling software. We applied formal methods to the BMK to demonstrate that formal methods can play a significant role in the design and development of a system-of-systems. Finally, we demonstrated a slice of the BMK by transforming several natural language assertions into temporal assertions, and demonstrating the utility of the temporal assertions by running the assertions in a runtime verification tool.

## D. SUMMARY OF CONTRIBUTIONS FROM THIS RESEARCH

This research extends the software engineering body of knowledge for the design, development, and fielding of large, complex systems-of-systems as follows:

1. Identification of distributed-system attributes for controlling software in a system-of-systems

2. Identification of real-time attributes for real-time controlling software in a reactive system-of-systems

3. Development of system-of-systems architecture views from system-of-systems view to component view in controlling software

4. Use of kernel in the controlling software of systems-of-systems to shape the behavior of such systems to be dependable

5. Reduction of software complexity from an exponential factor for a monolithic software program to a component-based construct in which the active components are decoupled by data stores

6. Development of assertions from collaboration diagrams

7. Adapting component-based software engineering by advanced use of assertions in interface contracts between components to assert protocols surrounding the components in reactive systems

8. Providing evidence that formal methods can be applied to large, complex system-of-systems developments

# VI. DEPENDABLE SYSTEM-OF-SYSTEMS

## A. TRUSTWORTHY AND DEPENDABLE SYSTEM-OF-SYSTEMS

In general, we do not have the luxury of beginning a system-of-systems development from scratch. We must work with the systems at hand that are in development and in operational use. We cannot begin anew so we must find other methods to apply to this common development situation.

Widely accepted definitions of a trustworthy and dependable system do not seem to exist. Indeed, authors seem to blur the lines of definition in discussions of the properties of trustworthiness, dependability, reliability, and fault tolerance. In [73] Neil Storey defines dependability as the "…a property of a system that justifies placing one's reliance on it." Andrew Tanenbaum and Maarten van Steen discuss dependability with respect to fault tolerance and offer availability, reliability, safety, and recoverability as key requirements for a dependable system. [75] Ivica Crnkovic and Magnus Larsson state that trustworthiness indicates "… a user's confidence that the system will behave as expected." [15]

Whereas many descriptions of dependable and trustworthy systems can be found, we will blend the above descriptions of dependable and trustworthy for this research as follows:

> A **dependable system** is one that provides the appropriate levels of correctness and robustness in accomplishing its mission while demonstrating the appropriate levels of availability, consistency, reliability, safety, and recoverability.
>
> A **trustworthy system** is one that provides the appropriate levels of correctness and robustness in accomplishing its mission while demonstrating the appropriate levels of availability, consistency, reliability, safety, and recoverability *to the degree that justifies a user's confidence that the system will behave as expected.*

With respect to dependable and trustworthy systems, we define the following properties in the context of a dependable system-of-systems:

*Availability:* The probability that a system is operating correctly and is ready to perform its desired functions.

*Consistency:* The property that invariants will always hold true in the system.

*Correctness:* A characteristic of a system that precisely exhibits predictable behavior at all times as defined by the system specifications.

*Reliability:* The property that a system can operate continuously without experiencing a failure.

*Robustness:* A characteristic of a system that is failure and fault tolerant.

*Safety:* The property of avoiding a catastrophic outcome given a system fails to operate correctly.

*Recoverability:* The ease for which a failed system can be restored to operational use.

(N.B.: Other properties can be used to describe a dependable system; however, we selected the above seven properties as these seven properties may be a minimum set of properties for a dependable system-of-systems. Other properties such as security might be enhanced by the techniques offered by this research (e.g., a security component); however, for the scope of the research to be manageable, we will focus on the above seven properties.)

## B. CHALLENGES FOR DEVELOPING A DEPENDABLE SYSTEM-OF-SYSTEMS

We must find new development methods for producing a dependable system-of-systems that exhibits predictable behavior and fault tolerance during runtime. As suggested in the assessment of previous research in the system-of-systems, our current development techniques fail to support system developers in producing systems with predictable behavior. Almost exclusively, developers rely on testing prior to fielding the completed product to assess system behavior. Rather than discovering system behavior at the end of the development phase, developers might apply techniques that support the design and realization of desired system behavior from the earliest phases of concept development and requirements development.

In ballistic missile defense, the warfighter must have confidence that the BMDS will correctly complete the kill chain in its operational environment regardless of the conditions in the operational environment. That is, the BMDS must be a trustworthy system. [57] Otherwise, the BMDS could engage non-threat objects such as satellites and manned spacecrafts. Other impacts of not realizing a trustworthy BMDS could be failure to launch at real threat objects that could result in tremendous loss of life in the U.S. from delivered weapons of mass destruction.

In the battle-management operations of the BMDS, the computations for discrimination, correlation, weapon assignment, and kill assessment must be correct and robust. That is, the BMDS Battle Manager should demonstrate correctness in that it does the right thing all the time and it is available all the time to engage potential threat ballistic missiles. Additionally, the BMDS should demonstrate robustness in that it handles unexpected states in a manner that minimizes performance degradation, data corruption, and incorrect output.

## C. BATTLE MANAGER CONSIDERATIONS

In the BMDS, the battle manager contains the controlling software. A sensor will detect external signals (i.e., external stimuli) and process this information to send to the battle manager that will make decisions based upon the input from the sensor and send control data to a weapon for execution of tasks. As the controller in the BMDS, the battle manager must be a trustworthy and dependable system within the system-of-systems; that is, the battle manager must execute and complete its functions correctly and robustly.

The battle manager must ensure that the kill chain is correctly executed in the BMDS. In all likelihood, a ballistic missile attack will involve multiple missiles so that the battle manager will be controlling the engagements on multiple, concurrent kill chains. The battle-management software must provide a degree of trustworthiness that is commensurate with the critical functions of battle management. Software system failures could result in massive civilian casualties.

In our consideration of dependable software in the battle manager, we should consider the development of specifications, interfaces, BMK, distributed system design, and real-time design. Our seven properties of a dependable system apply to each of these areas; however, each area has unique considerations that we should consider as follows:

### 1. Distributed System Design

The behavior of the system-of-systems with respect to distributed system behavior is equally as important as the functionality designed into the applications. The correctness of the computations in the system-of-systems applications is dependent on the supporting distributed-system implementation. We will discuss distributed systems in depth in Chapter VII.

### 2. Real-Time Design

If the battlespace imposes deadlines that the system-of-systems must meet, then one should consider developing the system-of-systems as a real-time system. We should ensure that the battle manager produces its results of computation to meet the deadlines in the kill chain. This will involve the consideration of concurrency of kill-chain activities as well as establishing computational priorities of the kill-chain activities. We will discuss real-time systems as applicable to the battle manager in Chapter VIII.

### 3. System-of-Systems Architecture

As suggested in the findings of the assessment of previous work, an architecture could be a key contributor towards the successful development of a system-of-systems. We will develop architectural views of the system-of-systems and discuss the framework of the controlling software in Chapter IX.

### 4. Battle-Management Kernel

The BMK must coordinate the work of the battle manager (not to be confused with the scheduling of work by the real-time operating system). The BMK must assign the appropriate sensor support to a weapon when assigning that weapon to engage on an assigned track. The BMK must monitor each engagement through its conclusion. The BMK must control the access of shared resources (i.e., weapons and sensors) to ensure that two or more processes do not attempt to concurrently manipulate variables and parameters in the shared resources. Finally, the BMK must ensure that the applications appropriately handled runtime faults. While the discussion on developing formal specifications and testing the formal specifications to determine the level of correctness and robustness in the evolving software is applicable to the development of the software in the kernel, we will discuss the BMK in depth in Chapter X.

### 5. Components and Interfaces

Given the component-based approach that we proposed, the interfaces between the BMK and the components must provide the required services for each interface while appropriately constraining the input and output parameters of each component. We will discuss components and contract interfaces in Chapter XI.

### 6. Development of Specifications

Recall that our definition of a dependable system included seven properties: availability, correctness, consistency, robustness, reliability, safety, and recoverability. If we desire the battle manager to exhibit the appropriate levels of these properties, then we should specify what an appropriate level would be for each property. We will discuss developing and testing the formal specifications to determine the level of achievement of the seven dependability properties for the battle manager in Chapter XII.

THIS PAGE INTENTIONALLY LEFT BLANK

# VII. DISTRIBUTED SYSTEM ENVIRONMENT

## A. BATTLE MANAGEMENT IN A DISTRIBUTED ENVIRONMENT

We reviewed the battle-management issues that David Parnas identified from his assessment of the Strategic Defense Initiative. [61] Among those issues, Parnas offered that battle-management computing will be accomplished through a network of computers that are connected to sensors and weapons as well as other battle-management computers. Given that a single sensor cannot view the entire battlespace, the future direction in defense acquisition is to connect the available sensors into a network from which warfighters can obtain information for the entire battlespace of interest. Given that a single weapon cannot engage all potential threats in the battlespace, DoD development organizations are attempting to connect the available weapons into a network from which warfighters can assign the appropriate weapon to an identified threat in the battlespace of interest.

To control these resources, warfighters require a battle manager that performs computations on received sensory information and makes assignments of weapons to threats in a coordinated fashion; that is, warfighters require a hierarchical and coordinating authority to direct the engagements of identified threats in their battlespace. Since the battlespace area can be as large as the entire globe such as the case for ballistic missile defense, a single authority (i.e., centralized battle manager) may not provide a timely and effective battle-management solution. Not only do warfighters require a coordinated solution for the sensors and weapons, warfighters require a coordinated solution for the various battle managers in the battlespace.

For these reasons, battle-management solutions will require a distributed system that connects battle managers, sensors, and weapons. Indeed, the vision of network-centric warfare is to join sensors, weapons, and C4I systems for integrated warfare. [25][33][45][72][80]

In the past, system developers have attempted to connect defense systems through data links that have served only to interconnect heterogeneous systems in accordance with complex protocol standards, but have not achieve DoD's goals for system-of-systems integration. [9] The result of these interconnections is a hobbling of disparate systems that produce a system-of-systems with significantly limited dependability as defined in [75].

Since the network-centric concept implies a distributed system, we should consider the issues associated with distributed systems with respect to battle management. For a battle-management distributed system, we should be very precise in our definitions and assumptions in the development of such a system lest we design our battle manager with inappropriate properties.

## B. DISTRIBUTED SYSEM DEFINITION

It can be easy to connect a number of computers together with a given means of communications; however, it is significantly harder to cause the software in that gang of interconnected computers to perform and behave as desired. Leslie Lamport offered the following observation as a result of a continuing problem in a distributed system: "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable." Lamport's observation seems to be characteristic of defense systems-of-systems. Far too often, defense acquisition organizations equate connecting together a group of computers as effectively engineering a distributed system. Before we define what we require in a distributed system for a battle manager, we should define precisely what we mean by a distributed system.

Many definitions exist to describe a distributed system. Vijay Garg defines distributed systems in [36] "…as those computer systems that contain multiple processors connected by a communication network." Andrew Tanenbaum and Maarten van Steen define a distributed system in [75] as "…a collection of independent computers that appears to its users as a single coherent system." James Michael defines a distributed system in [58] as "…any aggregation of automation that manages and mitigates the conflicts and incompatibilities of a problem domain by generating an abstraction of the domain."

For this research, we define a distributed system as a system that has multiple processors that are connected by a communications structure. We will not include any desired characteristics of a distributed system in the definition given that the properties of an operational distributed system may include undesired behaviors; however, the system is a distributed system nonetheless.

## C. BATTLE MANAGER CONSIDERATIONS

Key characteristics of a battle manager within a system-of-systems might include the following: (1) a distributed network, (2) an operational battlespace that includes land, sea, air, and space, (3) capability to address multiple targets that can threaten a specific theater of operations or region of the world, (4) management of concurrent battlespace activities, (5) automated decision making regarding the release or hold of lethal weapons, and (6) stringent requirements for high levels of dependability of the systems that provide BMD capabilities due to the fact that the encountered threats will consist of weapons of mass destruction.

The system-of-systems in a given battlespace might include a large variety of sensors, weapons, and battle-management components that will all be large, complex software systems. The suite of weapons and sensors will most likely increase in number during the acquisition lifecycle of the system-of-systems. The characteristics of these future weapons and sensors are not well defined and will likely remain fluid for many years. [61]

We can divide the battle-manager functionality into three major pieces: birth-to-death tracking, weapon assignment, and kill assessment. In the birth-to-death tracking, the battle manager considers the first detection reports of one or more sensors, and – through embedded logic – decides which reported objects are threat ballistic missiles by assessing incoming sensor information. After the birth-to-death tracking has identified a threat object, the weapon-assignment feature will pair an available weapon with known health and status to that threat object for the purposes of engagement. Following the engagement opportunity, the battle manager will assess incoming information to determine whether the interceptor negated the threat object. If not true, then the battle manager must assign a weapon to re-engage with the threat object.

For this research, we will consider three hypothetical battle managers in a hierarchical structure: Theater Battle Manager, Regional Battle Manager, and Homeland

Defense Battle Manager. Homeland defense can have precedence over Regional and Regional can have precedence over Theater. We will assume the following statements are true for the notional battle managers in our case study:

· Information sources may be global.

· Computations must be completed without depending on other battle managers.

· Peripheral components (e.g., C2) can impact the computation.

· Threat objects may be observed at more than one battle manager.

· A sensor may be associated with multiple battle managers – all of which can request services from that sensor, but only one battle manager's request can be fulfilled at any given time.

· A weapon may be associated with multiple battle managers – all of which can request engagements from that weapon, but only one battle manager's request can be fulfilled at any given time.

· Only one weapon will be assigned at any point in time to a single threat object. That is, there will not be a situation in which two or more weapons are simultaneously assigned to engage a single threat object.

· Each battle manager may have different sensors and weapons associated with it. These configurations are dynamic in that additional sensors and weapons may be added to the control of a battle manager. Conversely, sensors and weapons may be lost to the control of a battle manager.

· Battle managers may be isolated from other battle managers due to communications failures or the loss of a battle manager in battle. Each battle manager must have the capability to continue operations as well as assume operational control of a lost battle manager's battlespace.

· Legacy systems within the system-of-systems are not easily modified due to political, funding, and technical reasons. Solutions for a battle manager should not assume significant modifications to legacy systems.

· The timelines for engagements are measured in a handful of minutes. As such, the response to external events must be timely.

The battle managers will form a distributed system into which the BMD peripheral components will interface. As is characteristic of a distributed system, the battle managers in the network will not operate from a shared clock as it is difficult to precisely synchronize the clocks of various processors in a distributed system given that some measure of variable latency is inherent in communications. Additionally, detecting failures is difficult in an asynchronous distributed system given that a process cannot easily determine the difference between a slow processor and a halted processor. Finally, all memory will be local to a battle manager and it will not be shared by other battle managers as it is difficult for any one processor in the distributed system to determine the global state of the system.

Timing is a very important issue in the battle-management problem – especially in the birth-to-death tracking feature. Consider the problem of correlating received track data from multiple sensors that have overlapping coverage in the battlespace. The ability of a battle manager to correlate received track data depends on (1) achieving and maintaining a geodetic reference, (2) removing individual sensor bias, and (3) accuracy of the timestamp embedded in the track data. (N.B.: Geodetic referencing and sensor bias are not issues for this research.)

Consider developing a track-correlation algorithm which did not require that the synchronized timestamps of tracks from different sensors. Given that a track's position in the battlespace is relative to time, it is not fathomable to correlate tracks from different sensors that do not have synchronized timing. This must be a requirement for the sensors that provide track data to the BMDS battle managers.

As such, the current solution for this problem is to use the Global Positioning System (GPS); however, this design solution places a high dependency on the performance, accuracy, and dependability of GPS. Additionally, the latency errors in GPS transmission and internal sensor time stamping must be bounded, and considered in the correlation algorithms of the battle manager. While GPS-transmission latency is understood, it is difficult to know how a sensor applies timestamp within its track processing; that is, the application of a timestamp to track data is not standardized across all sensors. One sensor may apply the timestamp as the last step prior to transmission to the battle manager. Another sensor may apply the timestamp as first observed by the sensor.

We should consider the ability to detect failures in the distributed system to satisfy the seven dependability properties that we previously identified. A crashed battle manager is not easily distinguishable by another battle manager without careful design of a failure-detection solution in the distributed system. Without a failure-detection solution, a battle manager may wait forever for a message to arrive from a failed or lost battle manager. We should consider the incorporation of redundancy methods to handle battle-management failures. For example, we could employ error-correction coding techniques to correct a designed number of transmission errors in a message. We could employ a time-out scheme that allow a battle manager to wait for a specified time before resending a message to another battle manager. We could turn to physical redundancy of either software or hardware to increase the opportunities for successful operations. We could design each battle manager to raise an exception when that battle manager determines that another battle manager has failed. For such an exception, software engineers should develop an appropriate error-handling technique to overcome a failed battle manager.

A battle manager will be aware of its local state but not the state of other battle managers. While this characteristic is appropriate for the majority of battle-management functions, it is not satisfactory for continuity of operations given the failure or loss of another battle manager. In the event of a failure or loss of a battle manager, another battle manager must assume the responsibilities of the lost battle manager without interruption of battle-management services.

To solve the problem from a failed or lost battle manager, we can employ an election algorithm to designate the coordinator in the battle-manager distributed network. If a battle manager is determined to be lost (e.g., failure to receive a health and status message from a battle manager), the coordinator would designate the control of the lost battle manager's responsibilities to another battle manager until the original battle manager rejoins the distributed network.

The implementation of an election algorithm addresses another problem among battle managers which is access to the critical section of a distributed system. For the battle managers, we will consider the critical section to be the sensors and weapons of the BMDS. Recall that a sensor can only fulfill sensor tasking requests from a single battle manager at any given time. This is also true of a weapon. The problem is one of mutual exclusion. The solution to the mutual exclusion problem must satisfy the properties of safety (i.e., two processes cannot have simultaneous access to the critical section), liveness (i.e., every request to access the critical section should be granted eventually), and fairness (i.e., requests to access the critical section should be granted in the order that these requests are made).

For safety reasons, it is not desirable to have the situation in which multiple battle managers are attempting to direct a sensor's resource-management software. Additionally, it is not desirable to have the situation in which multiple battle mangers are attempting to simultaneously direct a weapons launcher at multiple targets. These two situations could generate outcomes that are unpredictable and potentially hazardous to allied forces and assets.

For liveness reasons, it is desirable to have each battle manager eventually access the critical section as requested. Given that a sensor has redirected its field of regard to satisfy the request by one battle manager, it is not desirable to break that access until the original request is fulfilled. The battle-manager should have the ability to direct the second battle manager's request to another sensor of equal ability to satisfy the request. This situation is true for multiple battle managers' request to a weapon.

For fairness reasons, it is desirable to grant requests in the order that the battle managers generate the requests to access the critical section. Each battle manager has an essential mission to perform so each request must be granted fairly to achieve maximum ballistic missile defense.

There are numerous algorithms that provide mutual exclusion in the critical section. There are pros and cons for each algorithm. For this research, we will consider a centralized algorithm, a distributed algorithm, and a token-ring algorithm.

In the centralized algorithm, each battle manager would request access to the critical section from the coordinator. If the coordinator determines that access is available, then the requesting battle manager would be granted access to the critical section. If another battle manager currently has access to the critical section, then the coordinator cannot grant access to the requesting battle manager; however, the coordinator queues the request for when the occupying battle manager vacates the critical section. This centralized algorithm satisfies the safety, liveness, and fairness properties. Only one battle manager can gain access to the critical section at any given time. All requests will eventually be fulfilled. Requests are satisfied in the order each request is made. The downside to this algorithm is that the coordinator becomes a single point of failure in the battle-management network. If the coordinator crashes, then the battle-management may go down. Additionally, the coordinator could become a bottleneck in the battle-management network.

In the token-ring algorithm, a token is passed from battle manager to battle manager in a prescribed pattern of token passing. This token must be held by a battle manager to access the critical section. If a battle manager receives a token, it checks to see whether it desires to enter the critical section. If so, the battle manager accesses the critical section and performs its work. After it leaves the critical section, the battle manager passes the token on to another battle manager. If not, then the battle manager passes the token on to another battle manager. The token continues to circulate in the prescribed pattern until a battle manager desires to access the critical section. The distributed algorithm satisfies the safety and liveness properties; however, it does not satisfy the fairness property as previously defined. Only one battle manager can gain access to the critical section at any given time. All requests will eventually be fulfilled. Requests are satisfied as the token becomes available. Every battle manager will have access to the token during every token-passing circuit on the network. Access is fairly granted but requests are not granted with respect to the time the request is made. The downside of this algorithm is that the detection of a lost token on the network is difficult in which to distinguish from the situation in which the token is being used by any given battle manager on the network. Additionally, if a battle manager fails or is lost, the prescribed pattern of token passing is interrupted.

In the distributed algorithm, a battle manager constructs a message that indicates the name of the critical section that it wants to access. This message is sent to all other battle managers. When another battle manager receives the access-request message, it will perform one of the following three actions based upon its current state:

1. If the battle manager is not in the requested critical section and does not want to enter the requested critical section, it will send an "OK" message back to the requesting battle manager.

2. If the battle manager is in the critical section, it will queue the request for access.

3. If the battle manager is not in the requested critical section but wants to do so, it will compare the timestamp on the requesting message to the timestamp on its own request message. The battle manager honors the message with the older timestamp. If the received message has an older timestamp, then the battle manager sends an "OK" message to the requesting battle manager. If the received message has a younger timestamp, then the battle manager queues the request message and waits for action on its access request.

The distributed algorithm can satisfy the safety, liveness, and fairness properties for the BMDS Battle Manager. Only one battle manager can gain access to the critical section at any given time. All requests will eventually be fulfilled. Requests are satisfied in the order each request is made. The downside of this algorithm is that each battle manager on the network is a potential point of failure; that is, if a battle manager is lost or failed, it will not respond to access requests from other battle managers. Consequently, this "silence" will be interpreted as a denial of access by battle managers that have submitted access requests to the critical section. Additionally, every battle manager in the network is a potential bottleneck given the number of messages generated for each access request.

## D. TECHNICAL CONTRIBUTION

For the first technical contribution in this research, we identified distributed system attributes for developing the controlling software in a system-of-systems. The design of distributed systems is not a trivial problem. It involves the consideration of issues that may not be typically considered in single-node system. In the design of a distributed system, we should recognize that synchronized clocks, detection of failures, and awareness of global state are difficult to accomplish in distributed systems. Also, we should uphold the properties of safety, liveness, and fairness in the consideration of mutual exclusion solutions for the access to the critical section.

# VIII. REAL-TIME ENVIRONMENT

## A. BATTLE MANAGEMENT IN A REAL-TIME ENVIRONMENT

The battle-management system will be a reactive system. That is, the battle-management system behavior will be characterized with respect to its response to external events in the operating environment. As Parnas noted in [61], the battle-management software must identify, track, and direct weapons towards targets whose characteristics may not be known with certainty until the moment of battle. The battle-management software must discriminate the threat objects from decoys and debris. Given that the battle-management system may be processing information on up to thousands of objects as in the ballistic missile defense battlespace [13], we will require the battle-management system to concurrently process a significant number of tasks.

Parnas further noted that the battle-management software will have absolute real-time deadlines for the computation that will consist of periodic processes to include detecting and identifying potential threat missiles, assigning a weapon to engage the threat missile, and providing an assessment of the interceptor-threat missile engagement. Because of the unpredictability of the computational requirements of each process, developers cannot predict the required resources for computation. [61] We must develop an approach that bounds the computational requirements for the battle-management system.

## B. REAL-TIME SYSTEM DEFINITION

Engineers and designers oftentimes equate “real time” with “real fast.” This misperception has permeated the thinking of system engineers as they discuss “near-real-time systems” that exhibit “near-real-time performance.” This thinking may have originated in the business-systems world from the concept of “near-real-time transactions” which is intended to mean somewhat fast transactions. Regardless of the origin, the phrases “near-real-time” and “real-time” can be found in system specifications to describe the desired performance conditions of fast and very fast, respectively.

There does not seem to be a universally accepted definition for a real-time system; however, many common themes seem to permeate from the offered definitions.

In [30], Bruce Douglass defines a real-time system as "…one that has performance deadlines on its computations and actions." Hassan Gomaa states in [41] that "…real-time systems are concurrent systems with timing constraints." In [52], Jane Liu suggests that "… a real-time system is required to complete its work and deliver its services on a timely basis."

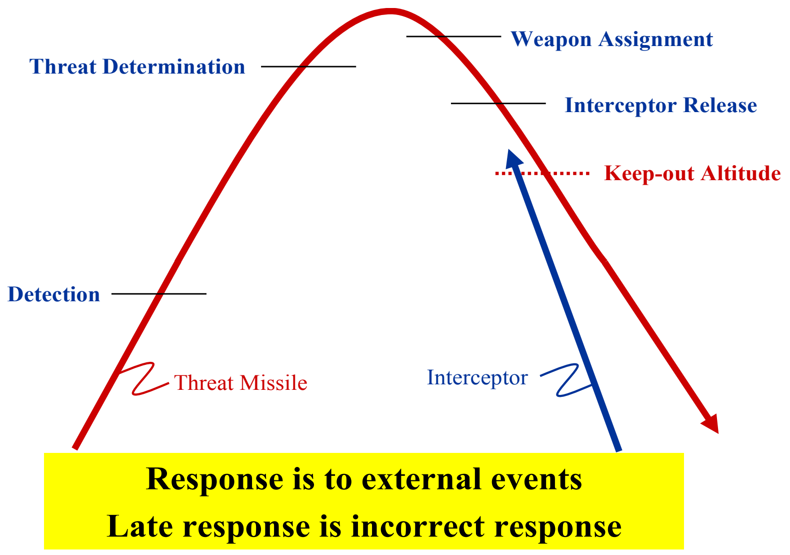
Real-time systems are frequently reactive systems in that real-time systems are event-driven and must respond immediately to stimuli from an external environment in which the real-time system exists. (N.B.: For this research, we define a reactive system as a system for which its behavior is primarily caused by reactions to external events as opposed to being internally generated stimuli. [30]) This external environment is typically non-human and involves input data from mechanical processes or alarm conditions. From sensory input data, real-time systems commonly make control decisions that are without human intervention. [41]

For this research, we define a real-time system as one for which producing correct computations as a result of an external event is equally as critical as meeting the performance deadlines for those computations.

## C. BATTLE-MANAGEMENT COMPUTATION DEADLINES

We identify the hard deadline for the battle manager in the context of the kill chain. Recall the five functions of the kill chain: Detect, Track, Assign Weapon, Engage, and Assess Kill. Along the kill chain is a point called keep-out altitude which we define as follows: The keep-out altitude for ballistic missile defense is the lowest altitude above an area on the surface of the Earth for which an engagement must occur to minimize the ground effects of debris from the engagement. The issue is that the debris from the resultant engagement will fall back to Earth, and it may contain nuclear, chemical, or biological agents that can negatively impact humans and assets in the volume of the debris fallout. The keep-out altitude is noted in Figure 1.

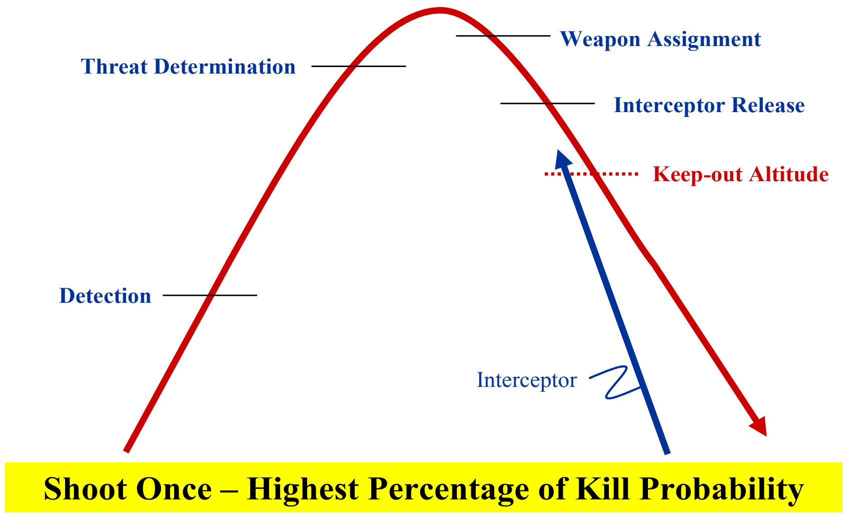
**Figure 1. Keep-Out Altitude in Kill Chain**



Note that the keep-out altitude drives the deadlines in the battle manager. Considering the fly-out time of the interceptor and the velocity of the ballistic missile, the interceptor must be released in advance of the ballistic missile descending to the keep-out altitude. (N.B.: For this research, we define fly-out time as the time difference from the time of launch of the interceptor to the time of engagement of the ballistic missile threat.) For the interceptor to be launched in time for the engagement to occur at or above the keep-out altitude, the battle manager assigns the ballistic missile track to a weapon assignment in sufficient time for the weapon system to develop a fire-control solution and launch the interceptor. (N.B.: For this research, we define the fire-control solution as the collection of calculations by a weapon system to determine the point of intercept, launch angle, and time of launch of an interceptor.) The weapon assignment in the battle manager is dependent on the determination of a ballistic missile threat in the sensor data which must be discriminated to identify threat objects, and potentially correlated to determine the true number and position of threat objects. (N.B.: For this research, we define discrimination as the capability to distinguish a threat object from benign objects such as debris, chaff, countermeasures, and satellites. Furthermore, we define correlation as the capability to associate one track with one sensed object.) Thus, the deadlines in the kill chain are: (1) identification of threat objects and (2) weapon assignment to a threat object.

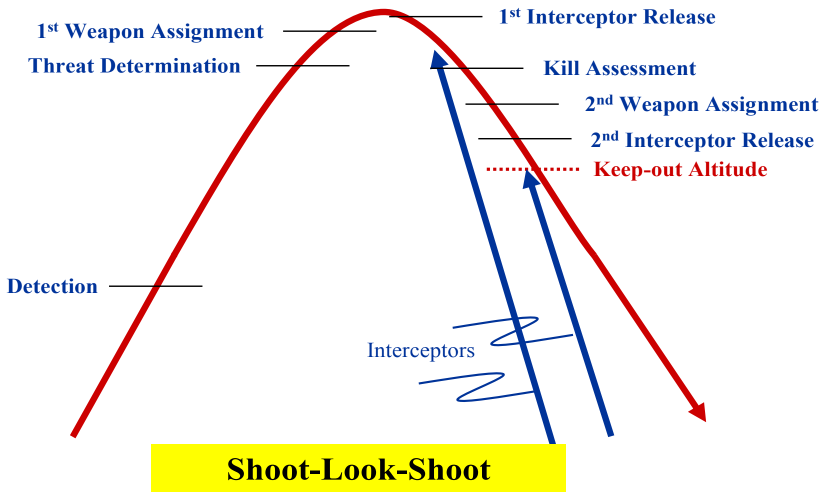
The hard deadlines must be calculated for each threat ballistic missile and will not be fixed; that is, the hard deadlines will be a function of battlespace conditions to include such characteristics as: (1) the selected shot doctrine (e.g., shoot at point of highest percentage of a kill probability as depicted in Figure 2 or shoot an interceptor – assess the kill – shoot again if necessary: shoot-look-shoot as depicted in Figure 3), (2) velocity of the ballistic missile threat, and (3) fly-out time of the interceptor.

**Figure 2. Shoot at Highest Percentage Shot Opportunity**



Note that soft real-time deadlines may exist for which the battle manager must address. We will differentiate a hard deadline from a soft deadline as follows: a hard deadline that is missed may be considered to be a fatal fault while a soft deadline that is missed is considered undesireable. For example, developers may specify a deadline for detection that should be met within 60 seconds of launch. If the battle manager completes the deadline 20 seconds late, then this is undesireable but not a fatal fault. This is an example of a soft deadline. If the battle manager completes the weapon assignment so late in the kill chain that the weapon cannot launch the interceptor in time to engage the threat at or above the keep-out altitude, then this is a fatal flaw as the ground effects from the engagement may cause either human fatalities or loss of assets. This is an example of a hard deadline.

Figure 3. Shoot-Look-Shoot



While the battle-management deadlines within the kill chain are a function of numerous variables, we offer the information in Table 1 that provides approximate time values for the required battle-management response times to a detected threat. For this research, we define response time as the time required to complete an activity in the kill chain. (N.B.: We offer the battle-management response times for addressing ballistic missile threats as a point of reference rather than absolute values. [55] offers a detailed analysis of response times for various missile types and atmospheric conditions.)

Table 1. Ballistic Missile Threat Flight Times *

| BM Class (kilometers) | Flight Time (seconds) | Boost Time (seconds) | Mid-course Time (seconds) | Terminal Time (seconds) |
|---|---|---|---|---|
| 1000 | 443 | 50 | 377 | 16 |
| 3000 | 767 | 75 | 682 | 10 |

* We offer the flight times for ballistic missiles in the three phases for minimum energy trajectories as a point of reference rather than absolute values. [55] offers a detailed analysis of flight times for various missile types and atmospheric conditions.

Given that current infrared satellite technology requires a minimum of three hits to determine a trajectory, then the minimum time to detect a ballistic missile launch is 30 seconds. These 30 seconds must be subtracted from the flight time in the calculation of the maximum available battle-management response time. Since the intercept must occur above the minimum altitude above the defended assets, then we must subtract 16 seconds and 10 seconds from the flight times of the 1000 kilometer and 3000 kilometer ballistic missile threats respectively. For the 1000 kilometer ballistic missile threat, the maximum battle-management response time to complete the kill chain is 397 seconds or 6.6 minutes. For the 3000 kilometer ballistic missile threat, the maximum battle-management response time to complete the kill chain is 727 seconds or 12.1 minutes.

So, the battle manager must detect the ballistic missile threat, track the threat, assign a weapon to the threat, authorize the launch of an interceptor, observe the engagement, conduct a hit assessment of the engagement, track any residual remaining threat, assign a weapon to any residual threat, and authorize the launch of the interceptor – all within the calculated maximum battle-management response time. [55] offers that an interceptor with a range of 500 kilometers and a final velocity of 2.0 kilometers per second would have a maximum flight time of 313 seconds for the maximum range. This would leave a maximum of 84 seconds for the detection, tracking, weapon assignment, launch authorization, and hit assessment for the defense against the 1000 kilometer ballistic missile threat. Note that there is insufficient time to launch a second interceptor if the first interceptor failed to negate the threat. The maximum time remaining for a 3000 kilometer ballistic missile threat would be 414 seconds or 6.9 minutes. This would leave sufficient time for a second interceptor launch; however, the maximum battle response time to complete the required kill-chain activities for a "shoot-look-shoot" engagement (i.e., one interceptor, hit assessment, second interceptor) would be reduced to 101 seconds.

If we equally divide the maximum available battle-management response time for each interceptor, then the timeline would be as shown in Table 2.

Table 2. Battle-Management Response Times *

| BM Class (kilometers) | Minimum Detection Time (seconds) | Available Time for Tracking, Assign Weapon, Authorize Launch (seconds) | Maximum Fly-out Time for First Interceptor (seconds) | Available Time for Hit Assessment, Tracking, Assign Weapon, Authorize Launch (seconds) | Maximum Fly-out Time for Second Interceptor (seconds) |
|---|---|---|---|---|---|
| 1000 | 30 | 84 | 313 | N/A | N/A |
| 3000 | 30 | 50.5 | 313 | 50.5 | 313 |

* We offer the battle-management response times for ballistic missiles in the three phases for minimum energy trajectories as a point of reference rather than absolute values. [55] offers a detailed discussion of response times for various missile types and battlespace conditions.

Given that a late computation in either battle-management deadlines will result in a late engagement below the keep-out altitude, the battle-management system might be a real-time system. If an engagement occurs below the keep-out altitude, then the debris fallout could result in the loss of human life and the contamination of physical assets.

The battle-management system must support the timely execution of these hard deadlines. (N.B.: For this research, we define a deadline as a point in time or a delta-time interval by which an action of the battle-management system must occur. [30]) The battle-management system must ensure that the required resources to execute its highest priority tasks are available for execution of these tasks.

The battle-management system must complete computations on threat evaluation based on sensor inputs from the ballistic missile defense battlespace and make decisions on weapons assignment without human intervention. The correctness of the computations and decisions by the battle-management system depends on the logical correctness of the computations and decisions as well as the timely termination of the computations and decisions – late computations and decisions will be wrong computations and decisions.

## D. BATTLE MANAGER CONSIDERATIONS

### 1. Interaction With External Environment

The battle manager will continually receive asynchronous inputs from various sensors to include radars and IR sensors. Based upon the location of the launch and sensors that can detect and track the ballistic missile threat, the specific sensors and number of threats will not be known to the battle manager in advance. As the battle manager determines that a track is a ballistic missile threat, it must activate the weapon-assignment software to complete the kill chain. As the weapon system engages a ballistic missile threat, the battle manager will receive sensor inputs that it will process to determine whether the interceptor destroyed the ballistic missile threat.

### 2. Timing Constraints

The battle manager must provide weapon assignments in sufficient time for the weapon to launch an interceptor for an engagement above the minimum altitude requirement of 15 kilometers for terminal-phase systems and 83 kilometers for mid-course-phase systems. [55] If the engagements fall below these altitudes, then the potential for the loss of human life is increased.

What is the risk of a late engagement below the keep-out altitude? Predictions of the loss of human life are offered in the following examples:

a. The Department of Defense has determined that a ballistic missile that delivers thirty kilograms of anthrax spores to an unprotected city could kill the entire population in the area of five to twenty square kilometers. If a major population center is defined as five thousand people per square kilometer, then the predicted loss of life from a ballistic missile carrying a payload of thirty kilograms of anthrax spores could range from 25,000 to 100,000 people. [55]

b. For a one megaton nuclear detonation at twelve to fourteen kilometers over a major population center, most materials within twelve to fourteen kilometers of the nuclear detonation will spontaneously ignite as a result of the heat generated by the blast. If a major population center is defined as five thousand people per square kilometer, then the predicted loss of life from a ballistic missile carrying a payload of a one megaton nuclear warhead could range from 1,130,000 to 1,540,000 people. [55]

The predicted loss of life for four major cities from the result of detonation of weapons of mass destruction in the United States is presented in Table 3.

Without question, the consequences of a late engagement due to a missed deadline in the battle manager could be a significant loss of human life. Thus, a late computation in the track processing and weapon assignment software in the battle manager will be deemed as a wrong computation.

**Table 3. Predicted Loss of life from Weapons of Mass Destruction**

| CITY | PEOPLE PER KM$^2$ | LOSS OF LIFE FROM 3O Kg ANTHRAX WARHEAD | LOSS OF LIFE FROM 1 MEGATON NUCLEAR WEAPON AT 12-14 KMs ABOVE CITY |
|---|---|---|---|
| Phoenix | 990 | 4,950 - 19,800 | 223,740 - 304,920 |
| Los Angles | 938 | 4,690 – 18,760 | 211,988 – 288,904 |
| Wash. D.C. | 3,236 | 16,180 – 64,720 | 731,336 – 966,688 |
| New York | 10,291 | 51,455 – 205,820 | 2,325,766 – 3,169,628 |

### 3. Concurrency

The track-processing software of the battle manager will execute discrimination and correlation algorithms on the received sensor data to identify ballistic missile threats. Additionally, the assign-weapon software of the battle manager must assign each identified ballistic missile threat to a specific weapon system. Given the potential for a high number of potential threat objects in the battlespace, the battle manager must concurrently discriminate and correlate input data for a multitude of threat objects as well as concurrently assign a weapon to each threat object and monitor the engagement for a possible re-engagement assignment to a weapon.

#### 4. Predictability

The battle manager must respond immediately to external sensor stimulation and execute its highest priority tasks as designed. In the interest of developing dependable software, we must know that the battle manager will execute its highest priority tasks without fail under all conditions in the battlespace. In the development of the battle-manager software, we must be able to determine in the design phase that the battle manager will meet its deadline requirements. Otherwise, we are leaving the meeting of the deadlines to chance.

## E. TECHNICAL CONTRIBUTION

For the second technical contribution in this research, we identified real-time system attributes for developing the controlling software in a reactive system-of-systems. In the development of the architectural views, BMK design, and BMK specification, we should consider the real-time aspects offered in this research to include the interactions between the BMDS sensors and shooters with the BMK, timing constraints, concurrency, and predictability.

# IX. SYSTEM-OF-SYSTEMS ARCHITECTURE

## A. ARCHITECTURE AND DESIGN

As Glinda the Good Witch of the North told Dorothy in her journey to find the Wizard of Oz: “It's always best to start at the beginning, and all you do is follow the yellow brick road." In the case of software development of a control function for a system-of-systems, we assert that the yellow brick road is the architecture.

Why is architecture important?

Martin Fowler offers an insightful observation about architecture in [32]. At a conference that he attended, an economist offered his analysis of the underpinning of the agile concepts in manufacturing and software development. According to the economist, one of the prime drivers of complexity is the influence of irreversibility. As software developers make a design decision upon which all future design decisions will be influenced, the design becomes irreversible at that point; that is, a design that is hard to change is, for all intents and purposes, irreversible.

The role of the architect is to find ways to eliminate irreversibility in designs. [32] The architect should ensure that today’s decision does not limit the flexibility of design decisions tomorrow. We will adopt this philosophy in this research and attempt to avoid irreversibility in the architecture of the battle manager.

In addition, developers of a system need to agree on the meaning of an architecture, in addition to distinguishing architecture from design: there are many formalisms—each with it own semantics—for specifying architectures and it is difficult to define a brightline between architecture and design.

In [12], Clements et al. differentiate between an architecture and a design as follows: an architecture is a design but not all design is an architecture. Furthermore, they tell us that an architecture “…establishes constraints on downstream activities – finer grained designs and code – that are compliant with the architecture, but architecture does not define an implementation.” In short, the architecture defines bounds so that the system can satisfy the required behavior, implementation, and quality objectives.

In [2], Bass, Clements, and Kazman define software architecture as the "…structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."[3] The externally visible properties are the assumptions that other elements may make of an element to include "…provided services, performance characteristics, fault handling, shared resource usage, and so on."

In [67], Shaw and Garlan state that a software architecture is the structural aspects of a system to include the "…organization of a system as a composition of components; global control structures; the protocols for communication, synchronization, and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance; dimensions of evolution; and selection among design alternatives." Moreover, Shaw and Garlan describe a system's architecture as the "computational components and interactions among those components." Shaw and Garlan remind us that as the "…size and complexity of software systems increase, the design and specification of overall system structure become more significant issues than the choice of algorithms and data structures of computation."

None of the preceding definitions of architecture are precise. Instead, the definitions describe what are thought to be key aspects of an architecture: structure, components, interactions of components, and system constraints.

Architecture may fit into a bin of concepts which defy a crisp, clear definition. The bin could also include interoperability, integration, quality, reliability, and other aspects of dependability. In an attempt to define one of the concepts in the bin, no one to our knowledge has given a precise, universal definition.

Another confounding factor in the attempt to define these concepts is that they may have different attributes depending on the context of the problem. For example,

[3] In the first edition of [2], Bass, Clements, and Kazman identified the primary building blocks of a system as components. Given the rising popularity of component-based software engineering, the authors changed the primary building blocks of a system from components to elements with the intent of avoiding confusion between their use of components in their definition of an architecture and the use of components in component-based software engineering.

interoperability between two word processors may have a different context than the interoperability of sensors and weapons in the BMDS.

In [32], Fowler offers that an architecture is the shared understanding of a system design. This definition implies two things: (1) an architecture represents that which results in a common comprehension of the desired system behavior, its limits, and operational environment and (2) humans must socialize that which comprises an architecture. If one followed Fowler's definition for an architecture, conceivably, all descriptions and artifacts related to a system design could be considered to be the architecture. However, the important implied statement in his definition is that humans must determine the content of an architecture: the task of delineating between architecture and design cannot be performed using a mechanical algorithm.

For this research, we define architecture and design as follows:

***Architecture:*** the collection of logical and physical views, constraints, and decisions that define the external properties of a system and provide a shared understanding of the system design to the development team and the intended user of the system.

***Design:*** the details of planned implementation that are defined, structured, and constrained by the architecture.

In these definitions, there is an implied responsibility of the system architect to collaborate with the development team and the intended system user to document the architecture. While numerous tools, artifacts, and methodologies exist on the market to support the system architect, the content and usefulness of the architecture will be dependent on the skill of the architect to select the appropriate tools, artifacts, and methodologies that bring about a shared understanding of the system design. One can argue that the shared understanding of the system design should be the primary objective of architecting a system.

We will not attempt to develop either a complete set of architectural views or the design of the controlling software for a system-of-systems in this research; however, we will develop a set of views that range from an architectural view of the BMDS to the architectural view of a component in the controlling software so that the reader might gain a shared understanding of one way to architect the controlling software in a system-of-systems.

## B. BATTLE-MANAGER BEHAVIOR

### 1. Introduction

We offer that the initial step in developing a system-of-systems architecture is to define the system-of-systems behavior. Although numerous methods exist to define system behavior, we favor the understanding of the operational concepts for the system-of-systems, identification of user goals, and the development of use cases that outline the required interactions between an actor and the system to achieve the user's goals

As we develop the user goals and use cases, we will accumulate a list of specifications for the system-of-systems that characterize the system behavior. (N.B.: For this research, we define a specification as either (1) a desired system behavior that is expressed as a feature, function, property, or capability, or (2) an undesired system behavior that can be expressed as a limitation, constraint, negative (e.g., "the system must not operate in this mode when…"), or condition. While the scope of a specification can be expanded beyond system behavior, we chose to limit the scope of the definition of specification to system behavior for this research.)

Specifications are different from the requirements of a system that acquisition organizations have produced for many years. As traditional development approaches mandated, engineers would use the formal users' documents (e.g., Mission Needs Statement, Operational Requirements Document) to develop functional requirements. (N.B.: For this research, we define a requirement as a criterion that a system must meet.) Typically, the formal requirements would define what a system must do, characteristics it must have, and levels of performance it must attain. After completing this document, the system engineer would allocate these functional requirements to developmental areas of hardware, software, and skinware (i.e., people), and expand the requirements in the development areas to include non-functional requirements such as reliability, recoverability, and usability.

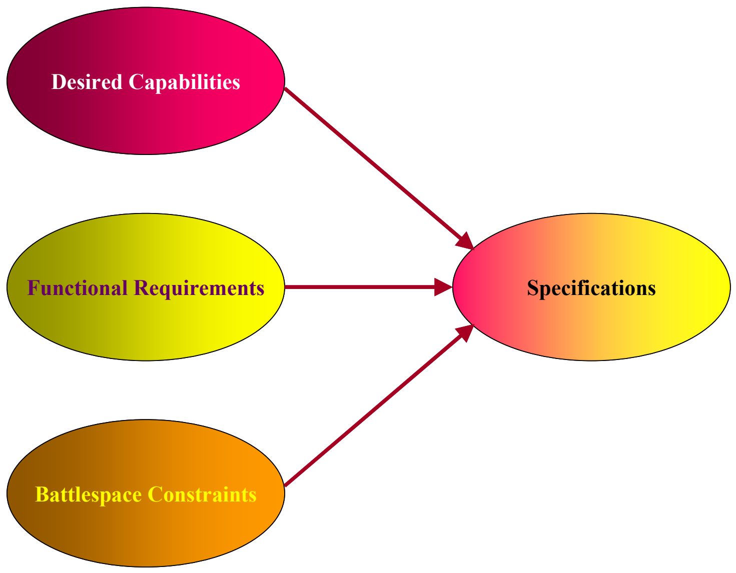
At this point in the development, the hardware engineers would acquire hardware. The communications engineers would purchase communication services and acquire communication devices. Before the software engineers could outline a software architecture, the engineers in the other developmental areas of focus would have saddled them with hardware, operating system, database application, and communication structures within which the software applications must be integrated. The less than sterling results of this traditional approach to eliciting requirements and developing systems include substantial cost and time overruns to deliver a product that contains significant reductions in delivered functionality as compared to required functionality. [49][50][69][70]

The senior leadership in defense acquisition recognized the so-called software crisis in the early 1990's and instituted significant acquisition reforms to the traditional system-development approach. Capability-based acquisition is one such modification with which defense acquisition organizations are struggling to incorporate in their development processes. (N.B.: For this research, we define a capability as the ability to perform a course of action or sequence of activities leading to a desired outcome. Furthermore, we define capability-based acquisition as the process of identifying system capabilities in terms of specifications and acquiring the software applications, hardware, and information services to support these desired capabilities in an integrated environment.) [20][26][28]

In the capability-based acquisition approach, the traditional products such as formal requirements disappeared in favor of a natural language description of desired capabilities. Although senior leadership within the U.S. DoD believe that they gain greater insight on the definition of a system as compared to the development approaches of the past, the development engineers have a daunting challenge of developing acquisition documents that can be used to develop a system. Specifically in the software area of focus, engineers recognized that software programmers cannot be handed merely a laundry-list of capabilities to code. Software engineers should translate the list of desired capabilities into specifications and design documentation to avoid the situation in which programmers must interpret the design.

To develop specifications, we should understand the source of specifications. For this research, we will adopt the methodology for developing specifications that is suggested in Figure 4. That is, we should recognize that development of specifications ought to have a level of rigor in the process to support validation and verification of the specifications. (N.B.: For this research, we define validation as the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. Furthermore, we define verification as the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [44])

**Figure 4. Information Sources for Developing Specifications**



In Figure 4, we propose that specifications represent the functional model of a system. (N.B.: For this research, we define a functional model as a system abstraction that contains the set of observations, modeling data, pre-conditions, post-conditions, invariants, boundary conditions, and algorithms that describe the physical system.) To develop the specifications of a system, we propose three sources of information be used: (1) the list of desired capabilities, (2) functional requirements, and (3) battlespace constraints. (N.B.: For this research, we define battlespace constraints as the forces, facilities, and other features that serve to restrain, restrict, or prevent the implementation of proposed military improvements in the defined battlespace. Battlespace constraints may include natural and physical forces, doctrine, threat descriptions, and environmental features.)

Thus, we offer that a system's specifications are the synthesis of desired capabilities, functional requirements, and battlespace constraints. For this research, we will anchor the development of the functional requirements to the UML use cases that explored the achievement of the user goals. We will consider the impacts of the battlespace constraints in the use cases. We will synthesize the desired capabilities with the functional requirements and battlespace constraints to form the system specifications.

To develop the functional requirements, a kill chain could be identified for the mission area of interest. (N.B.: For this research, we define a kill chain as the sequence of activities that must occur to complete a mission goal.) The kill chain should represent the high-level activities of the mission that can serve as a point of departure in developing the use cases and vision document for use in identifying the system's specifications. Given that the kill chain captures the totality of the major mission functions, then one can use the major mission functions to establish the initial set of user goals that span the mission area; that is, one can restate the major mission functions of the kill chain in terms of summary user goals with the confidence that summary user goals address the entire mission area. At that point, one can define sub-goals to the summary user goals and continue to define sub-goals as required.

With respect to the battle manager, we will identify a kill chain that defines the military activities involved in destroying a potential adversary's ballistic missile threat. Before we begin developing the use cases and vision document for the battle manager, let us examine the role of the battle manager in ballistic missile defense.

### 2. Planning, Command and Control, Battle Management

Because the BMDS Battle Manager will react in response to external events in the battlespace, it will be deemed a reactive system. (N.B.: For this research, we define a reactive system as one for which its behavior is primarily caused by reactions to external events as opposed to being internally generated stimuli.) Because the BMDS Battle Manager must meet hard deadlines along the kill chain, we propose that the BMDS Battle

Manager software be developed as a real-time system. (N.B.: For this research, we define a real-time system as one for which producing correct computations as a result of an external event is equally as critical as meeting the deadlines for those computations.)

Battle management relies on two functions that influence the outcomes of battles: planning and command and control (C2). For this research, we define planning as that military planning that produces either an Operation Plan (OPLAN) or an Operations Order (OPORD) to employ military force against an adversary. We define C2 as the exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission. Recall that our definition of battle management for this research is the decisions and actions executed in direct response to the activities of enemy forces in support of the Joint Chiefs of Staff's concept of precision engagement. [18]

Planning includes the initial lay-down of joint and coalition forces, rules of engagement, provisioning, and re-supply. Planning "sets the table" for the military and establishes the initial ruleset that the warfighters will follow at the onset of the battle. Planning is a coordinated joint staff procedure used by a commander to determine the best method of accomplishing assigned tasks and to direct the action necessary to accomplish the mission. [21] Planning includes both deliberate planning and crisis-action planning (CAP). Combatant commanders (COCOMs) conduct deliberate planning to develop a military response to a future hypothetical contingency while CAP takes place in response to a crisis in which the United States' national security interests are threatened and the President is considering a military response. [29]

C2 functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures employed by a COCOM in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission. [21] Through C2, the senior military leadership modifies and enhances the initial ruleset that governs the battlespace. (N.B.: Battlespace is defined as the environment, factors, and conditions that must be understood to successfully apply combat power, protect the force, or complete the mission. This includes the air, land, sea, space, and the included enemy and friendly forces; facilities; weather; terrain; electromagnetic spectrum; and the information environment within the operational areas and areas of interest. [21])

### 3. Kill Chain[4]

Recall from previous discussion that the Joint Staff defined Precision Engagement as follows:

> *…the ability of joint forces to locate, surveil, discern, and track objectives or targets; select, organize, and use the correct systems; generate desired effects, assess results; and reengage with decisive speed and overwhelming operational tempo as required, throughout the full range of military operations*. [23]

The basic construct of the definition for precision engagement is the identification of the functional flow of military activities that must occur to engage a threat object. This functional flow of military activities is colloquially known as the kill chain. The kill chain defines what must occur from the moment of the detection of a threat through the engagement to the determination of the negation of the threat.

Rather than capriciously defining a kill chain for the battle-management function, we treat the functional flow of events that occur in the engagement of a military threat, starting with an examination of the original work of Colonel John Boyd (USAF, Ret.) and followed by the Navy's functional construct for missile defense, the Army's functional flow of events for deep operations, the Air Force's kill chain, and the Joint Chiefs of Staff's functional flow of events for theater ballistic missile defense (TBMD).

#### *a. Observe-Orient-Decide-Act*

Colonel John Boyd was an avid student of military engagements. From his analysis of the engagement actions of commanders and famous battles, he formed a concept of what is known today as the Observe-Orient-Decide-Act (OODA) loop. He noted that in many of the engagements, one military force presented the other with a series of unexpected and threatening situations with which they had not been able to keep pace. The faster military force eventually defeated the slower military force. Boyd observed that military conflicts are time-competitive.

[4] Kill Chain discussion extracted from [10].

In the OODA Loop, Boyd incorporated a temporal aspect in his analysis of military decision-making before and during battle. Decisions and actions that are delayed are often rendered ineffective because of the constantly changing circumstances. When a military adversary is involved, the operation is not only time-sensitive but also time-competitive. Time or opportunity neglected by one adversary can be exploited by the other. [14]

According to Boyd, military conflict can be seen as a series of time-competitive cycles through an OODA loop. Each military force in a conflict begins by observing themselves, the physical surroundings, and the adversary. Next, the military force orients itself; orientation refers to making a mental image or snapshot of the situation. Orientation is necessary because the fluid, chaotic nature of conflicts makes it impossible to process information as fast as military commanders can observe it. This necessitates applying a freeze-frame concept and provides a perspective or orientation.[5] Once we have an orientation, military commanders must make a decision. The decision takes into account all the factors present at the time of the orientation. Finally, the military commander must implement the decision. This requires action. One tactical adage states: "Decisions without actions are pointless and actions without decisions are reckless." Then the cycle begins anew as military commanders believe that their actions will have changed the situation. The cycle continues to repeat throughout a tactical operation. [6]

The military force that can consistently go through the OODA loop faster than the other enemy force can, *ceteris paribus*, gains a tactical advantage. By the time the slower adversary reacts, the faster force is doing something different and the slower adversary's action may become ineffective. With each cycle, the action of the slower military force becomes increasingly ineffective by an increasingly larger margin.

The aggregate resolution of these episodes will eventually determine the outcome of the conflict. For example, as long as the actions of the faster military force continue to prove successful, the slower military force will remain in a reactive posture while the commander of the faster military force maintains the freedom to act. No matter

[5] This is analogous to creating a materialized (i.e., stored) view of data by querying a database.

how desperately the slower military force strives to accomplish its military objectives, every action becomes less useful than the preceding one. As a result, the slower military force falls farther and farther behind. [6][14]

### *b. Detect-Control-Engage*

At a Millennial Challenges Colloquium presentation in April 2000, Vice Admiral Rodney Rempt (then Rear Admiral and Deputy Assistant Secretary of the Navy for Theater Combat Systems) discussed Naval theater air and missile defense for the twenty-first century. He observed that some level of defense is the "price of admission" for carrying the battle to the shores of potential adversaries. He discussed the threat to the Fleet of cruise missiles, ballistic missiles, fighter-bombers, and unmanned aerial vehicles (UAVs); these threats are steadily increasing in lethality, accuracy, and range. Hence, Vice Admiral Rempt concluded that the Naval theater air and missile defense must formulate and apply a concept of Detect, Control, and Engage. [65]

For the detect aspect of Naval theater air and missile defense, the concepts of multi-spectrum sensor netting and data fusion must be realized from a variety of active sensor arrays, passive staring infrared sensors, and bistatic radars. The timely and accurate detection of current and future threats is absolutely essential in triggering military action to negate the threat.

For the control aspect, the Navy should realize a network of planning tools, automated decision aids, and the single integrated battle space. The Navy must develop solutions to potential threats before the threats are realized. As in all competitions and conflicts, planning and identifying potential engagement zones, rules of engagement, and consequence management will lead to the success of Naval theater air and missile defense.

For the engage aspect, the Navy should deliver the appropriate force to negate current and future threats to the Fleet and its defended assets. The received information must be processed in a timely fashion so that Naval officers can make timely decisions for engaging potential threats. Indecision due to inconclusive or untimely information can have catastrophic consequences to Fleet resources.

### *c. Decide-Detect-Deliver-Assess*

The Army defines targeting as the process of selecting targets and matching the appropriate response to them on the basis of operational requirements and capabilities. COCOMs use the functional construct of decide, detect, deliver, and assess to transform a COCOM's targeting intent into an engagement.

The objectives of targeting are to: (1) identify those resources that the enemy can least afford to lose and (2) identify the greatest weakness of the enemy that is most susceptible to attack by friendly forces. By attacking and destroying such resources as munitions stockpiles, tactical communication centers, operations centers, the enemy and his military assets are more vulnerable to the COCOMs' battle plans. Successful targeting enables the COCOM to synchronize intelligence, maneuver, fire-support systems, and in addition to special operations forces, by attacking the right target with the best system and munitions at the right time.

The decide function, as the first step in the targeting process, provides the overall focus and sets priorities for collecting intelligence and planning attacks. Targeting priorities must be addressed for each phase or critical event of an operation.

Detect is the next critical function in the targeting process. The intelligence cell is the main figure in directing the effort to detect high-payoff targets identified in the decide function. This process determines accurate, identifiable, and timely requirements for collection systems.

The deliver function of the targeting process executes the target attack guidance and supports the COCOM's battle plan once the high-payoff targets have been located and identified. Some targets will not appear as anticipated. Target attack takes place only when the forecasted enemy activity occurs in the projected time or place. The detection and tracking of activities associated with the target becomes the trigger for target attack.

Combat assessment is the determination of the effectiveness of force employment during military operations. On the basis of battle damage assessment (BDA) reports, the COCOM continuously estimates the enemy's ability to make and sustain war and centers of gravity. During the review of the effects of the campaign, re-strike recommendations are proposed or executed. BDA is the timely and accurate estimate of damage resulting from the application of military force, either lethal or non-lethal, against a target. BDA in the targeting process pertains to the results of attacks on targets designated by the commander. [18]

#### *d. Find-Fix-Track-Target-Engage-Assess*

According to General John Jumper (Chief of Staff of the United States Air Force), today's Air Force is a "community of stovepipes." General Jumper wants to achieve horizontal integration that he defines as the "…ability to fuse data from every Air Force platform into a single repository of information, such as crews, planes, targets, and loads." His vision is to achieve horizontal integration through the assimilation of the entire "kill chain" from a single source of information. General Jumper defines the kill chain as find, fix, track, target, engage, and assess. [37]

As avowed by Lieutenant General Leslie Kenne (Air Force Deputy Chief of Staff for Warfighting Integration), the Air Force must "close the seams" in the kill chain by "integration of manned, unmanned, and space systems." Historically, technology limited the flow of information. Battlefield information delivery was limited to the speed of the horses and the ability of the commander to assess the battlefield information from afar. Execution was centralized as only the commander had the situational awareness of the entire battlefield.

Consequently, reinforcement troops had no time to gain situational awareness. Thus, troops had to rely on their commander to direct their movements and placements, and hoped that the enemy had not conducted movements that countered the commander's situational awareness. [46]

Today, technology provides the potential to maintain situational awareness for the entire military force. The military has developed an interconnected network of information with the objective of providing timely and accurate information to all points of the battlespace. The stovepipes discussed by General Jumper prevent the achievement of this objective and prevent effective battle-management in the battlespace.

#### *e. Detect-Identify-Locate-Track-Destroy*

In recent years, the threat of missile attack to American forces and allies in foreign lands has dramatically increased. The proliferation of theater missiles to numerous nations, advances in missile technology, and the pursuit of weapons of mass destruction have provided potential adversaries with a lethal-attack capability against United States' interests. This fact has forced the United States to address the potential threat that these missiles pose to National security.

As outlined by the Joint Chiefs of Staff, theater missile defense applies to the "…identification, integration, and employment of forces supported by other theater and national capabilities to *detect, identify, locate, track, minimize the effects of, and/or destroy* enemy [theater missiles]." Through this process, military commanders should be capable of countering threats from theater missiles and have the capability for rapid global deployment and theater mobility. [22]

#### *f. Detect-Track-Assign Weapon-Engage-Assess Kill*

For this research, we will employ a kill chain that consists of the following five functions: ***Detect, Track, Assign Weapon, Engage, and Assess Kill***. These five functions address all the functions outlined in the definition of precision engagement to which the Joint Chiefs of Staff subscribe, in addition to all of the functions identified in the Boyd, Navy, Army, Air Force, and Joint Chiefs of Staff functional models.

Of the five kill chains described in the preceding paragraphs, only the Army and the Air Force identified an assess function that is required to determine whether the threat object is indeed negated. The assess function is essential to complete the engagement as defined by the Precision Engagement. The fix function of the Air Force kill chain is captured within the track function of our defined kill chain.

As can be observed in Table 4, the proposed kill chain is complete with respect to addressing the major functions required to negate a threat object.

## C. ARCHITECTURE

### 1. System-of-Systems Considerations

The nature of global ballistic missile defense drives the assumption that the BMDS Battle Manager must provide services in every potential theater of battle as well as the defense of our homeland:

Table 4. **Summary of Kill Chains**

| Boyd | Navy | Army | Air Force | JCS | Dissertation |
|---|---|---|---|---|---|
| Observe | Detect | Decide<br>Detect | Find | Detect | Detect |
| Orient | | | Fix<br>Track | Identify<br>Locate<br>Track | Track |
| Decide | Control | | Target | | Assign Weapon |
| Act | Engage | Deliver | Engage | Destroy | Engage |
| | | Assess | Assess | | Assess Kill |

Recall the following from Chapter II: The flight time for a ballistic missile with a range of 1000 kilometers is approximately 7.4 minutes and the flight time for a ballistic missile with a range of 3000 kilometer is approximately 12.8 minutes. Moreover, the available time for tracking, assigning a weapon system, and authorizing a launch is approximately eighty-four seconds for a ballistic missile with a range of 1000 kilometers with no possibility of a second-shot opportunity if the first shot is not successful. In addition, the available time for tracking, assigning a weapon system, and authorizing a first launch opportunity is approximately fifty-one seconds for a ballistic missile with a range of 1000 kilometers and approximately fifty-one seconds for a second-shot opportunity if the first shot is not successful.

With the knowledge of the short timelines to conduct battle management for missile defense, it is not possible to realize the battle-manager capability in a centralized fashion. That is, it is not reasonable to expect such a system to be positioned in location within the United States and require the system to direct the engagements of all possible ballistic missiles from all parts of the globe as described in Chapter I. As such, we will consider a distributed system construct for the BMDS Battle Manager. (N.B.: For this research, we define a distributed system as one that has multiple processors that are connected by a communications structure.)

The distributed battle manager must be able to communicate with all the sensors and all the weapons systems in the BMDS; however, the distributed battle manager should be transparent from the perspective of the sensors and weapons connected to it. (N.B.: For this research, we define transparent as a distributed system that appears to be a single system to the users that operate the distributed system, and the applications that reside and execute on the distributed system. [75])

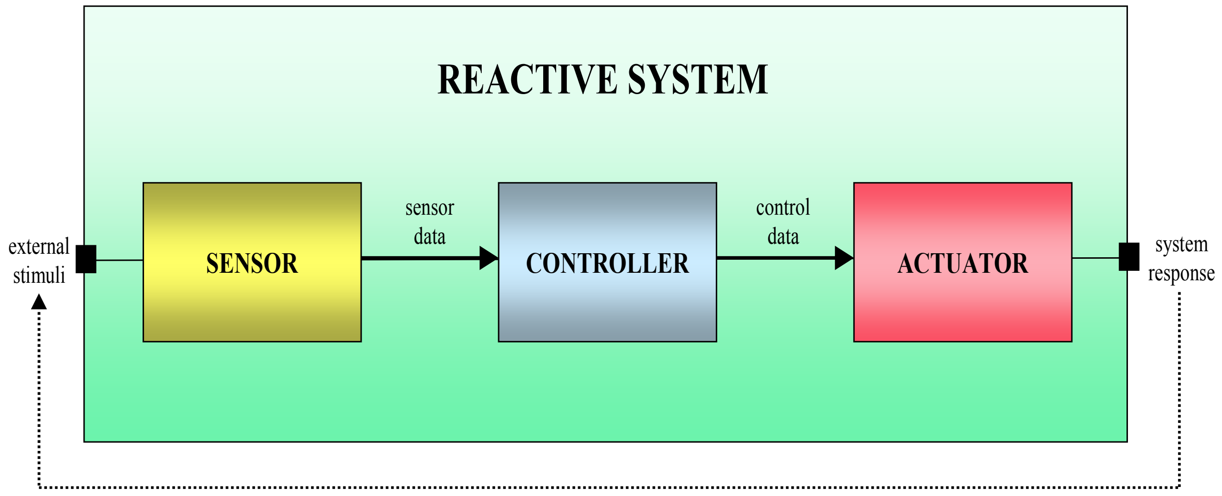
The BMDS Battle Manager may continually experience modifications and upgrades to its applications. As such, it would be useful to isolate the software that will change slowly over time from the software that will change more frequently. From the discussion of the kill chain, it seems apparent that the basic five functions of battle management will remain regardless of the methods in which we realize the battle manager. For example, the basic structure of track processing and weapon assignment may change occasionally as required. However, the computation methods used in track processing and weapon assignment are likely to change frequently as new algorithms evolve and new technologies emerge.

To this point, we have established that the battle manager must have interfaces with the sensors and weapons connected to it. Additionally, the battle manager will be a distributed system for which its properties might be transparent to the weapons and sensors connected to it. Finally, the battle manager may experience change over time. While the basic tenets of battle management will hold true, the methods of computation in the battle manager may experience frequent changes.

### 2. BMDS Architecture

Before considering the battle-manager architecture, we will set the context for the BMDS. We will model the controlling software in the BMDS as a reactive system as depicted in Figure 5.

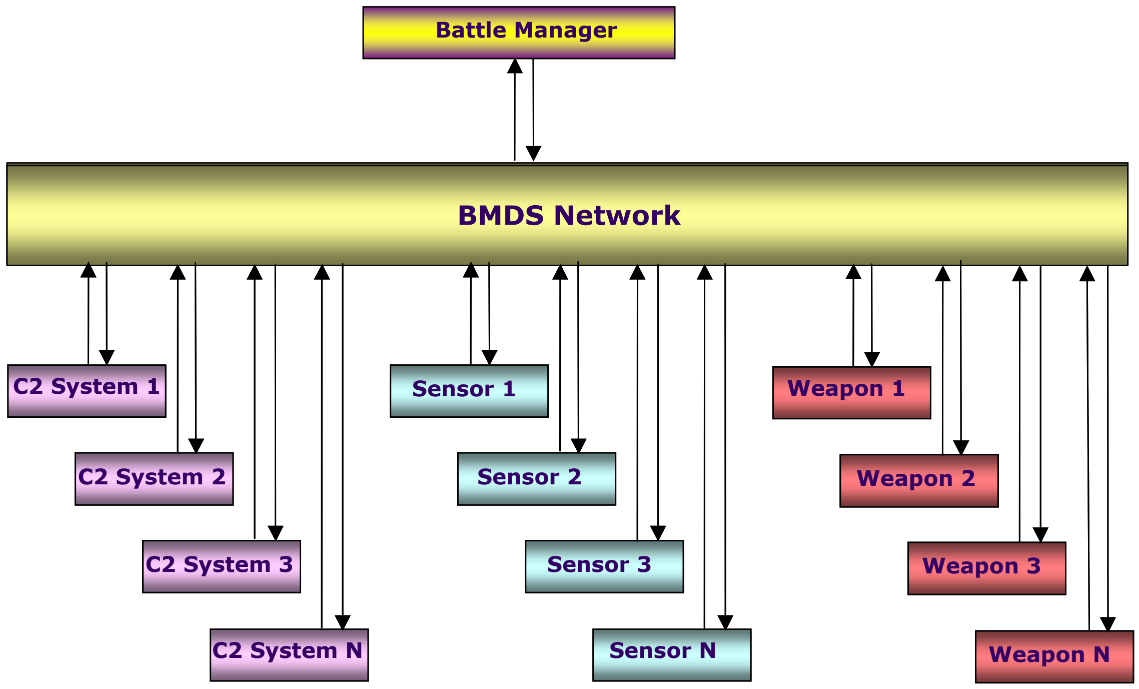
**Figure 5. Reactive System Model**



In our model, we consider the Battle Manager to be the Controller as depicted in Figure 5; that is, the Sensor will detect external signals (i.e., external stimuli) and process this information to send to the Controller. The Controller will make decisions based on the input from the Sensor and send control data to the Actuator (i.e., BMDS weapons) for execution of tasks. This results in the system response from the reactive system. The Sensor will sense new signals as a result of any environmental change that was stimulated by the Actuator and so the cycle continues.

From an external view of the BMDS Battle Manager, we depict a number of interfaces between the BMDS Network and external C2 systems, sensors, and weapons. We depict the Battle Manager interfaced with the BMDS Network as the controlling software in the system-of-systems. For this research, we define the external view of the BMDS as the logical construct of the subsystems of the BMDS from a black-box perspective. Our external view of the BMDS is depicted in Figure 6.

Figure 6. External View of BMDS Battle Manager



## 3. Battle Manager Architecture

From the perspective of the internal view of the BMDS Battle Manager, recall that we want to separate those applications that may change infrequently over time as compared to those applications that may change frequently. For this research, we define the internal view of a battle manager as the logical construct of the components that compose the battle manager. We desire to physically separate the two categories of components (i.e., those components that may change infrequently and those components that may change frequently) through a distinct interface so as to facilitate the anticipated changes of the components in the fielded system.

We propose to employ the concept of component-based engineering to design and develop the internal view of the BMDS Battle Manager. For this research, we define component-based engineering as the design and development of a system through the assembly of components which can be developed independently of the system, and we define a component as a software unit of composition with contractually specified interfaces and explicit context dependencies. We propose that the component software contain the algorithms required to perform the computations of the BMDS Battle Manager.

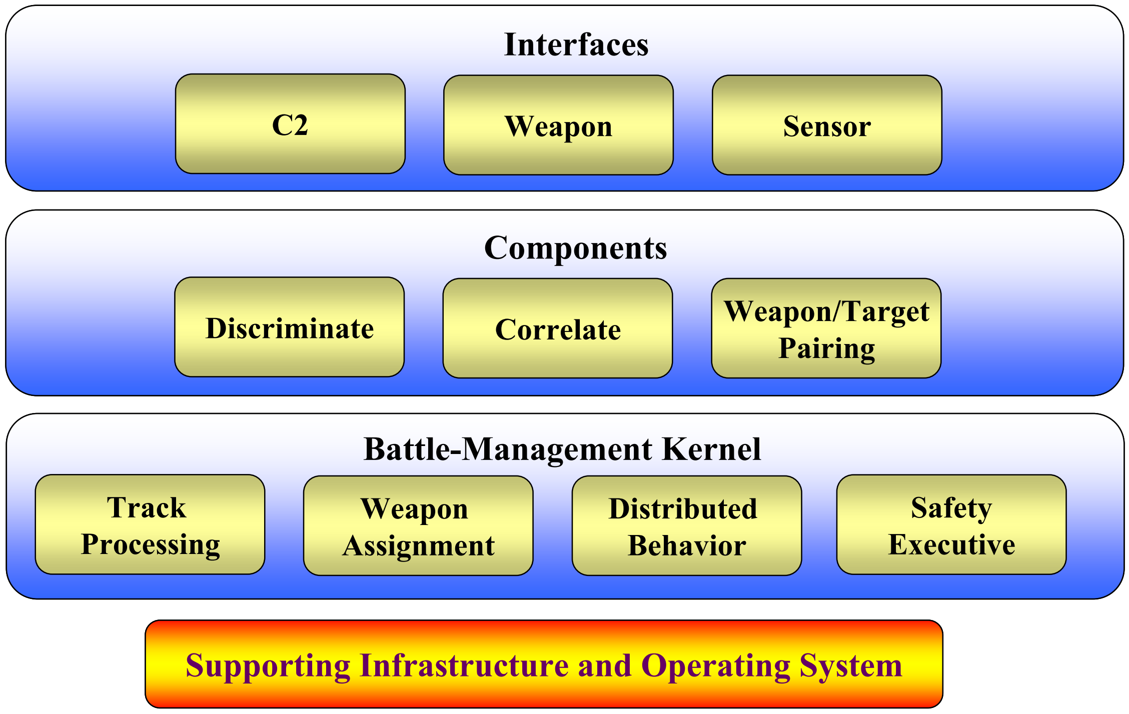
Furthermore, we propose that we develop the software that contains the basic functions of battle management as a kernel given that this software should experience limited modification over time. Derived from the kill chain [10], these basic battle-management functions are called tasks, and will manage the use of the system's computing resources to ensure that all time-critical, battle-management events are processed as efficiently as possible.

Moreover, we propose to add another software component to the kernel that controls the distributed processing in the battle manager. While we anticipate that the battle-management functions will execute on a single, multi-processor platform, the distributed functions to include control sensor-resource management, engagement control, sensor tasking, and survivability will exhibit different behavior than the kill-chain functions. As the kill-chain functions must execute regardless of the status of the distributed system, we do not desire to mix the distributed functions with the kill-chain functions.

Finally, we should consider the external data that will come into the battle manager as well as the information transported from the battle manager to external subsystems. These interfaces of the battle manager are critical for the operation of the battle manager. We must deliberately consider these interfaces and how we will handle the transported data in the system-of-systems environment.

Our internal view of the BMDS Battle Manager is depicted in Figure 7:

Figure 7. Internal View of BMDS Battle Manager



### 4. Battle-Manager Interfaces

#### *a. C2 to Battle Manager*

The C2 subsystem sets the parameters in the battle manager. Given that the different C2 subsystems may provide different parameters to the battle manager, each battle manager will employ the appropriate C2 parameters assigned to it. For instance, a theater battle-manager may be filled with rules of engagement (ROE) that are specific to that theater but not applicable to the Homeland Battle Manager. As such, the ROE that are designated for the theater battle manager must be transferred into the theater battle-manager and no others. This feature is the tailoring of a battle manager to its specific mission in the BMD battlespace.

Given that a theater battle manager may actually be multiple battle-manager platforms for the purposes of survivability, all theater platforms should receive the C2 parameters. We anticipate the C2 platform may not have visibility into the location of each battle manager. As such, we propose that the C2 and the battle manager should be decoupled from each other.

To support the decoupling of the C2 from the battle manager, we will consider the use of the publish-subscribe architectural style for these interfaces. In the publish-subscribe architectural style, subsystems subscribe to a set of events and the publish-subscribe infrastructure ensures that each published event is provided to all subscribers of that event.

The primary connector in the publish-subscribe architectural style is an event bus. C2 puts an event (e.g., ROE) on the bus by announcing the event. The connector delivers the event to each subscriber (i.e., battle manager) of that event. The C2 has no visibility into the consumers of the event data. As such, the C2 is decoupled from the battle manager as well as other potential consumers of the event data from the C2. [12]

#### *b. Battle Manager to Weapon.*

The BMDS Battle Manager will pair a specific weapon to a threat object. Unlike the C2 and sensor situation, the battle manager must have visibility into the health and status of the weapons. Additionally, the weapon must acknowledge the receipt of a target assignment. If the battle manager cannot determine the health and status of a weapon with an assigned target, it must reassign that target to another weapon that reports a positive health and status.

It seems reasonable for each weapon to post its health and status at prescribed intervals from which the battle manager can receive. Additionally, it seems reasonable for the battle manager to post target assignments to weapons and that weapons can respond to that post as to whether the assignment is accepted. Finally, it seems reasonable for the non-assigned weapons to receive the posted weapon assignments to determine the engagement status.

We will use a publish-subscribe architectural style for these interfaces to support the decoupling of the battle manager from the weapons. The battle manager receives the posted health and status from a weapon and considers this information along with other factors into the weapon-target pairing calculation. The battle manager posts the weapon assignment and the weapons receive the weapon-target pairing assignment.

The assigned weapon acknowledges the receipt of the assignment, builds a firing solution, and launches the interceptor within the constraints imposed by the battle manager. The other weapons see that the battle manager assigned the target to another weapon and hold fire on that target until instructed to do otherwise by the battle manager.

#### *c. Sensor to Battle Manager*

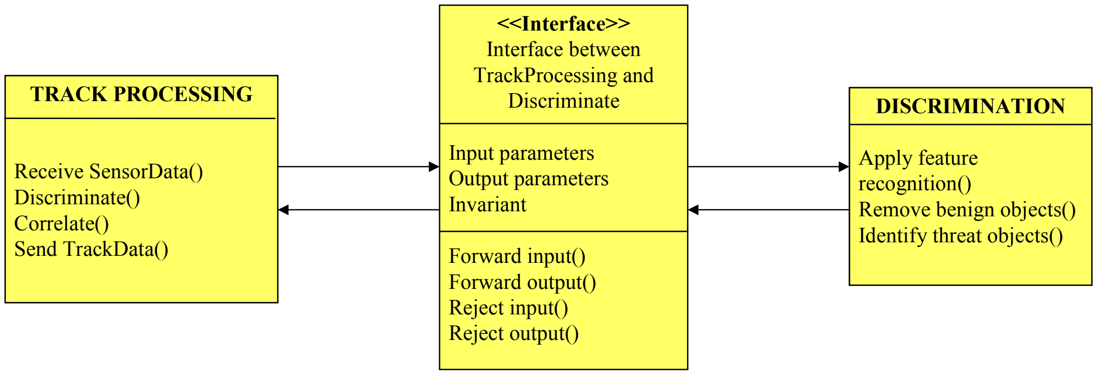
The sensors detect ballistic missile launches and send track data to the battle manager and weapons in the BMDS. The sensors do not require visibility into how that information will be used. Additionally, the success of the sensor mission is not dependent on knowing the location of the consumers of its information. As such, the sensors should be decoupled from the other BMDS subsystems.

As with the C2 to battle-manager interface, we propose the use of the publish-subscribe architectural style for these interfaces to support the decoupling of the sensors from the battle manager. Similar to the C2 to battle-manager construct, a sensor puts an event on the bus by announcing the event. The connector delivers the event to each subscriber of that event. The sensor has no visibility into the consumers of the event data. As such, the sensor is decoupled from the battle manager as well as other consumers of the event data from the sensor. [12]

### 5. BMK Architecture

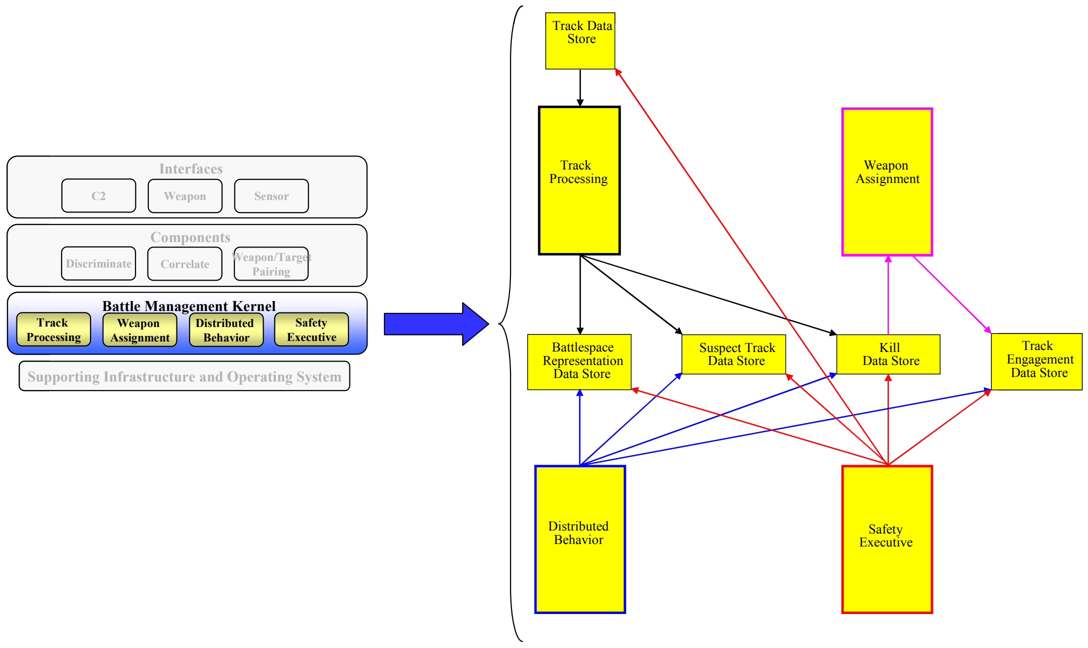
The BMK depends on components in another layer to accomplish computations in track processing, weapon assignment, and distributed processing. In the BMK, the software will call specific components to do work for the kernel software. For example, the track processing component will call upon the discrimination component to discriminate various benign objects from the threat ballistic missile. The track processing component needs to know how to call the discrimination component to do its work. The class diagram for the interface between the Track Processing component and the Discrimination component is depicted in Figure 8.

**Figure 8. Class Diagram**



The architecture for the BMK will include the components that call computational components to do work as well as various data stores that separate the BMK components. We discuss this construct later in this document. The architecture for the BMK is depicted in Figure 9.

**Figure 9. BMK Architecture**



## D. TECHNICAL CONTRIBUTION

For the third technical contribution in this research, we developed architectural views from the system-of-systems view to the component view in the BMK of the BMDS Battle Manager. This demonstrates it is possible to develop system-of-systems architectural views that a developer can use to reason about the system-of-systems as well as the controlling software. This contribution addresses the first of the three research questions from Chapter V.

# X. BATTLE-MANAGEMENT KERNEL

## A. BACKGROUND

Software engineers initially applied the concept of a kernel in the development of operating systems to address the growing problem of increasingly large and unmanageable operating system programs. In 1968, Edsger Dijkstra proposed that systems could be developed as a strict hierarchy of layers. He proposed a five-layer model that featured a progressive layering of abstraction that hid the details of the computer hardware from the software program. In this model, the innermost layer surrounds the computer-system hardware. Dijkstra considered this layer to be the kernel as it contained the only system software that had access to the hardware. This kernel provided services to the other four layers outside of it. [64][17]

In the 1980's, the Unix kernel included a great deal more functions than other kernels that sought to minimize the functions in a kernel. The Unix concept was that of isolating specific functions from the user software by designing the system's hardware to call those specific functions. The result is a monolithic program that contains a significant amount of the system's software. The disadvantage of the large, monolithic approach is that software engineers experience a considerable difficulty in replacing or upgrading hardware components without a complete shutdown or a recompilation of the software. [64][75]

Nancy Leveson proposed the realization of a safety kernel that provides a structuring concept that would support the detection and recovery of safety-critical software faults. Leveson maintained that the detection of a software fault should occur through the application of logic-based assertions in non-kernel software. The recovery from software faults should occur by direction of the kernel to other non-kernel software. The importance of this work was to establish that the software that provided the detection of safety-critical software faults and the recovery from those faults does not have to reside in the kernel; the kernel directs non-kernel software to recover from the safety-critical faults. [64]

John Rushby introduced the concept of independence and separation for a kernel: a safety kernel must be uncoupled from the events, activities, and faults of the software programs from the perspective that an action or fault in non-safety-kernel software must not result in an alteration or fault to the safety-kernel software. [64]

## B. DEFINITION OF A KERNEL

For this research, we define a kernel to be that set of software components that are necessary to provide management of the message transfer among the non-kernel software applications and the computer-system hardware.

## C. BATTLE-MANAGEMENT KERNEL[6]

The concept of a kernel was envisioned decades ago. [64] There is a significant amount of literature on the use of a kernel to monitor and enforce required system safety policies. In [64], Preckshot proposes a definition and rigor to a safety kernel. However, he concludes that his safety kernel is for a restricted set of applications. In [73], Storey discusses the use of a safety kernel for a safety-critical system. However, he foresees the success of a safety-kernel approach as dependent upon the developer to "…protect the kernel from outside influences that might interfere with its operation." Brown suggests in [7] that a safety-kernel could "…significantly enhance the overall safety of the BMDS."

In [10], Caffall and Michael propose extending the concept of a kernel to the battle management of sensors and weapons in the BMDS. They liken the concept of a BMK to that of a safety kernel. There does not appear to be any literature that suggests research for employing the kernel concept in the control of military systems in a system-of-systems environment.

In this research, we propose the use of a BMK that consists of the set of software components that are necessary to provide correct real-time execution of battle-management tasks in a system-of-systems context, both in nominal and degraded modes of system operation. We propose that the BMK must exhibit the following characteristics:

[6] This section includes extracts from [10].

1. The BMK has absolute priority, that is, no other component can interrupt the kernel from accomplishing its work.

2. System parameters and external events are measurable and observable by the BMK. When presented with measurements for a given set of parameters and external events, the BMK will exhibit correct system behavior. (N.B.: We define correct as the reaching of the desired state given the previous state is presented with a given set of inputs.)

3. Detection of errors will be through the use of assertions. The BMK will direct non-kernel software components for the recovery of observed errors such as violation of pre-conditions, post-conditions, and invariants.

A BMK is similar in purpose to an operating system (OS) kernel in that both kernels manage resources shared by competing entities. In the case of an OS kernel, the competing entities are computer processes vying for processor and memory resources. In the case of a BMK, the competing entities are all of the components of the system-of-systems that comprise the battle-management system, such as the C2 and weapon systems.

The active components in the kernel are expected to be stable compared to the other components in the system-of-systems. (N.B.: For this research, we define an active component as one that will execute based on external conditions and a defined set of rules.)

For instance, device drivers tend to be updated frequently and therefore in principle should not be included in the OS kernel. If they are included (i.e., the case of a monolithic kernel), and even worse, tightly coupled to OS management functions, then it becomes challenging to make modifications to the kernel that do not affect other parts of the kernel. We would like to apply this same reasoning to the BMK in order to simplify the design and maintenance of the BMK.

We also draw a parallel between BMK and safety kernels. The functions to be included in a safety kernel are those that must be performed to maintain a safe system state or bring a system back into a safe state after the occurrence of a safety-critical event.

No other functions may be included in a safety kernel. An automated train protection (ATP) system is an example of a safety kernel. [103] Such kernels are well documented, validated, and verified before being considered for certification and accreditation. We view BMKs in a similar light: they must work as advertised because the ability of the entire system-of-systems to conduct warfare in the BMD battlespace is dependent on the BMK.

In our proposed approach, we envision software engineers developing the BMK as a real-time set of system functionality that addresses its use by warfighters, starting from a high-level statement of capabilities and refining these statements into successively lower levels of system artifacts. We define the BMK to be the software that contains the basic functions of battle management that will remain stable over time. Derived from the kill chain [10], these basic battle-management functions are called tasks, and will manage the use of the system's computing resources to ensure that all time-critical, battle-management events are processed as efficiently as possible.

Recall from Chapter IX that we defined and depicted the internal view of the battle manager. In Figure 7, the BMK is the kernel that contains four active components: Track Processing, Weapon Assignment, Distributed Processing, and Safety Executive. In our model of the Battle Manager, the active components of the BMK task the passive components in the Components layer to perform the required computations and return the requested results of the computation to the BMK. (N.B.: For this research, we define a passive component to be one which must be triggered from an external source in order to operate.)

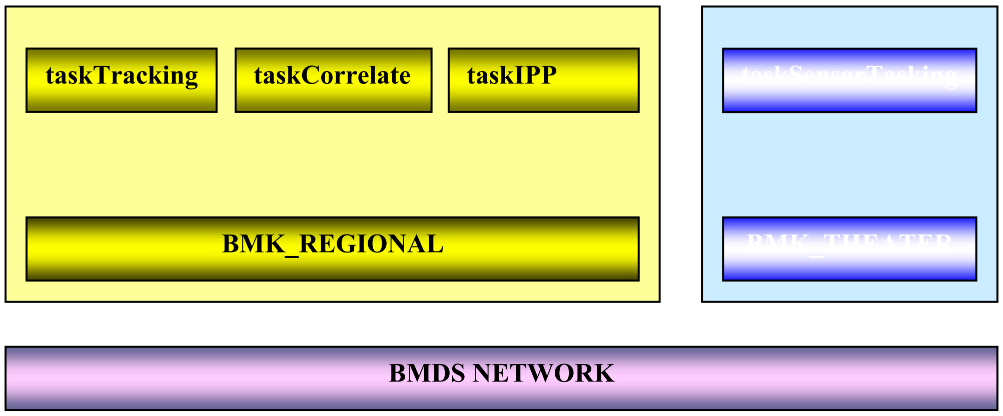
Recall that we proposed that the battle-management framework would feature distributed processing. To achieve the desired property of transparency in the distributed system, we propose that BMK direct the message communications among tasks in the system-of-systems.

We propose that each battle manager in the system-of-systems contain an instance of the distributed BMK. The global BMK would maintain the master copy of the name table while each BMK would maintain a local copy of the name table. As a battle manager joins the battle-management network, its BMK would request a copy of the name table from the global BMK.

As a source task at one battle manager sends a message to a destination task at another battle manager, the local BMK references the name table and determines the location of the destination task. If the destination task is local, the BMK routes the message to the destination task. If the destination task resides at another battle manager, the BMK sends the message to the remote BMK. On receipt of the message, the remote BMK routes the message to the destination task. [41]

These activities are depicted in Figure 10. Consider that the tracking software task in the regional battle manager requires the correlate task to do work. The regional BMK (i.e., BMK_Regional) routes the message from taskTracking to taskCorrelate. This is handled locally by BMK_Regional. Consider that the tracking software task in the regional battle manager requires that a sensor do work. The regional BMK references the name table to determine the location of the specific sensor and routes the sensor-tasking message to the theater BMK (i.e., BMK_Theater) where the sensor is associated. On receipt of the message, BMK_Theater schedules the tasking locally and sends a sensor-tasking message to the sensor to do the required work.

**Figure 10. Messaging Example in a Distributed Battle-Management Network**



In the actual design, each battle manager should have the identical software components and kernel. While the trivial example above illustrates the distributed nature of the BMK, the following example is more akin to the issues facing the software engineers that design the operational battle manager.

Recall that various BMDS systems may be operationally interfaced with two or more battle managers. Consider the example of a sea-based X-band radar that provides mid-course sensing as well as sensor updates to launched interceptors. For the majority of its service the sea-based X-band radar will be in the surveil mode for mid-course sensing. However, for short periods of the fight, a system may require the use of this asset to provide threat-object update information for a launched interceptor. The challenge to software engineers is to design the BMK so that the BMDS can accomplish both missions.

Consider the battle-manager framework in Figure 10. Consider that the sea-based X-band radar has an operational interface to the theater battle manager and the regional battle manager. In normal operations to include execution, the regional battle manager will control the sea-based sensor to include the adjustment of its field of regard. Consider during a ballistic missile fight that the Homeland Defense Battle Manager tasks the ground-based mid-course defense (GMD) system to launch an interceptor against an ICBM. The GMD system requests the sea-based X-band radar to adjust its field of regard to support the threat-object information updates to the interceptor. The global BMK receives this request and references the name table to determine the location of the destination task. The global BMK sends this request to the regional BMK which processes the information. In the Assign Sensor logic, the support to an active engagement is a higher priority than the surveil mode so the regional BMK forwards the GMD system's sensor request to the sea-based X-band radar for execution. After the interceptor "opens its eyes" and assumes onboard tracking responsibilities, the sea-based X-band radar resource is released and the regional BMK directs the radar to resume the surveil mode.

## D. LOGIC IN BMK[7]

We will construct a set of specifications for each active component using assertions and temporal-logic statements that will serve as the functional model of the BMK. The goal is to achieve a greater degree of clarity and focus in the specification of the desired BMK behavior as compared to that obtained from the traditional method of simply listing the system requirements. We will develop a sufficient amount of information to automatically produce test cases for the implementation. Otherwise, we run the risk of developing so-called "cartoon models" that are only useful for drafting and refining potential solutions.

We will develop a slice of the test-ready model for the BMK. According to Binder [3], in order to be testable, a model should contain all the features of the system-under test (in the present context, the BMK), preserve sufficient detail that is critical for discovering faults, and faithfully represent the essential states, actions, and transitions in the state diagram. If the BMK model is to be useful for this effort and in future development efforts, it might exhibit the following properties outlined in [66]: appropriate level of abstraction, high degree of understandability, high measure of accuracy, and high level of predictiveness.

We will use temporal-logic assertions to define the temporal aspects of the BMK specifications. We anticipate that these assertions will yield specifications that are verifiably consistent and accurate, and in turn, verifiably predictable behavior of the BMK.

As an example of such logic in the BMK, let us consider the situation for which the BMK has determined that an observed object is a threat and must now assign a weapon to engage that threat track. Clearly, we want to establish a time constraint by which the BMK has assigned a weapon to engage a threat track. This is a situation for which a late computation is an incorrect computation so we should establish a time constraint to ensure that the BMK has completed its weapon assignment work as desired.

[7] This section includes extracts from [10].

For this example, we will assume that the BMK must assign a weapon to engage a threat track within thirty seconds of the BMK determining that the tracked object was indeed a threat.

We could state the natural language assertion as follows:

> *Within thirty seconds of determining a given track is a threat object, the BMK will assign a weapon system to engage the threat object.*

We could write the temporal assertion as follows:

*Boolean: Ballistic_Threat*
*// Tracked object is ballistic-missile threat is true*

*Boolean: Weapon_Assigned*
*// Weapon assigned to tracked object is true*

*Always (Ballistic_Threat) Implies Eventually*$_{timer<30}$ *(Weapon_Assigned)*

# XI. BATTLE-MANAGER COMPONENTS

## A. BATTLE-MANAGEMENT FRAMEWORK

Recall from Chapter XI that we proposed the concept of component-based engineering to design and develop the BMDS Battle Manager. We proposed that we develop the software that contains the basic functions of battle management as components in the BMK, given that this software should experience limited modifications over time. Derived from the kill chain [10], these basic battle-management functions are called tasks, and will manage the use of the system's computing resources to ensure that all time-critical, battle-management events are processed as efficiently as possible. We proposed to develop other component software that contains the algorithms required to perform the computations of the BMDS Battle Manager.

Finally, we proposed to add another software application to the kernel that controls the distributed processing in the battle manager. While we anticipate that the battle-management functions will execute on a single, multi-processor platform, the distributed functions to include control sensor-resource management, engagement control, sensor tasking, and survivability will exhibit different behavior than the kill-chain functions. As the kill chain functions must execute regardless of the status of the distributed system, we do not desire to mix the distributed functions with the kill-chain functions.

Due to continual research and development in many areas of missile defense, we anticipate that researchers will discover improved algorithms that will replace currently realized algorithms in the components of the battle manager. In this chapter, we will explore various hypotheses of components and component engineering, and offer a recommended course of action for using the concept of a component in the development of a controller for system-of-systems software.

## B. DEFINITION OF COMPONENT

For this research, we will adopt Szyperski's definition in [74]: "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party." This definition implies that one must precisely and clearly specify the interfaces and ensure that the component software is encapsulated and can be accessed by software external to the component only through its interface. [15]

## C. BATTLE MANAGER CONSIDERATIONS

Why should we consider components for the BMD Battle Manager?

We will frequently modify and update the operational software for the BMD Battle Manager. Given that one requirement for the BMD Battle Manager is 24x7x365 operations, the warfighters cannot tolerate a system that is non-operational due to uploading new software or for debugging newly uploaded software. As such, we must develop a solution in which we do not interrupt operations and in which we can recover to the last known operational state if the upload is not successful.

From operational perspective, the employment of the BMK and associated components will support the requirement for no downtime due to uploading new software. Rather than downloading the old software and uploading the new software in a monolithic software program, we can change out the modified components while leaving the remainder of the system intact.

From the software development perspective, the employment of the BMK and associated components might offer four potential advantages over a large, complex monolithic software program [1]:

1. *Independent Extensions*. The component model and framework outline the methods of extending the capability beyond the original design. By adherence to the interface specifications, the extensions to a component can be designed, developed, and fielded without undesired interactions.

2. *Component Markets*. The component model and framework can significantly reduce system complexity as compared to a large monolithic software program. The component model and the framework define the standards to ensure that independently developed components can be employed in the system without unintended interactions. If one integrates the support services into the framework, then the

development of components can be simplified. The component model and framework allow developers to seek third-party vendors to develop specific components.

3. *Reduced Time-to-Fielding.* The component model and framework promote concurrent code development. Given that the architectural decisions are set to include the definition of the component interfaces, the development team can develop the components concurrently or use existing components in the component product line. Additionally, the component model and framework allow for the incremental delivery of a system which supports the ability to get a system into operational use quickly.

4. *Improved Predictability.* The designers specify the design rules for the component model so that these rules are consistently imposed on all components in the system. The consistency of the imposed design rules will ensure that global properties such as security, safety, and recovery are designed into the system.

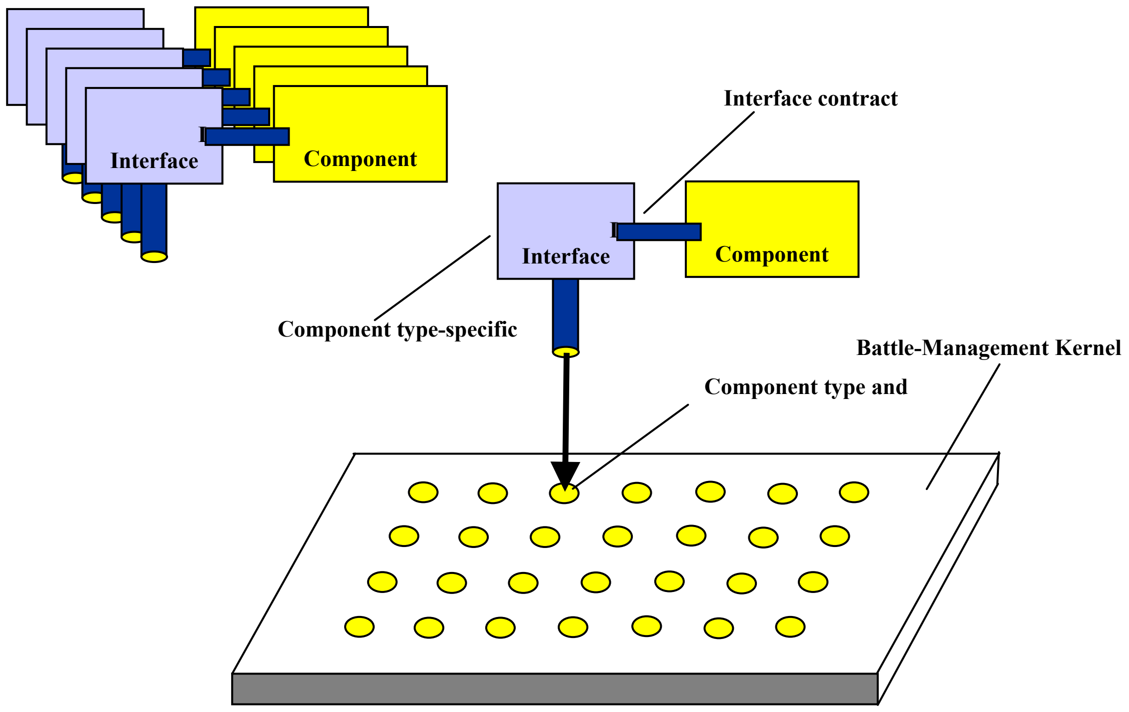
## D. COMPONENT ENGINEERING CONSIDERATIONS

### 1. Component-Based Software Engineering

Component-based software engineering is the design and construction of a system by integrating software components, interfaces, contracts, and a component framework. The concept is to employ portions of legacy software that contain the desired functionality as expressed in the software architecture, shape the legacy software into a "unit of composition with contractually specified interfaces [74]," develop new components as required, and integrate the components into a component framework to realize the desired system functionality.

The logical view of components, interfaces, contracts, and frameworks is depicted below as the component-based design pattern in Figure 11:

**Figure 11. Component-Based Design Pattern**



### 2. Component Framework

The foundation for developing and designing a system using component-based software engineering is the component framework. It is the "glue" that binds the components to do the desired work that is specified in the architecture. In the battle manager, the kernel will contain the component framework which is comparable to an electronic circuit board with vacant slots into which components can be inserted to achieve a desired capability. [15] That is, the component framework depicts where a component resides and its interface to other software.

The primary role of a component framework is that it compels components to do work in accordance to the methods controlled by the framework such as the inclusion of temporal invariants to enforce system time constraints. Additionally, the component framework specifies the context into which one can integrate the various components to compose a system. [15]

To increase the capability of a system, we will add other components to the framework that provides additional functionality. To increase the precision of a system, we could replace the current components with new components that contain more precise algorithms. Note that new components must meet the contract of the component interface as it is a distinct, configurable entity that is modified only through established software maintenance procedures.

### 3. Component Properties

In developing the software architectural views with respect to components, we might consider the following properties of components as outlined in [15], [38], and [74]:

a. Provide services through well-specified interfaces.

b. Encapsulate state and behavior so that neither is visible to the component framework.

c. Rely on the component framework to initialize and communicate with other components.

### 4. Component Interfaces

We propose to employ the concepts of design by contract in the specification and design of the component interfaces. For this research, we define a component interface as the specification of the access to the software component by the component framework. [15]

Design by contract is a formalized way of writing comments to incorporate specification information into the software to express the requirements for the component. That is, the contracts describe the assumptions the developer made when writing the code, and the assumptions that the system can make about a piece of code. The concept of design by contract is that software entities have obligations to other entities based upon formalized rules between them. We create a functional specification (i.e., contract) for each component in the system whether it is salvaged code from legacy software or developed as new code. Thus, the execution of the software is the interaction between the component framework and the various components as bound by these contracts. [15][74]

For example, if a developer assumes that a given variable will never receive a null or negative input, then the developer should include this information in the contract. Additionally, if a developer writes a piece of code that is always supposed to return a value greater than 100, he should add this information in a contract so the developers working with the other parts of the application know what to expect. These contracts describe requirements such as:

a. Conditions that must be satisfied before a method is invoked

b. Results that need to be produced after a method executes

c. Assertions that a method must satisfy at specific points of its execution

A contract can specify three types of constraints in the interface. A contract specifies the constraints that the component will preserve – an invariant. (N.B.: We define an invariant as a condition that does not change in the presence of system or environmental transformations.) The contract specifies the constraints upon the component framework (i.e., a pre-condition). Additionally, the contact specifies the constraints upon a component with respect to what it returns to the component framework with respect to the input to the component operation (i.e., a post-condition).

### 5. Specifications

In the specification of the component framework, components, and interfaces, we recommend that the control and coordination features be specified in the component framework and the method of computation be specified in the component. We recommend specifying the interface as a separate entity than the component framework and the component. We want to specify the contract between the interface and the component framework in the interface. Thus, the interface is specified as an independent entity rather than a portion of either the component framework or the component.

In the specification of the component framework, one must specify more than the desired functionality of the component. Included in the specification should be: (1) the required response time of a component's computation that supports the desired behavior of the system, (2) the required precision of the result to ensure that matching of precision between the component framework and a component, (3) the required throughput of the data streams to ensure that data loss does not occur as a result of a throughput mismatch, (4) required protocol and formatting to ensure the matching of data and fields during data transfer, (5) legal values for inputs and outputs, and (6) data dictionary to include the specification of the units of measure in the component framework. [12][15][74]

In the specification of the component, one must specify more than the required input parameter from the component framework. Included in the specification should be: (1) time to complete a computation, (2) the required precision of the input to ensure that matching of precision between the component framework and a component, (3) memory requirements to ensure sufficient memory is designed into the software architecture, and (4) data dictionary to include the specification of the units of measure in the component framework. [12][15][74]

## E. CHALLENGES

Although the benefits of component-based software engineering are widely touted in numerous publications, challenges exist that should be understood. We should consider mitigation strategies for these challenges to enhance the probability of realizing the benefits of component-based software engineering. These challenges include the following:

### 1. Modeling

A single, chosen architecture method does not exist that defines the relationships among component frameworks, components, and component interfaces. Although such tools as the Unified Modeling Language provide various options for documenting the relationships, a clear roadmap is not available for one to model the use of components in the design of a controller for a system-of-systems.

### 2. Specifications

Frequently, the specifications for a system-of-systems are focused solely on functionality. As such, the specification of system properties that can increase the level of trustworthiness may be ignored.

### 3. Trusted Components

One potential benefit of component-based software engineering is the ability to upgrade or replace an existing component with ease. The flip side of this benefit is that modified or new components can precipitate failures in the system by producing undesired side effects or feature conflicts regardless of whether the other components of the system remain unchanged.

### 4. Component Reuse Versus Component Salvaging

Another potential benefit of component-based software engineering is the ability to replace existing software components with components from another system. This capability is frequently referred to as software reuse; however, it is more often than not a situation of software salvage. (N.B.: For this research, we define software reuse as the act of selecting and employing a chunk of software that was designed and implemented for use in other systems without modification to that chunk of software. We define software salvage as the act of selecting a chunk of software and modifying it for use in another system.)

With respect to components, both software reuse and software salvage bring about the question of trusted components as previously described which is that new components can precipitate failures in the system by producing undesired side effects or feature conflicts regardless of whether the other components of the system remain unchanged. For the situation of software salvage, this negative impact is likely to increase as the degree of modification increases.

## F. BATTLE MANAGER

To address the previously discussed challenges, we offer the following recommendations in the application of component-based software engineering approach to the system-of-systems problem. While the offered solutions are not the only solutions, these solutions can improve the probability of a successful development of a controller for a system-of-systems.

### 1. Modeling

To move beyond the simple structural model of sticks and circles to represent a system-of-systems, we will consider a modeling technique that captures information on the desired behavior of the system as well as the desired behavior of each element in the system with respect to its contribution towards the desired behavior of the system. With respect to components, we should understand the behavior and constraints of the component framework, each component, and each interface between the component framework and a component.

Recall that we chose to model the BMDS as a reactive system. In our model, we considered the Battle Manager to be the Controller as depicted in Figure 5. That is, the Sensor will detect external signals (i.e., external stimuli) and process this information to send to the Controller. The Controller will make decisions based upon the input from the Sensor and send control data to the Actuator for execution of tasks. This results in a response from the reactive system.

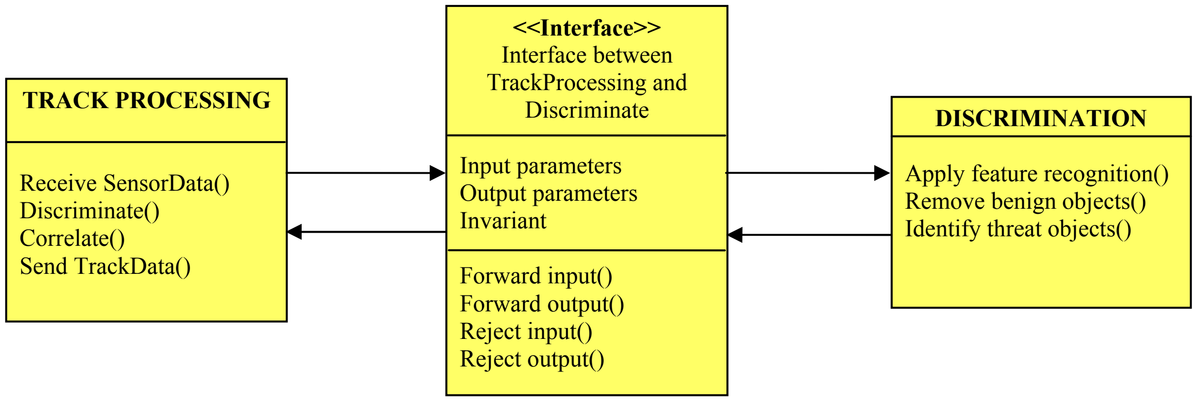
Furthermore, we proposed the internal view of the BMDS Battle Manager to be a layered construct of interfaces, components, and a kernel as depicted in Figure 7. Recall that we defined the BMK as a composite of active components and we proposed that the components that perform computations in the Battle Manager be passive components. Thus, in our model of the Battle Manager, the active components of the kernel task the passive components to perform the required computations and return the requested results of the computation to the kernel.

For the Battle Manager, we desire that the state and behavior of the active components in the kernel not be visible to the passive components. Ideally, we would prefer that the passive components be pure computations without state and behavior; however, if a passive component cannot meet this ideal goal, then the state and behavior of the passive component should not be visible to the active components in the kernel; that is, we do not desire that the state and behavior in the active components inadvertently trigger state transitions in the passive components that result in undesired behavior in a passive component. Equally, we do not desire that the state and behavior in the passive components inadvertently trigger state transitions in the active components that result in undesired behavior in an active component.

Now, we will consider the interface between an active component in the kernel and a passive component. As one means of increasing the level of trustworthiness in the Battle Manager, one might define the desired services, behavior, and constraints in the interface between an active component and a passive component. In interface, we will find that it receives data (i.e., input) from the active component that is intended for the passive component and provides data (i.e., output) from the passive component that is intended to return to the active component.

We model this relationship for the active component Track Processing and the passive component Discrimination as depicted in Figure 12.

**Figure 12. Track Processing Component Interface to Discrimination Component**



## 2. Specifications

To specify the behavior and constraints in the interface, we suggest the use of assertions to check the input and output parameters as well as to establish any invariants required for the interface. For the input parameters, we could assert the pre-conditions that are required to process data that lies within the set of legal values. For the output parameters, we could assert the post-conditions that are required to return the results of the passive component's computation as related to the input to the active component. For the invariant, we could assert any property which must hold true regardless of any and all computations.

The use of assertions in the pre-conditions, post-conditions, and invariants can enhance the safety properties of a system but verifying input parameters. By allowing only legal inputs to be processed, the probability of an errant computation on illegal data is reduced. By allowing only legal values of the output to be returned, the post-condition assertion reduces the probability that an incorrect result will be passed to the active component. By requiring a computation to terminate within a specified timeframe, the invariant ensures that the system will not halt while waiting for a never-ending computation to terminate.

As a simple example of applying assertions to implement the behavior and constraints of an interface, we will use the class diagram depcited in Figure 12. Consider that we want to define two input variables (X,Y) for discrimination in terms of range of value. Consider that we want to define the output to include (ThreatObject) as a Boolean statement that has a relationship to the input variables (X,Y) and the type of threat object based upon the computation in the passive component. Consider that we want the computation to terminate no more than ten seconds after passing the input variables to the component. We suggest the following assertions reflect these considerations:

Pre-condition

*Always (X,Y) Implies> {(X≥0 and X≤10) and (Y≥100 and Y≤1000)}*

//(X,Y) will be valid only if:

X is equal to or greater than 0 and X is equal to or less than 10

AND

Y is equal to or greater than 100 and Y is equal to or less than 1000

Post-condition

*Always ThreatObject Implies {(X>5 and Y< 500) or (X< 10 and Y≥500)}*

//ThreatObject will be true only if:

X is greater than 5 and Y is less than 500

OR

X is less than 10 and Y greater than or equal to 500

Invariant

*Always (X,Y)Implies Eventually*$_{t<10}$ *(Terminated)*

//The condition of the Boolean value "Terminated" will be true sometime in the future but less than 10 seconds have expired following the passing of valid and legal X,Y input variables to the passive component. (N.B.: The Boolean value "Terminated" must be set to not true each time new variables are sent to the passive component.)

### 3. Trusted Components

One method that can increase the degree of trustworthiness of a component and the Battle Manager is the development of a certified test suite for each component as well as a certified test suite for the Battle Manager. [3][15] (N.B.: For the purposes of this research, we define a certified test suite as one in which the results of the testing are known, verified, and certified with respect to the test inputs to a system under test. For the purposes of this research, we define certified as the guarantee that a system or component will operate correctly and will operate correctly in adverse conditions.)

If assertions were employed in the development of the component interfaces, then the test suite should include test cases that examine the inclusive range of values for the assertion, the boundary values, and illegal values. More than checking whether the assertion holds, one should investigate the error-handling procedures associated with a logic break in an assertion. It is imperative to determine that a system will not experience a fail-hard condition for a break in the logic of an assertion.

The test suite should include the inspection of the (1) the time to complete a component's computation to determine whether the component supports the desired behavior of the system, (2) the precision of the result to ensure that matching of precision between the component framework and a component, (3) the throughput of the data streams to ensure that data loss does not occur as a result of a throughput mismatch, (4) protocol and formatting to ensure the matching of data and fields during data transfer, (5) legal values for inputs and outputs, and (6) data dictionary to include the specification of the units of measure to ensure consistency of the received and returned data with respect to units of measure.

The certification of trustworthy components should include the verification of functionality, degree of fault tolerance, level of compliance with the interface contract, speed of service, throughput, time of computation, and degree of consistency with the data dictionary of the component framework.

### 4. Component Reuse Versus Component Salvaging

In the situation of component reuse, the black-box testing and component certification as previously described can reveal issues with the correctness, robustness, and reliability of the component. (N.B.: For this research, we define black-box testing as a software testing technique whereby explicit knowledge of the internal workings of the component being tested are not known and the outputs are examined with respect to the inputs.)

In the situation of component salvage, it may be insufficient to employ solely black-box testing and component certification. We recommend the addition of white-box testing techniques to the test suite for the situation of component salvage. (N.B.: For this research, we define white-box testing as a software testing technique whereby explicit knowledge of the internal workings of the component being tested is used to examine the outputs.) That is, it may be necessary to test such items as the algorithms employed in the component to ensure that the precision in the computation meets the precision requirement of the system, exception handling to ensure that software faults are caught and handled appropriately, degree of encapsulation of class methods from interface, presence and implementation of multiple inheritance, and termination of computation.

THIS PAGE INTENTIONALLY LEFT BLANK

# XII. SPECIFICATION OF THE BATTLE MANAGER

## A. BACKGROUND

Software is becoming increasingly more critical and complex in system-of-systems development. Since acquisition life-cycles, failure models, and verification methods that have performed satisfactorily for hardware systems are not always optimal for systems that include a significant software component, the identification and evaluation of better specification and verification techniques for system-of-systems is a never-ending search in defense acquisition.

One tool for specifying and implementing the desired system behavior is formal methods. (N.B.: We define a formal method as one that precisely describes a specification in mathematical terms to make possible the verification of the specification in the requirements phase as well as the testing phase of system development.) One can use formal methods in the definition and verification of system specifications. Additionally, one can implement the formal specifications with formalisms in the software. (N.B.: For this research, we define a formal specification as the precise definition of a system behavior that is typically expressed in mathematical terms.)

The application of formal methods for specification and verification is a technique for consideration by developers of system-of-systems. Formal methods can complement traditional techniques such as testing and can help developers improve the degree of trustworthiness in defense acquisitions.

## B. SPECIFICATION PROBLEM

The less-than-sterling success of software development in the United States is well documented in [49], [69], and [70]. While the actual statistics and cited numbers might be challenged by skeptics, the information seems to tell us that we have a serious problem in developing software that provides the required functionality of the user in a timely fashion and within budget.

If we are to develop dependable software-applications for a system-of-systems, then it would seem that we need to address the specification issue. (N.B.: For this research, we define a specification as the description of a desired system behavior that is expressed as a feature, function, property, or capability.)

Recall specific examples of specification issues from Chapter III:

- *In 1991, the PATRIOT system failed to intercept a Scud missile which resulted in the deaths of twenty-eight American soldiers.*
- *In December of 2001, a 2000-pound, Joint Direct Attach Munitions (JDAM) bomb killed three U.S. Special Forces airmen and five Afghan soldiers, and wounded nineteen other military personnel. The root cause of this friendly-fire incident was the inadvertent passing of the coordinates of the US air controller's own position to the bomber.*
- *From a study of 387 software errors discovered during the integration and testing phase of the Voyager and Galileo spacecraft, Robyn Lutz observed that the safety-related, functional faults Voyager could be categorized as follows: 50% as behavioral faults, 31% as conditional faults, and 19% as operating faults. For Galileo, the safety-related, functional faults could be categorized as follows: 38% as behavioral faults, 18% as conditional faults, and 44% as operating faults.*
- *Delores Wallace and Richard Kuhn analyzed software faults from 342 medical systems and determined that 43% of the software faults were logic-related errors such as incorrect logic in the systems' specifications, unexpected behavior of multiple conditions occurring simultaneously, and improper limits.*

## C. FORMAL SPECIFICATION OF THE BATTLE MANAGER

Conventional software development methods may not be suitable for the development of safety-critical systems. [79] In safety-critical systems, system faults could prove fatal to human life or lead to loss of valuable physical assets. [34][73] With the use of formal methods, developers can analyze formalized statements and the associated impacts in a repeatable manner. Formal methods help one test a significant number of test cases and support analysis that can be checked by verified model checkers. In operational software, the use of formal methods can significantly enhance the ability to catch and handle runtime errors.

### 1. Formal Specifications

Traditional specification engineering uses natural language statements to describe the desired system behavior. Typically, the specifications are inaccurate, inconsistent, and ambiguous. [79][49] Graphical representations of specifications (e.g., UML) typically have limited precision in semantics and can lead to an ambiguous interpretation of requirements.

Indeed, Dean Leffingwell and Don Widrig make the following four statements in [50] that should give us cause to pause, and understand the relationship of seemingly insufficient specification elicitation impact of and why acquisition organizations cannot develop good software:

*Specification errors are likely to be the most common class of error in software developments.*

*Specification errors are likely to be the most expensive errors to correct after fielding.*

*Specification errors will contribute up to 70% of all software rework costs.*

*Specification errors can consume 25%-40% of the total program budget.*

Testers can develop test cases from natural language specifications and graphical representations of the system; however, they will need to interpret the natural language specifications and graphical representations which are typically limited in content and consistency. [3] So, it would seem wise for acquisition organizations to adopt techniques that could lead to a significant reduction in specification errors.

For this research, we develop a functional model formed by the specifications are verifiable and can produce a test-ready model as described in [3]. (N.B.: For this research, we define a test-ready model as one that contains sufficient information for which to automatically produce test cases for its implementation.) As outlined in [3], a test-ready model should meet the following requirements:

*The model should be a complete and accurate representation of the implementation to include all features, functions, properties, and capabilities of the system.*

*The model should preserve the level of details that is essential for testing fault tolerance while abstracting out unnecessary detail.*

*The model should represent all states, guard conditions, actions, and triggers in the system state model.*

As one develops the test-ready model, the issue of verification of the model arises. That is, developers should ensure that the evolving test-ready model of the system is progressing towards faithfully realizing the system specifications. Given that the expression of requirements is typically in the form of natural language, developers cannot easily verify system requirements. System engineers oftentimes employ traceability techniques that focus upon tracing system requirements to user requirements. Besides being tedious, the constraints of natural language can lead to misinterpretations, inability to detect incomplete or conflicting logic statements, and limited verification of robustness. Unfortunately, traditional system developments rely on unit testing and integration testing to verify whether the system meets its requirements.

The development of formal specifications in the test-ready model can lead to a significantly high level of confidence in the implementation phase of a software development. The development of formal specifications typically clarifies the specification, surfaces latent errors and ambiguities, and supports the shaping of the desired system behaviors. [42]

For the battle manager, we propose the use of assertions in the development of formal specifications. (N.B.: For this research, we define an assertion a predicate expression whose value is either true or false.) We propose that developers develop assertions to define pre-conditions, post-conditions, and invariants. (N.B.: For this research, we define a pre-condition as a fact that must always be true just prior to execution of a specific section of code. Furthermore, we define a post-condition as a fact that must always be true just after the execution of a specific section of code. Finally, we define an invariant as a property that holds true under any transformation in the system.)

Assertions can help us find defects in specifications and designs earlier than they would be otherwise and greatly reduce the incidence of mistakes in interpreting and implementing correct requirements and designs. Additionally, the development and verification of formal specifications can support the development of error-handling specifications to appropriately manage runtime errors and logic breaks.

The use of assertions can significantly reduce the errors introduced in the specifying system behavior. [3] Assertions can considerably increase the level of clarity in the assumptions and responsibilities of system behavior, and reveal errors such as logic omissions and conflicting logic-statements. Assertions can catch common interface faults (e.g., processing out-of-range or illegal inputs) by precisely asserting the legal interface values for variables passed in through an interface.

An example of using an assertion to specify a pre-condition is as follows:

```
Variables:
        //Track data Input to interface contains two variables: Input_A
        and Input_B
        Integer: Input_A
        Real: Input_B
        Boolean: Detect_Track
        //True if sensor has provided data to battle manager for a
        reportable object
        Boolean: Track_Data
        // True only if Detect_Track is true and valid track data exists for
        observed object

Assertion:
        //Restrict Input_A to be greater than or equal to zero, and less than
        100 and Restrict Input_B to be less than 1000
        //Confirm adherence to pre-conditions
        read (Input_A,Input_B)

        assert: Always Track_Data Implies (Detect_Track) and
        {(Input_A ≥ 0 and Input_A < 100) and (Input_B < 1000)}
```

An example of using an assertion to specify a post-condition is as follows:

***Variable:***

*Real: Output_B*
*Boolean: Track_Data*
*// True if Detect_Track is true and valid track data exists for observed object*

***Assertion:***

*//Confirm that Output_B is not empty*
*read (Output_B)*

*assert: Always (Track_Data) Implies (Output_B ≠ null)*

An example of using an assertion to specify an invariant is as follows:

***Variables:***

*Boolean: Detect_Track*
*//True if sensor has provided data to battle manager for a reportable object*
*Boolean: Track_Data*
*// True if Detect_Track is true and valid track data exists for observed object*
*Set: Track_Dataset*
*//Contains observed characteristics from sensor*
*Boolean: Threat_Object*
*// True only if tracked object is a ballistic missile threat*
*Boolean: Benign_Object*
*//True only if tracked object is not a ballistic missile threat*

***Assertion:***

*//Confirm data processing of track data within twenty seconds of detecting object*
*Read (Detect_Track,Tracked_Data)*

*assert: Always (Detect_Track and Track_Data) Implies*
*Eventually*$_{t<20}$ *(Threat_Object or Benign_Object)*

## 2. Model checking[8]

Software developers should consider verifying the functional specifications via a tool such as model checking. (N.B.: For this research, we define model checking as the systematic approach for testing functional assertions and substantiating the desired system behavior in the model.) Model checking is not a proof of correctness; instead, model checking involves creating functional models of a system and analyzing the model

[8] This section was extracted from [10]

against the formal representations of the desired behavior [51]. For the battle manager, we propose to verify the functional specifications using an automated model-checking tool that can accept either developed specifications or UML statecharts as discussed in [40], and exercise the temporal-logic assertions over a number of time cycles. Such an approach can support the identification of inconsistencies and breaks in logic through the use of the model-checking tool. From the results of the model checking, developers can correct our specifications and the artifacts from the domain analysis as required.

However, the use of model checking is constrained by the state explosion problem as the size of the state space exceeds the memory capacity of the automated tool to check every trace in the model. [16] Through abstraction of the battle-manager functions in our specifications, we can employ the concept of symbolic model checking in which Boolean functions are employed to represent transition relations and sets of states, using, for instance, a compact representation of the state space (e.g., binary decision diagrams [11]), to simplify the battle-manager states by removing sub-trees and redundant edges on the battle manager's Boolean decision tree. In other words, we can transform the complex logic decisions at the bottom of the tree into simple Boolean statements so that we can capture the essence of the system behavior in the upper portions of the decision tree. By reducing the high number of lower-level logic statements that develop very specific solutions and have limited impact on the overall system behavior, we should be able to manage the state-explosion problem.

As an example of the state-explosion problem in terms of the BMK, consider the following assertion:

*Always Intercept_Point_Min_Within_Intercept_Range* Implies
(*Min_Intercept_Point* is contained within *Interceptor_Range_Volume*)

Note that the number of points in *Interceptor_Range_Volume* could be large and that we are seeking to ensure that one specific point (*Min_Intercept_Point*) is within the set of points that define *Interceptor_Range_Volume*. Rather than use model checking to ensure that this condition is true, we could abstract the assertion to either a True or False for *Intercept_Point_Min_Within_Intercept_Range*. This will reduce the number of traces through the model to verify this assertion.

Model checking is not within the scope of this research; however, we will recommend the application of model checkers as a potential research topic in the development of dependable systems.

**3. Testing**

The Standish Group reported a dismal report on the state of software development in 1994. According to [70], software developers in the United States produced successful software in only 16.2% of the development efforts. The following is an extract from [70]:

> *The Standish Group research shows a staggering 31.1% of projects will be cancelled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates. The cost of these failures and overruns are just the tip of the proverbial iceberg. The lost opportunity costs are not measurable, but could easily be in the trillions of dollars.*
>
> *Based on this research, The Standish Group estimates that in 1995 American companies and government agencies will spend $81 billion for cancelled software projects. These same organizations will pay an additional $59 billion for software projects that will be completed, but will exceed their original time estimates. Risk is always a factor when pushing the technology envelope, but many of these projects were as mundane as a driver's license database, a new accounting package, or an order entry system.*
>
> *On the success side, the average is only 16.2% for software projects that are completed on-time and on-budget. In the larger companies, the news is even worse: only 9% of their projects come in on-time and on-budget. And, even when these projects are completed, many are no more than a mere shadow of their original specification*

*requirements. Projects completed by the largest American companies have only approximately 42% of the originally-proposed features and functions.*

In 1999, the Standish Group updated their research from the 1994 study. The report offered that the success rate of software developments had increased to a 26% success rate from the dismal 16.2% success rate in 1994. The research revealed that project size seemed to be a factor in predicting success as projects that cost over $10 million had a 0% chance of success while projects costing less than $750 thousand had a 55% chance of success. The following is an extract from [69]:

> *Company size does not guarantee success. The Standish group has found no correlation between a company's size and its project success rate. As with project size, bigger is not necessarily better. While large companies (over $500 million) do experience more failures and fewer successes than medium companies ($200 million to $500 million), project failure rates are generally distributed quite uniformly across companies of all sizes. Project failure is everyone's problem.*
>
> *Another way to look at project resolution is to compare the value of successful projects with the waste of challenged and failed ones. Along with improvements in time and costs overruns, companies' waste- to-value ratios have improved substantially. In 1996, CHAOS research fond 50% waste in IT projects. By 1998 the data identified only 37% waste.*

In 2003, the Standish Group released their latest findings which listed the success rate for software projects was 34% in 2003 as compared to 16.2% in the 1994 report and 26% in the 1999 report. [71] Furthermore, the report claimed that the failure rate of software projects was 15% in 2003 as compared to 31% in the 1994 report and 28% in the 1999 report. However, the report revealed that time overruns had significantly increased to 82% as compared to 63% in the year 2000. Additionally, the report claimed that software developers fielded only 52% of the required features and functions as compared to 67% in the year 2000.

With respect to the limited success of defense software development, Leishman and Cook offered the following observation which is an extract from their article in the April 2002 issue of CrossTalk [49]:

> *At the 5th Annual Joint Aerospace Weapons Systems Support, Sensors, and Simulation Symposium in 1999, the results of a study of 1995 Department of Defense (DoD) software spending were presented.*
>
> *As indicated, of $35.7 billion spent by the DoD for software, only 2 percent of the software was able to be used as delivered. The vast majority, 75 percent, of the software was either never used or was cancelled prior to delivery. The remaining 23 percent of the software was used following modification.*

Recall that Leffingwell and Widrig claimed in [50] that specification errors will be the source of seventy percent of the system rework costs. Furthermore, they state that given that rework costs are typically 30% to 50% of a program budget, the correction of system-specification errors can cost 25% to 40% of an entire program budget. Software bugs cost the United States economy $59.5 billion annually according to a 2002 report by the National Institute of Standards and Technology. [19] Additionally, this report claims that software developers expend approximately eighty percent of development costs towards identifying and correcting discovered bugs.

The use of assertions in specifying a system can result in more detected defects than traditional testing and can provide a higher degree of dependability than systems that have undergone traditional testing without supporting formal methods. Due to resource constraints in development efforts, one cannot exhaustively test the entire test suite that is required for complete test coverage.

As an example of the futility of attempting to realize complete, exhaustive testing for a system, let us consider the problem of developing a test for a program that reads three integers that represent the lengths of the sides of a triangle. The output of the program is a statement that identifies the triangle as isosceles, equilateral, or scalene. If we limit the points on an x,y axis to be integers between one and ten, there are $10^4$

possible ways to draw a line. If we test three lines at a time, then we have $10^{12}$ possible inputs for the three lines to include all invalid combinations. Consider that we automate the test procedures for this problem to the extent that we can continuously execute a thousand tests every second. Under these conditions, the automated tester would require about 31.7 years to test every possible input combination if the automated tester ran twenty-four hours each day for every calendar day in each year of the continual automated testing. [3]

Exhaustive testing may be possible for the most trivial of systems; however, the complex systems in the Department of Defense are significantly more complex than the triangle problem. Consequently, defense software testers typically execute a finite number of test cases that test a portion of the system software. Unfortunately, finding a runtime software fault in large, complex system can be difficult. Oftentimes, the location of the observed fault is not the location of the true defect. Therefore, as the number of instructions and computations increase between the observed fault and the true defect, the software tester can experience increased difficulty in identifying the true defect in the software.

The risk of drawing conclusions about a system behavior by extrapolating the results from a finite number of tests can be reduced by using formal methods. Additionally, the use of formal methods can support the software tester in identifying inappropriate system behavior (i.e., observed faults) and locating the defects in the code. That is, the testing with assertions in the software can identify software faults earlier than software without assertions and the testing with assertions can lead to the software testers closer to the defect as compared to testing software without assertions.

The use of assertions can support the design and implementation of built-in testing of the software. [3] Such built-in tests can check implementation details such as assumptions and constraints. Additionally, the use of assertions provides a mechanism to test the trapping of software faults and the runtime handling of the trapped software faults. For example, assertions can support the tester in determining that legal and valid inputs are processed as designed. On the other hand, assertions can trap unanticipated inputs that may be illegal or invalid as defined by the assertion.

In component-based software engineering, the use of assertions can provide an instrument for testing interface contract-violations. These violations could include inappropriate calling of a component by the component framework or the called component fails to deliver its computational results. [68]

In the test development of software with assertions, testers should understand that an assertion will hold true for all legal and valid inputs. [3] That is, an assertion will not pop for legal and valid inputs. Thus, assertion errors can be hidden from testers who assume that the absence of popped assertions equates to the software exhibiting the desired behavior. Therefore, testers should include inputs that cause the assertion to pop.

Assertions can enable testers to develop test cases to check the behavior at domain limits and boundary values. [3] Oftentimes, asserting pre-conditions will suffice to define domain limits and boundary values. For example, consider the assertion previously developed for a pre-condition:

*Always Track_Data Implies (Detect_Track) and*
*{(Input_A $\geq$ 0 and Input_A < 100) and (Input_B < 1000)}*

Testers can use the assertions to develop a test oracle. (N.B.: For this research, we define an oracle to be a tool to evaluate the results of a test case as either pass or not passed. The oracle is the test key that contains the inputs for a system and the associated required output for each input.)

In our test case for the above assertion, we might develop procedures that test the following combinations of Input_A and Input_B as depicted in Table 5. While not an exhaustive input/output table for this assertion, this oracle can provide confidence that the assertion will trap illegal or invalid inputs for Input_A and Input_B.

Table 5. **Test Oracle for Assertion**

| Input_A Values | Input_B Values | Assertion | Comment |
|---|---|---|---|
| 0 | 0 | True | Legal and valid values for Input_A and Input_B |
| (-1) | 0 | False | Invalid value for Input_A |
| 0 | (-1) | True | Legal and valid values for Input_A and Input_B |
| 0 | 999 | True | Legal and valid values for Input_A and Input_B |
| 0 | 1000 | False | Invalid value for Input_B |
| 99 | 999 | True | Legal and valid values for Input_A and Input_B |
| 100 | 999 | False | Invalid value for Input_A |
| 1.5 | 0 | False | Illegal value for Input_A (not an integer) |
| 0 | AAABBB | False | Illegal value for Input_B (not a real number) |

## D. TECHNICAL CONTRIBUTION

For the fourth technical contribution in this research, we proposed the use of a kernel in the controlling software for a system-of-systems to shape the dependable behavior of the system-of-systems. To develop the specifications for the controlling software in the BMDS, we proposed the development of BMK specifications in this fashion: (1) develop natural language assertions to support a common understanding of what behavior the specification is attempting to define and constrain and (2) developing assertions for which we can verify the desired behavior and timing constraints through such mechanisms as model checkers.

The technical contribution and concepts offered in the previous two chapters and this chapter concepts addresses the second of the three research questions in Chapter V. We have demonstrated that a component-based structure could be useful in the design and specification of the controlling software in a system-of-systems.

# XIII. PROTOTYPE

## A. INTRODUCTION

We will develop a prototype of the BMK to demonstrate the use of assertions in the kernel to define functional and dependability goals. Included with the prototype will be specifications of contracts between the active components of the BMK and the passive components. Additionally, we will specify distributed behaviors in the BMK to demonstrate the ability to use assertions in the kernel to achieve liveness, dependability, and survivability in a distributed environment.

We will develop specifications for the BMK from a synthesis of functional requirements, desired capabilities, and battlespace constraints. We will employ use cases for the elicitation of the functional requirements of the battle manager. We will make assumptions about the objectives of the battle manager to define desired capabilities. Finally, we will consider battlespace constraints that might impact the execution of battle-management functions during operations.

To develop the use cases for battle management, we propose the use of the kill chain for ballistic missile defense. Recall that we developed the kill chain for ballistic missile defense with the following five functions: Detect, Track, Assign Weapon, Engage, and Assess Kill. For the prototype, we will limit the scope to Track and Assign Weapon.

We employed component-based software engineering to develop the BMK prototype. We developed the Track Processing, Weapon Assignment, and Distributed Processing components as active components and define the necessary passive components and interfaces to support the activities of the active components. We used the framework depicted in Figure 7. Finally, we analyzed the prototype to determine whether it exhibits the desired behavior and provides the desired non-functional behavior. We used simple inputs of valid and invalid inputs to assess the behavior of the prototype.

## B. ASSERTIONS IN BMK SPECIFICATIONS

We will employ assertions to specify the desired behavior of the BMK prototype to include the desired dependability properties of availability, consistency, correctness, reliability, robustness, safety, and recoverability.

In the specifications of the BMK's components, we will employ the use of assertions and exception-handling routines to achieve the following design goals:

1. Fault Avoidance - Design to avoid the occurrence of software hazards.

2. Fault Warning - Design to detect conditions which could be hazardous and provide operator warning in order that the operator can take appropriate corrective action.

3. Fault Correction - Design for fault detection but also provide automatic means for self-correction.

4. Fault Tolerance - Design for fault detection but also provide alternate paths which are automatically selected.

5. Fail Operational - Design such that when a single failure or error occurs the system fails operational (and safe). It should be noted that safety may have an extra burden trying to ensure that the system is also safe in this situation. (N.B.: For this research, a system that is characterized as fail operational is one that tolerates system faults and remains operational in a safe manner.)

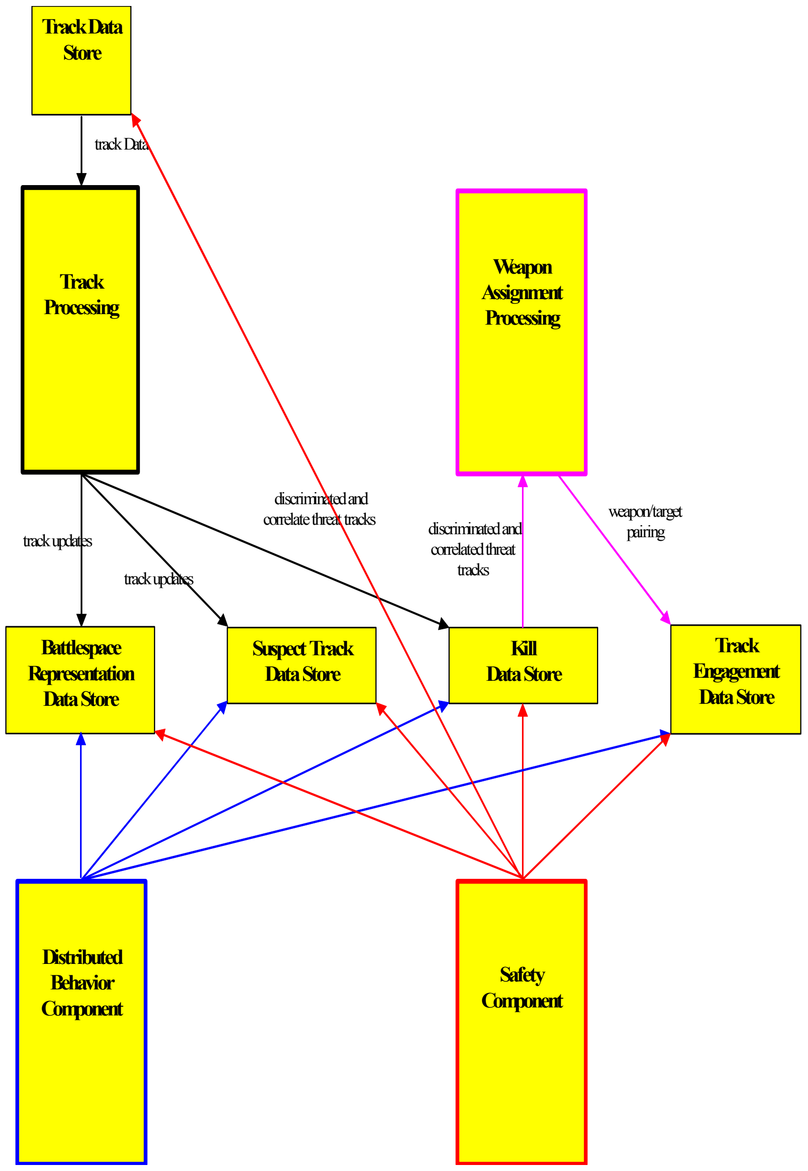
6. Fail Safe - Design such that when two independent failures or errors occur the system fails safe (but not necessarily operational). (N.B.: For this research, a system that is characterized as fail safe is one that shuts down safely after experiencing system faults.

With each assertion used to achieve the above design goals, we will specify an associated exception-handling routine to either maintain safe operations or shut down safely. We will develop a safety component to ensure that the BMK implements the specified safety policies and maintains the desired dependability properties.

## C. BMK PROTOTPE ARCHITECTURE

To reduce the impact of undesired state behavior of any given active component on any other active component, we will decouple each active component from all other active components. We will use data stores to connect the active components and use pulled data from continual polling of specific data stores as a trigger for activities in active components as depicted in Figure 13.

**Figure 13. BMK Active Components and Data Stores**


Track Data Store
track Data
Track Processing
Weapon Assignment Processing
discriminated and correlate threat tracks
discriminated and correlated threat tracks
weapon/target pairing
track updates
track updates
Battlespace Representation Data Store
Suspect Track Data Store
Kill Data Store
Track Engagement Data Store
Distributed Behavior Component
Safety Component

Along with the other benefits of decoupling active components from each other, we hypothesize that we can increase the degree of test confidence in the BMK. As the first testing step, we propose the testing of each passive component (i.e., computation) with its contract interface. The objective of this testing should be the testing of output as a function of the set of valid and invalid inputs to the contract interface. We propose the next step of testing to be the testing of each active component with its passive components connected via the contract interface. Given that active components will have state (unlike their passive counterparts), we propose that this step of testing include both black-box testing to ensure the appropriate outputs from the inputs of a test oracle and white-box testing to determine whether the active components may have exhibited coincidental correctness during black-box testing. (N.B.: For this research, we define coincidental correctness to be a characteristic of a system that can produce the correct outputs for specific inputs as defined by the system specifications. However, incorrectly implemented software in the system does not always impact the final output of the system. That is, a system that is said to demonstrate coincidental correctness does the right thing some of the time.)

Rather than using actual discrimination and correlation algorithms in the passive components, we will use relatively simple algorithms that are intended to demonstrate the effectiveness of the contract vice accuracy of discrimination and correlation algorithms. We will describe the work in the timelines that we accomplished the work to record the thinking at each step in the prototype development process.

## D. TRACK PROCESSING COMPONENT

For our prototype system, we will develop the track processing part of the BMK to satisfy the Track function of the kill chain. For the prototype, we will use the following assumptions: (1) we are not concerned about the source of the track data (i.e., radar, IR sensor, optical sensor), (2) the track data is normalized to a specific format in the data store that contains the track data, and (3) we are not concerned with the specific details of discrimination and correlation.

The first step is to develop a use case that outlines track processing in the BMK. The user goal in Track Processing is to identify ballistic-missile threat objects in the observed track data from a sensor. The following use case outlines the steps required to achieve this goal:

**Track Processing Use Case**

**Goal:** Identify ballistic missile threat objects from observed track data.

**Trigger:** Track data returned to Track Processing as a result of polling track-data store

**Actors:** Track Processing, Discrimination Computation, Correlation Computation

**Main success scenario:**

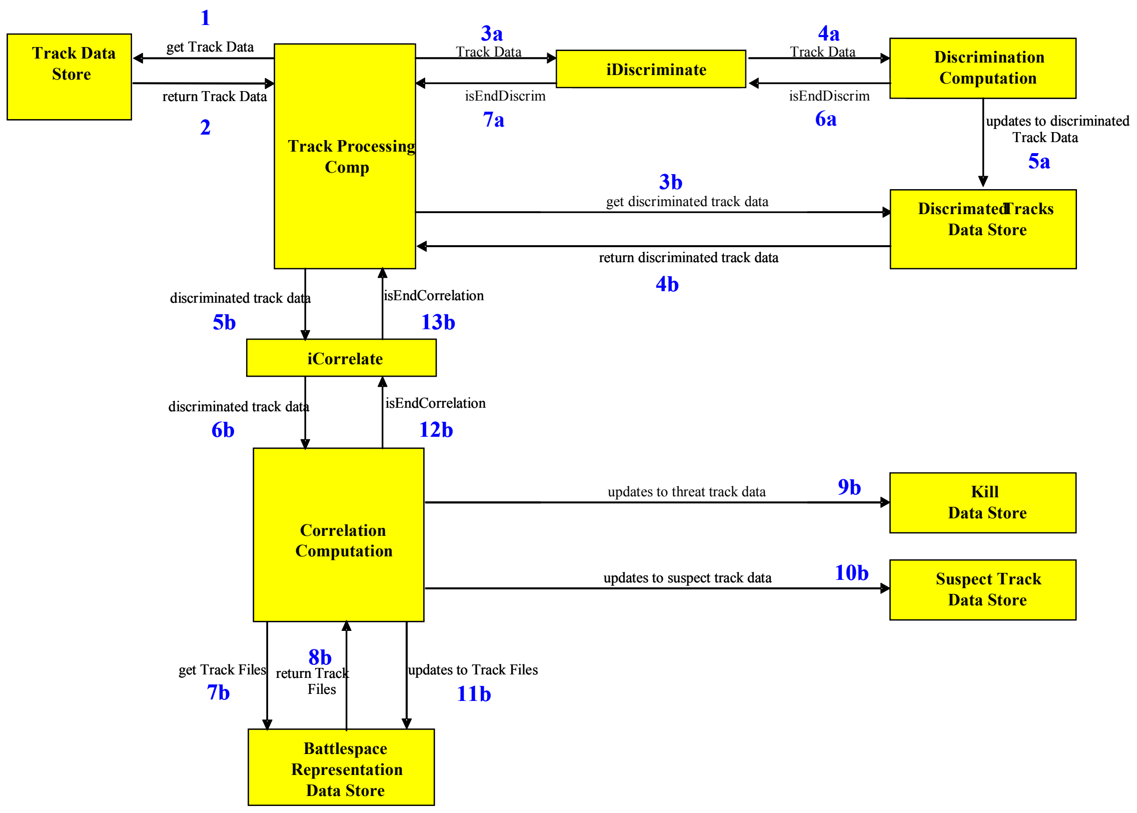
1. Track Processing polls Track Data Store for track data and Track Data Store returns track data.
2. Track Processing sends track data to Discrimination Computation to determine whether track data is a threat object.
3. Discrimination Computation stores discriminated tracks in data store and returns an end of discrimination message to Track Processing.
4. Track Processing pulls discriminated tracks from data store and sends track data to Correlation Computation.
5. Correlation Computation associates threat track to existing track file, updates track files, and returns an end of correlation message to Track Processing.

**Extensions:**

1a. Track Data-Store returns null message.
    1a1. Repeat 1 until Track Data-Store returns track data.
3a. Discrimination Computation cannot discriminate track data.
    3a1. Discrimination Computation sends track data to Suspect Data Store.
    3a2, Discrimination Computation sends Track Processing end of discrimination message.
5a. Correlation Computation cannot associate threat track to existing track file.
    5a1. Correlation Computation creates new track file for threat track
    5a2. Correlation Computation returns an end of correlation message to Track Processing.

From this use case, we developed the collaboration diagram in Figure 14 to depict the relationships of the classes suggested in the use case. Note that we have identified a Track Processing class that coordinates the activities in this work. Additionally, we have identified two classes that perform work: Discrimination Computation and Correlation Computation.

**Figure 14. Track Processing Component**



From the Track Processing use case and diagram, we can prepare a description of the components, interfaces, and data stores as well as a set of specifications for track processing. Additionally, we can develop natural language assertions and associated error-handling code that can enhance our desired dependability properties of availability, consistency, correctness, reliability, robustness, safety, and recoverability.

### 1. Track Processing Component

This is an active component that coordinates activities and computations as track data is processed. Track Processing should ensure that interrupts or slow processing in the passive components (i.e., computations) do not result in an interrupt in the processing of the track data. Track Processing will poll data from Track Data Store, and send the track data to the Discrimination Computation. Concurrently, Track Processing begins to poll the Discriminated Track Data Store. After receiving a discriminated track, Track Processing will send the discriminated track data to the Correlation Computation. Note that discrimination and correlation are concurrent activities and denoted on the diagram as 3a, 4a, etc. for the discrimination activities and 3b, 4b, etc. for the correlation activities.

The specifications for Track Processing are as follows:

a. Track Processing will continually poll Track Data Store every two seconds and pulls a single track data set from the top of the stack.

b. Track Processing will poll Track Data Store every two seconds or whenever Track Processing receives *isEndDiscrim* message – whichever event occurs first. This assertion checks for continued polling of the Track Data Store by Track Processing thereby realizing a safety property of the BMK processing track data whenever track data is presented to the BMK. If the assertion is violated, then the developed error-handling code will reset the polling in Track Processing and resume polling every two seconds.

c. Track Processing will not poll Track Data Store while discriminating current track data. This assertion checks that Track Processing only tasks a single set of track data to a single instance of the Discrimination Computation and that Track Processing will not send a different set of track data to that instance of the Discrimination Computation until it completes its processing of the current set of track data.

d. If track data from Track Data Store is not valid data, then Track Processing will discard the invalid data (e.g., null set) and poll Track Data Store in two seconds. Track Processing will not pass invalid data to Discrimination Computation. This assertion checks for valid track data to avoid processing invalid track data that may cause a system failure.

e. If Track Processing sends valid track data to Discrimination

Computation, then the Discrimination Computation and associated actions must be completed within one second of Track Processing presenting track data to iDiscrimination. This includes the update to the Discriminated Tracks Data Store. If the computation is not complete within one second, then Track Processing will terminate the tasking to the Discrimination Computation and send that set of track data to the Suspect Data Store. This assertion and associated error-handling code checks to ensure the Discrimination Computation has completed its work within the specified time limitation.

f. If Track Processing sends valid discriminated track data to the Correlation Computation, then the Correlation Computation and associated actions must be completed within one second of Track Processing presenting discriminated track data to iCorrelate. This includes the updates to the Kill Data Store, Suspect Track Data Store, and Battlespace Representation Data Store. If the processing is not complete within the specified time limit, then Track Processing will direct that Correlation Computation create a new track file for the current discriminated track data. This assertion and associated error-handling code checks to ensure the Correlation Computation has completed its work within the specified time limitation.

**2. iDiscriminate**

This is an interface that describes a contract between the discrimination activities in Track Processing and the Discrimination Computation. Along with the parameters passed to/from the computation, the interface will contain pre-conditions, post-conditions, and invariants as required to ensure that the Discrimination Computation completes its work within the specified timeframe and within the constraints set forth in the interface contract. The specifications for iDiscriminate are as follows:

a. Within one second of presenting track data to iDiscriminate, the track data will be returned to Track Processing with one of three possible labels: Threat, Benign, or Suspect; otherwise, iDiscriminate will terminate the discrimination computation and discard the invalid output of the discrimination computation. If the computation exceeds one second, then the track data will be labeled as Suspect. If track data is not valid data, then iDiscriminate will terminate processing and return *isEndDiscrim* message to Track Processing.

b. If the velocity in the track data is anything other than a real number greater than zero and less than 10 (anything other than a real number would include alpha characters, mathematical symbols, null values, etc.), then iDiscriminate will terminate processing and send *isEndDiscrim* to Track Processing.

### 3. Discrimination Computation

This is a passive component that receives track data and determines whether the track data represents a threat, benign, or suspect track. A threat track is defined as one that is identified as a tracked ballistic missile. A benign track is defined as one that is identified as a mass moving through space at a given velocity but does not represent a ballistic missile threat. A suspect threat track is defined as one that cannot be classified as either a threat or benign track, or contains features of both a threat and benign track. The specifications for the Discrimination Computation are as follows:

a. If track data contains a velocity of less than one kilometer per second, then the discrimination computation will label the track data as suspect.

b. If track data contains a velocity of greater than nine kilometers per second, then the discrimination computation will label the track data as suspect.

c. If track data contains features that indicate both a threat and a benign identification for the same track, then the discrimination computation will label the track data as suspect.

### 4. iCorrelate

This is an interface that describes a contract between the correlation activities in Track Processing and the Correlation Computation. Along with the parameters passed to/from the computation, the interface will contain pre-conditions, post-conditions, and invariants as required to ensure that the Correlation Computation completes its work within the specified timeframe and within the constraints set forth in the interface contract. The specifications for iCorrelate are as follows:

a. After Track Processing presents track data to iCorrelation, the correlation computation must return *isEndCorrelation* message to iCorrelate within two seconds. If the computation exceeds two seconds, then the track data will be toggled as

Suspect and stored in Suspect Track Data Store, and iCorrelation will send *isEndCorrelation* message to Track Processing.

b. If track data from Track Data Store is not valid data, then iCorrelation will discard track data and return *isEndCorrelation* message to Track Processing. iCorrelation will not pass invalid track data to Correlation Computation.

**5. Correlation**

This is a passive component that receives discriminated track data and correlates that data with the current track files that the Correlation Computation pulls from the Battlespace Representation Data Store.

a. All track data will be correlated to an existing track file or the correlation computation will generate a new track file.

b. If current Track File is toggled from Threat to Benign, then Correlation will send track data to Suspect Track Data Store and update Battlespace Representation Data Store with suspect track data.

c. If predicted impact point of current Track File differs from the predicted impact point of the track data (absolute value of the distance between the two impact points) by more than 50 kilometers, then the correlation computation will toggle the track file as suspect and send a copy of track file to Suspect Track Data Store.

d. If threat track file has two consecutive position updates in which the absolute difference between the two reported positions is less than three kilometers, then the correlation computation will toggle the track file as suspect and send a copy of track file to Suspect Track Data Store.

e. If track data is labeled Suspect, then the correlation computation will send a copy of the suspect track data to Suspect Track Data Store after either correlating the track data to an existing track file or creating a new track file in the Battlespace Representation Data Store.

f. The correlation computation will complete all computations, and store all correlated track data and new track files before sending *isEndCorrelation* to iCorrelation.

### 6. Kill Data Store

This data store contains the correlated threat track data.

### 7. Battlespace Representation Data Store

This data store contains the entire set of active tracks (benign, threat, and suspect) in the battlespace. .

### 8. Suspect Track Data Store

This data store contains the tracks that are classified as suspect.

### 9. Track Data Store

This data store contains the normalized track data from various sensor sources.

## E. WEAPON ASSIGNMENT COMPONENT

The next effort in developing our prototype will be developing the weapon assignment processing part of the BMK. As with Track Processing, our first step is to develop a use case that outlines weapon assignment processing in the BMK. The user goal in Weapon Assignment Processing is to assign a weapon system to engage each track in the Kill Data Store. The following use case outlines the step required to achieve this goal:

**Weapon Assignment Processing Use Case**

**Goal:** Assign a weapon system to engage each track in Kill Data Store.

**Actors:** Weapon Assignment Processing, Track Prioritization Computation, Weapon Assignment Computation

**Pre-condition:** Kill Data Store contains one or more threat tracks.

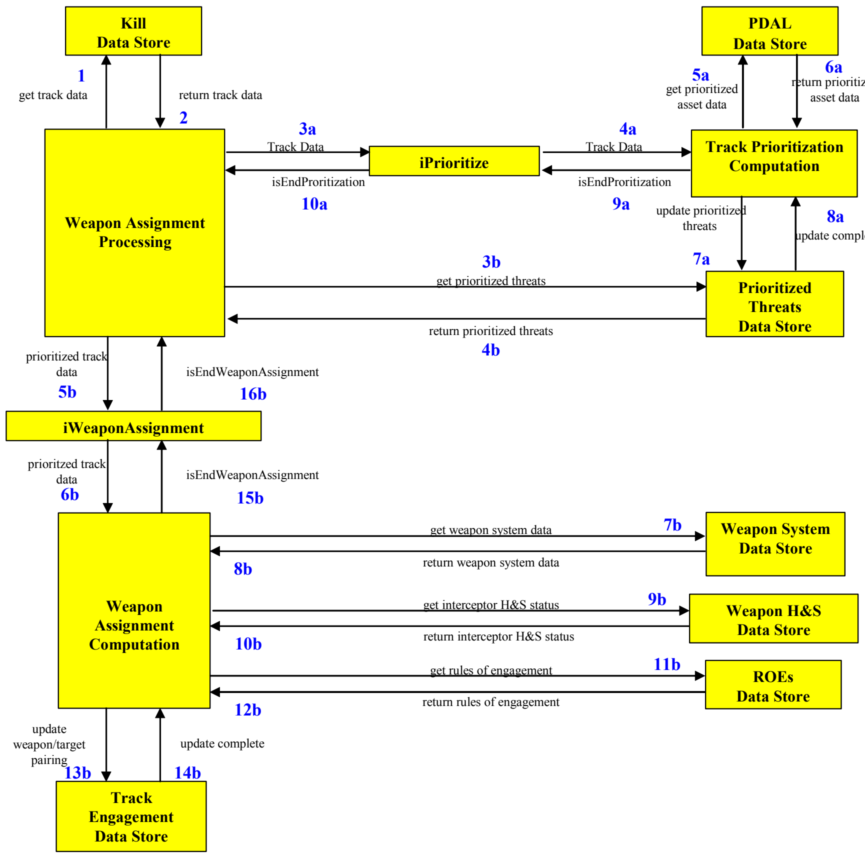
**Trigger:** Track data returned to Weapon Assignment Processing as a result of Weapon Assignment Processing polling Kill Data Store

**Main success scenario:**
1. Upon receipt of track data from Weapon Assignment Processing, Track Prioritization Computation prioritizes the threats in accordance with the PDAL data store.
2. After completing the track prioritization tasking, Track Prioritization Computation sends an end of prioritization message to Weapon Assignment Processing.

3. Upon receipt of prioritized track data from Weapon Assignment Processing, Weapon Assignment Computation determines which weapon it will assign to engage threat track, and stores the weapon assignment in Track Engagement Data Store.
4. After completing the weapon assignment tasking, Weapon Assignment Computation sends an end of weapon assignment message to Weapon Assignment Processing.

From this use case, we developed the diagram in Figure 15 to depict the relationships of the classes suggested in the use case. Note that we identified a Weapon Assignment Processing class that coordinates the activities in this work. Additionally, we identified two classes that perform work: Track Prioritization Computation and Weapon Assignment Computation.

Figure 15. Weapon Assignment Processing Component



From the use case and the diagram, we can prepare a description of the components, interfaces, and data stores as well as a set of specifications for weapon assignment processing. Additionally, we can develop natural language assertions and associated error-handling code that can enhance our desired dependability properties of availability, consistency, correctness, reliability, robustness, safety, and recoverability.

### 1. Weapon Assignment Processing

This is an active component that coordinates activities and computations for the pairing of a weapon system to a given track. Weapon Assignment Processing should ensure that interrupts or slow processing in the passive components (i.e., computations) does not result in an interrupt in the weapon assignment processing. Weapon Assignment Processing will poll data from Kill Data Store and send the track data to the Track Prioritization Computation. When triggered, Weapon Assignment Processing will task the Weapon Assignment Computation to pair a weapon to a track. Note that prioritization and weapon assignment are concurrent activities and denoted on the diagram as 3a, 4a, etc. for the prioritization activities and 3b, 4b, etc. for the weapon assignment activities. The specifications for Weapon Assignment Processing are as follows:

a. Weapon Assignment Processing will continually poll Kill Data Store every two seconds and pull a single track data set from top of stack.

b. Weapon Assignment Processing will poll Kill Data Store every two seconds or whenever Weapon Assignment Processing receives *isEndPrioritization* message – whichever event occurs first.

c. Weapon Assignment Processing will not poll Kill Data Store while prioritizing current track data.

d. If track data from Kill Data Store is not valid data, then Weapon Assignment Processing will discard returned track data and resume polling after two seconds has elapsed. Weapon Assignment Processing will not pass invalid track data to associated computations.

e. Weapon Assignment Processing will not process any track data that is identified as either Benign or Suspect. If a Weapon Assignment Processing pulls a track that is identified other than Threat, then Weapon Assignment Processing will send an alert to the user display along with the track data.

g. If Weapon Assignment Processing sends valid track data to Track Prioritization Computation, then prioritization computations and actions must be completed within two seconds of Weapon Assignment Processing presenting track data to iPrioritize. This includes the update of Prioritized Threats Data Store.

h. After Weapon Assignment Processing has received valid prioritized track data from the Prioritized Threats Data Store, it will send the prioritized track data to the Weapon Assignment Computation.

i. If Weapon Assignment Processing sends valid track data to Weapon Assignment Computation, then weapon assignment computations and actions must be completed within two seconds of Weapon Assignment Processing presenting track data to iWeaponAssignment. This includes the update of Track Engagement Data Store.

### 2. iPrioritize

This is an interface that is described as a contract between Weapon Assignment Processing and Track Prioritization Computation. Along with the parameters passed to/from the computation, the interface will contain pre-conditions, post-conditions, and invariants as required to ensure that the track prioritization computation completes its work within the specified timeframe and within the constraints set forth in the interface contract. The specifications for iPrioritize are as follows:

a. Within two seconds of presenting track data to iPrioritize, the Track Prioritization Computation must return *isEndPrioritization* message which iPrioritize will forward to Weapon Assignment Processing. If the temporal assertion fails to hold, then Weapon Assignment Processing will direct Track Prioritization Computation to place track data at the bottom of the priority stack in the Prioritized Threats Data Store, reset the polling in Weapon Assignment Processing, and resume polling of Kill Data Store.

b. If track data is not valid data, then iPrioritize will terminate processing and return *isEndPrioritization* message to Weapon Assignment Processing.

c. If the Predicted Impact Point (IPP) in the track data is anything other than a string of 8 integers (N.B.: anything other than a string of 8 integers would include real numbers other than integers, alpha characters, mathematical symbols, null values, etc), then iPrioritization will terminate processing and send *isEndPrioritization* message to Weapon Assignment Processing.

### 3. Track Prioritization Computation

This is a passive component that receives track data and determines the priority of the track based upon the PDAL Data Store. The specifications for the Track Prioritization Computation are as follows:

a. After Weapon Assignment Processing presents track data to iPrioritize, the track prioritization computation must prioritize the threat track in accordance with the Prioritized Defended Asset List (PDAL). If the prioritization computation exceeds one second, then the track will be placed at the bottom of the priority stack in the Prioritized Threats Data Store and the prioritization computation will return *isEndPrioritization* message to iPrioritize.

b. If the absolute value of the distance from the IPP to any asset on the PDAL is equal to or less than 50 kilometers, then the threat will be assign a priority to the track that is equal to the priority of the PDAL asset that is within that absolute value of 50 kilometers of the IPP. If there are more than one PDAL assets within 50 kilometers of the IPP, then the priority of the track will be equal to the asset that has the highest priority within a circle that is defined having a 50 kilometer radius and the center is the IPP. If two or more tracks hold the same priority based upon threatening the same asset, then the track with the shortest time remaining to impact will be given the higher priority.

c. If a the absolute value of the distance from the IPP to any asset on the PDAL is greater than 50 kilometers, then the threat will be labeled as "Deliberate Pass), monitored, but not engaged unless the IPP moves within 50 kilometers of a defended asset. No weapon will be assigned to engage Deliberate Pass tracks.

**4. iWeaponAssignment**

This is an interface that is described as a contract between Weapon Assignment Processing and Weapon Assignment Computation. Along with the parameters passed to/from the computation, the interface will contain pre-conditions, post-conditions, and invariants as required to ensure that the Weapon Assignment Computation completes its work within the specified timeframe and within the constraints set forth in the interface contract. The specifications for iWeaponAssignment are as follows:

a. After Weapon Assignment Processing presents track data to iWeaponAssignment, the correlation computation must return *isEndWeaponAssignment* message to iWeaponAssignment within two seconds. If the computation exceeds two seconds, then the track will be toggled as "Not Assigned" in Track Engagement Data Store, and iWeaponAssignment will send *isEndWeaponAssignment* message to Weapon Assignment Processing.

b. If track data from Weapon Assignment Processing is not valid data, then iWeaponAssignment will discard track data and return *isEndWeaponAssignment* message to Weapon Assignment Processing. iWeaponAssignment will not pass null track data to weapon assignment computation.

**5. Weapon Assignment Computation**

This is a passive component that will receive prioritized track data from Weapon Assignment Processing and assign a weapon to each threat based upon the characteristics of the weapon systems, health and status of the available weapon systems, and rules of engagement as defined by the user. The specifications for the Weapon Assignment Computation are as follows:

a. Each track that is identified as a threat and prioritized will be paired with a weapon. Any track that is identified as either benign or suspect will If a weapon system is not available to engage the track, then the Weapon Assignment Computation will tag the track as Unassigned Threat Track.

b. After all computations and updates are completed, Weapon Assignment Computation will send *isEndWeaponAssignment* message to Weapon

Assignment Processing.

#### 6. PDAL Data Store

This data store contains the prioritized threats as computed by the Track Prioritization Computation.

#### 7. Prioritized Threats Data Store

This data store contains the prioritized threats as computed by the Track Prioritization Computation.

#### 8. Weapon System Data Store

This data store contains information about each weapon system in the BMDS to include range and accuracy of organic sensor, altitude and range of interceptor, maximum number of available launchers, reload time, and maximum number of concurrent engagements.

#### 9. Weapon H&S Data Store

This data store contains the continually updated health and status information of each weapon associated with the battle manager to include readiness of weapon system, number of interceptors that are at the ready, current engagement assignments assigned to the weapon system, and current engagements of the weapon system.

#### 10. ROEs Data Store

This data store contains the rules of engagement (ROEs) as set in the BMD planning phase to include shot doctrine, firing trigger (e.g., first available shot, 90% probability of kill ($P_K$), desired interceptor reserve).

#### 11. Track Engagement Data Store

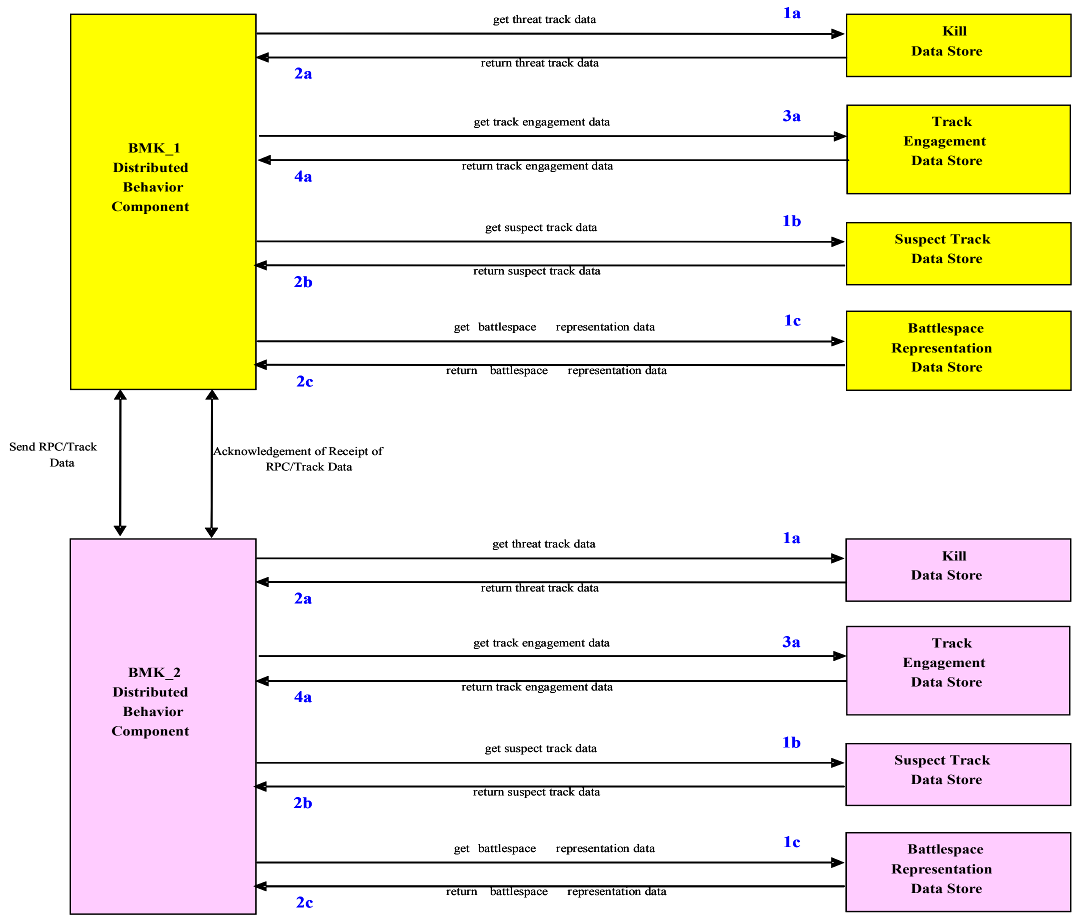
This data store contains the current engagement status (Engaged, Not Assigned, or Deliberate Pass) of every prioritized track.

## F. DISTRIBUTED BEHAVIOR COMPONPENT

The BMK Distributed Behavior Component is an active component that monitors the status of multiple, independent battle-management processes. The first process copies the track data in Kill Data Store and sends this data to other BMKs as situational awareness information. The second process copies the track data in the Battlespace Representation Data Store and sends this data to other BMKs and C2 elements as situational awareness information. The third process monitors the Track Engagement

Data Store to find assigned engagements with weapon systems that require additional sensor support (modify WAP). The fourth process copies the engagement data in Track Engagement Data Store and provides it to other BMKs and C2 elements as situational awareness information. The fifth process searches the engagement data in Track Engagement Data Store to find Unassigned Threat Tracks that have remained unassigned for more than 30 seconds. The sixth process monitors the Suspect Track Data Store to find suspect track data that have remained in the suspect status for more than 30 seconds without new track updates. The BMK Distributed Behavior Component is depicted below in Figure 16:

**Figure 16. BMK Distributed Behavior Component**


BMK_1 Distributed Behavior Component
get threat track data
1a
Kill Data Store
return threat track data
2a
get track engagement data
3a
Track Engagement Data Store
return track engagement data
4a
get suspect track data
1b
Suspect Track Data Store
return suspect track data
2b
get battlespace representation data
1c
Battlespace Representation Data Store
return battlespace representation data
2c
Send RPC/Track Data
Acknowledgement of Receipt of RPC/Track Data
BMK_2 Distributed Behavior Component
get threat track data
1a
Kill Data Store
return threat track data
2a
get track engagement data
3a
Track Engagement Data Store
return track engagement data
4a
get suspect track data
1b
Suspect Track Data Store
return suspect track data
2b
get battlespace representation data
1c
Battlespace Representation Data Store
return battlespace representation data
2c

The specifications for the BMK Distributed Behavior Component are as follows:

### 1. Situational Awareness: Kill Data Store

BMK Distributed Behavior Component pulls track data from Kill Data Store every 5 seconds and sends it to other BMKs.

### 2. Situational Awareness: Battlespace Representation Data Store

BMK Distributed Behavior Component pulls track data from Battlespace Representation Data Store every 10 seconds, and sends it to other BMKs and C2 elements.

### 3. Sensor Support RPC

BMK Distributed Behavior Component searches for engagement assignments that require additional sensor support as stored in Track Engagement Data Store and requests sensor support from another BMK.

### 4. Situational Awareness: Track Engagement Data Store

BMK Distributed Behavior Component pulls track data from Track Engagement Data Store every 5 seconds, and sends it to other BMKs and C2 elements.

### 5. Weapon Assignment RPC

BMK Distributed Behavior Component searches the engagement data in Track Engagement Data Store to find Unassigned Threat Tracks that have remained unassigned for more than 30 seconds, and requests transfer of track responsibility and weapon assignment to another BMK.

### 6. Sensor Support RPC: Suspect Track Data Store

BMK Distributed Behavior Component searches for suspect track data that have remained in the suspect status for more than 30 seconds without new track updates data in engagement assignments and requests sensor support from another BMK.

## G. SAFETY COMPONENT

Safety is a system attribute. System software faults can lead to system accidents; however, the system software safety is highly dependent on the operational application of the system as well as the environment in which we operate the system. Software is not unsafe when considered in isolation. It is only when the software is integrated into a system that it can contribute to system accidents.

For the battle manager, we will use the following partial failure analysis for ballistic missile defense that will form the basis for our battle-management safety policies:

1. Failure of weapon to engage a threat track

    a. Potential causal factor

    Failure of Battle Manager to assign weapon to threat track in sufficient time for weapon to engage threat track

    b. Safety policies

(1) For each identified threat track, the Battle Manager must either assign a weapon to engage that threat track or classify the threat track as Deliberate Pass within thirty seconds of a track being identified as a threat track.

(2) For each track that is presented to the Battle Manager, the track processing code must identify the track as threat, benign, or suspect within fifteen seconds of track presentation to the Battle Manager.

(3) For each track that is labeled suspect by the Battle Manager, the track processing code must classify the suspect track as either threat or benign within thirty seconds of the original classification of the track as suspect.

(4) A threat will not toggle between Deliberate Pass and a weapon assignment more than two times.

(5) A suspect track will be updated within thirty seconds of its previous update or original classification – whichever is later.

(6) A track that has a velocity of less than one kilometer per second will not be identified as a ballistic missile threat.

(7) Every processed track in the BMD battlespace will be updated within thirty seconds of its previous update or original classification – whichever is later.

2. Successful engagement of weapon on non-threat track

a. Potential causal factors

(1) Weapon assignment pairing weapon to either benign or suspect track.

(2) Prioritized threats include one or more benign or suspect track(s).

b. Safety policies

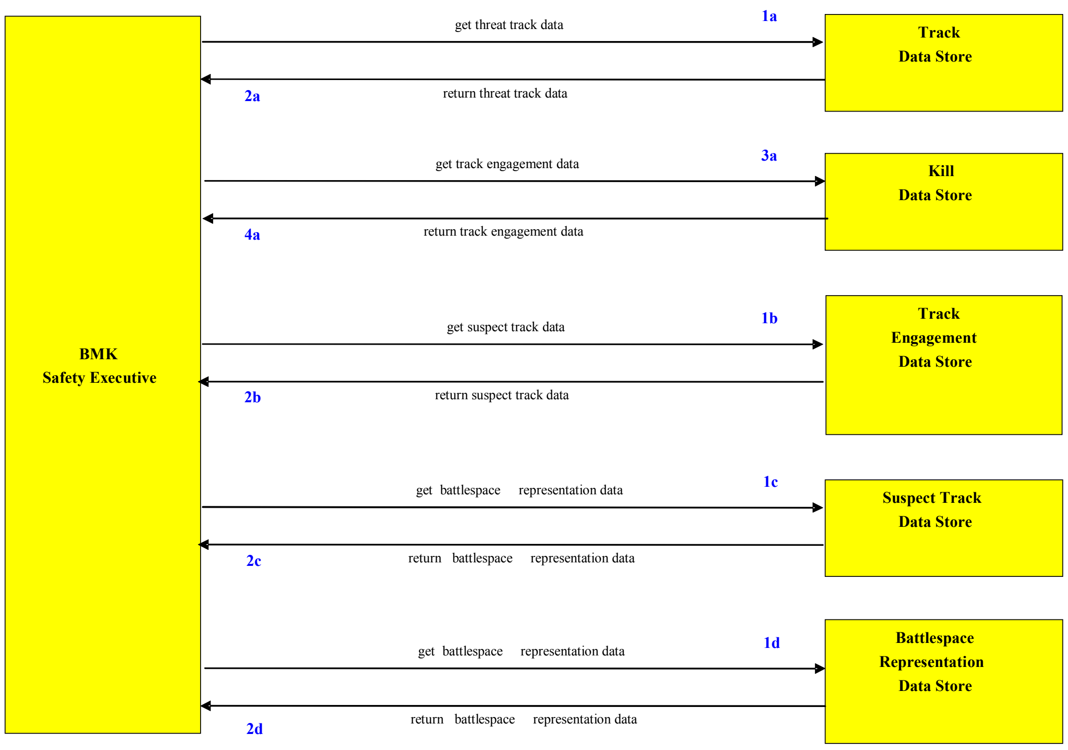
(1) Only threat tracks will be paired with a weapon for engagement. Suspect tracks and benign tracks will not be paired with a weapon for engagement.

(2) Prioritized threats will only come from threat track storage. Suspect tracks and benign tracks will not be prioritized for weapon assignment.

Rather than scattering the safety assurance through all the active components, we will employ the use of a safety component to ensure compliance with our identified safety policies. The checking for compliance to the safety policies might be managed easier in a single entity (i.e., safety component) because the complexity of the battle manager could make the safety-policy compliance checking more difficult. Our premise is that verification of the safety component will suffice for ensuring the enforcement of the safety policies over the battle manager. While the safety component may not be responsible for implementing all the safety policies, it will be responsible for checking compliance to all the implemented safety policies and exercising implemented exception-handling code for detected safety violations.

For the prototype, the BMK Safety Component will monitor the BMK data stores and perform checks to determine whether the BMK active components are complying with the implemented safety policies. Upon a detection of a safety policy violation, the BMK Safety Component will return the BMK to a safe operational state if possible or fail safe at a minimum. As such, it is equally important to specify the exception-handling routine for the violation of a given assertion as the specification of the desired objective for that assertion. The BMK Safety Component is depicted below in Figure 17:

Figure 17. BMK Safety Component



BMK Safety Executive
get threat track data
1a
Track Data Store
return threat track data
2a
get track engagement data
3a
Kill Data Store
return track engagement data
4a
get suspect track data
1b
Track Engagement Data Store
return suspect track data
2b
get battlespace representation data
1c
Suspect Track Data Store
return battlespace representation data
2c
get battlespace representation data
1d
Battlespace Representation Data Store
return battlespace representation data
2d

The BMK Safety Component is an active component that monitors the status of three concurrent BMK processes. The first process ensures that Weapon Assignment has assigned a weapons system against each identified threat in the Kill Data Store. The second process ensures that the tracks in the Suspect Data Store are updated as specified. The third process ensures that the tracks in the Battlespace Representation Data Store are updated as specified. The specifications for the BMK Safety Component are as follows:

1. Threat Engagement Status. A valid threat track in the Kill Data Store will be paired with a weapon system to engage the valid threat track within thirty seconds of the first instance of that track in the Kill Data Store. If a track in the Kill Data Store has either not been assigned a weapon system to engage it or classified as Deliberate Pass within thirty seconds after entering the track into the Kill Data Store, then the Safety Component will send a warning to user display.

2. Threat Engagement Toggle. A valid threat track can toggle between deliberate pass and a weapon system pairing only one time. If a track toggles between Deliberate Pass and weapon assigned (i.e., deliberate pass to weapon assigned is one toggle and weapon assigned to deliberate pass is one toggle) more than two times, then a warning will be sent to the user display.

3. Unassigned Threat Track. A valid threat track in the Track Engagement Data Store will be paired with a weapon system to engage that valid threat track within thirty seconds of the first instance of that track in the Track Engagement Data Store. If an unassigned threat track is not assigned a weapon system to engage it within thirty seconds of the first instance of the Weapon Assignment Computation posting this track into the Track Engagement Data Store, then a warning will be sent to the user display.

4. Threat Track Identification. A track in the Kill Data Store can only be identified as a Threat. Any other identification for a track in the Kill Data Store is invalid. Safety Component will pull all invalid tracks from the Kill Data Store and send an alert to the user display along with the invalid track data.

5. Suspect Track Status. Each suspect track must be classified as either threat or benign within thirty seconds of the first instance of the track that is posted in the Suspect Track Data Store. If the track in the Suspect Track Data Store has not been classified as either threat or benign within thirty seconds after the first instance of that track appears in the Suspect Track Data Store, then the Safety Component will send a warning to user display along with the track data.

6. Suspect Track Update. A suspect track file must be updated within thirty seconds of its previous update after the first instance of the track in the Suspect Track

Data Store. If a suspect track file has not been updated within thirty seconds from its previous update, then Safety Component will send suspect track warning to user display along with the track data.

7. Suspect Track Velocity Status. A suspect track cannot have more than two consecutive velocity updates of less than one kilometer per second. If suspect track file has three consecutive velocity updates of less than one kilometer per second, then Correlation Computation will send suspect track data to user display.

8. Battlespace Representation Track Update. A track file must be updated within thirty seconds of its previous update after the first instance of the track in the Battlespace Representation Data Store. If a track file (benign or threat) in the Battlespace Representation Data Store has not been updated for thirty seconds or more, then the BMK Safety Component will toggle the track file to Suspect and send track data to Suspect Track Data Store.

9. Continuity of Operations: Track Processing

a. A track must be pulled from Track Data Store within fifteen seconds of its appearance in the Track Data Store.

b. The first instance of a track file must appear in Battlespace Representation Data Store within thirty seconds of the first appearance of that valid track data in the Track Data Store.

c. If (9a) and (9b) are not true at the same time, then Safety Component will send reset signal to Track Processing Component.

d. If (9a) and (9b) are not true at the same time following a Track Processing reset that occurred within the past sixty seconds, then the Safety Component will deem the BMK as inoperable and direct the transfer of control to another BMK.

10. Continuity of Operations: Weapon Assignment Processing

a. A track must be pulled from Kill Data Store within fifteen seconds of its appearance in the Kill Data Store.

b. The first instance of a threat track file must appear in Track

Engagement Data Store within thirty seconds of the first appearance of that valid threat track data in the Kill Data Store.

c. If (10a) and (10b) are not true at the same time, then Safety Component will send reset signal to Weapon Assignment Processing Component.

d. If (10a) and (10b) are not true at the same time following a Weapon Assignment Processing reset that occurred within the past sixty seconds, then the Safety Component will deem the BMK as inoperable and direct the transfer of control to another BMK. No other data will be processed by the BMK.

## H. ANALYSIS OF PROTOTPE

We specified the BMK prototype with the use of assertions. We identified error-handling procedures for violations of the assertions. Functionally, the prototype should exhibit the desired behavior to support battle-management operations.

To support dependability of the BMK, we modified our assertions to specify time constraints that are driven by the operational battlespace and added assertions that would support our design goals. In the following discussion, we will assess the assertions that we used to specify the prototype to determine whether the BMK contains the seven dependability properties that we identified for the battle manager.

### 1. Availability

In Chapter VI, we defined availability as the probability that a system is operating correctly and is ready to perform its desired functions.

#### *a. Track*

For the Track function, we developed two assertions that support the availability of the Track Processing component:

(1) Track Processing will continually poll Track Data Store every two seconds and pulls a single track data set from top of stack. This assertion specifies the time constraint for polling the data store for track data. This assertion supports continuity of operations.

(2) Track Processing will poll Track Data Store every two seconds or whenever Track Processing receives *isEndDiscrim* message – whichever

event occurs first. If the assertion is violated, then the developed error-handling code will reset the polling in Track Processing and resume polling every two seconds.

We chose to use temporal assertions to ensure that Track Processing will continually poll the Track Data Store and process track data. Note that Track Processing will return to poll new track data at either two-second intervals or whenever discrimination is completed. As such, the BMK will continually poll and process data as specified. If the assertion is violated, then Track Processing will reset itself and resume polling at two-second intervals.

### b. Weapon Assignment

For Weapon Assignment function, we developed two assertions that support the availability of the Weapon Assignment Processing component:

(1) Weapon Assignment Processing will continually poll Kill Data Store every two seconds and pull a single track data set from top of stack.

(2) Weapon Assignment Processing will poll Kill Data Store every two seconds or whenever Weapon Assignment Processing receives *isEndPrioritization* message – whichever event occurs first. Within two seconds of presenting track data to iPrioritize, the Track Prioritization Computation must return *isEndPrioritization* message which iPrioritize will forward to Weapon Assignment Processing. If the temporal assertion fails to hold, then Weapon Assignment Processing will direct Track Prioritization Computation to place track data at the bottom of the priority stack in the Prioritized Threats Data Store, reset the polling in Weapon Assignment Processing, and resume polling of Kill Data Store.

We chose to use temporal assertions to ensure that Weapon Assignment Processing will continually poll the Kill Data Store and develop a weapon/target pairing for each threat in the Kill Data Store. Note that Weapon Assignment Processing will return to poll new threat data at either two-second intervals or whenever threat prioritization is completed. As such, Weapon Assignment Processing will continually pole and process data as specified. If the assertion is violated, then Weapon Assignment Processing will reset itself and resume polling at two-second intervals.

### c. *Component Interfaces*

In each interface between an active component and a passive component (i.e., iDiscriminate, iCorrelate, iPrioritize, and iWeaponAssignment), we specified a temporal invariant for the return of output from the passive component. The intent is to ensure that the BMK would continue to operate correctly if a passive component failed to return the required output in the specified constraint of the temporal assertion.

For example, consider the following assertion in the specification for iDiscriminate:

*Within one second of presenting track data to iDiscriminate, the track data will be returned to Track Processing with one of three possible labels: Threat, Benign, or Suspect; otherwise, iDiscriminate will terminate the discrimination computation and discard the invalid output of the discrimination computation. If the computation exceeds one second, then the track data will be labeled as Suspect. If track data is not valid data, then iDiscriminate will terminate processing and return isEndDiscrim message to Track Processing.*

Case #1: Consider the case in which track data is not valid data (e.g., the null set). This is a violation of the precondition assertion for iDiscriminate. The discrimination process will be terminated, and Track Processing will resume polling and processing track data. The system continues to be available for operations.

Case #2: Consider the case in which discrimination computation does not conclude within one second. The invariant temporal assertion in iDiscriminate will be violated. The discrimination computation will be terminated, and Track Processing will resume polling and processing track data. The system continues to be available for operations.

Case #3: Consider the case in which an output from the discrimination computation is something other than Threat, Benign, or Suspect. This is a violation of the post-condition assertion for iDiscriminate. The discrimination computation will be terminated, and Track Processing will resume polling and processing track data. The system continues to be available for operations.

#### *d. Findings and Conclusions*

The assertions in Track Processing component, Weapon Assignment Processing component, and the component interfaces could support the availability of the BMK to coordinate battle-management activities. We asserted the desired behavior with time constraints to ensure that the BMK continues to process track data as specified and provided error-handling specifications for violations of the assertions to support continuity of operations. Thus, we conclude that the assertions support the availability of the BMK as defined for this research.

### 2. Consistency

In Chapter VI, we defined consistency as the property that invariants will always hold true in the system.

#### *a. Track*

For the Track function, we developed two assertions that support the consistency of the Track Processing component:

(1) Track Processing will poll Track Data Store every two seconds or whenever Track Processing receives *isEndDiscrim* message – whichever event occurs first. If the assertion is violated, then the developed error-handling code will reset the polling in Track Processing and resume polling every two seconds.

(2) Track Processing will not poll Track Data Store while discriminating current track data.

#### *b. Weapon Assignment*

For the Weapon Assignment function, we developed two assertions that support the consistency of the Weapon Assignment component:

(1) Weapon Assignment Processing will poll Kill Data Store every two seconds or whenever Weapon Assignment Processing receives *isEndPrioritization* message – whichever event occurs first. Within two seconds of presenting track data to iPrioritize, the Track Prioritization Computation must return *isEndPrioritization* message which iPrioritize will forward to Weapon Assignment Processing. If the temporal assertion fails to hold, then Weapon Assignment Processing will direct Track Prioritization Computation to place track data at the bottom of the

priority stack in the Prioritized Threats Data Store, reset the polling in Weapon Assignment Processing, and resume polling of Kill Data Store.

(2) Weapon Assignment Processing will not poll Kill Data Store while prioritizing current track data.

#### *c. Component Interfaces*

In each interface between an active component and a passive component (i.e., iDiscriminate, iCorrelate, iPrioritize, and iWeaponAssignment), we specified a temporal invariant for the return of output from the passive component. The intent is to ensure that the BMK would continue to operate correctly if a passive component failed to return the required output in the specified constraint of the temporal assertion.

For example, consider the following assertion in the specification for iCorrelate:

*After Track Processing presents track data to iCorrelation, the correlation computation must return isEndCorrelation message to iCorrelate within two seconds. If the computation exceeds two seconds, then the track data will be toggled as Suspect and stored in Suspect Track Data Store, and iCorrelation will send isEndCorrelation message to Track Processing.*

Consider the case in which correlation computation does not conclude within two seconds. The invariant temporal assertion in iCorrelate will be violated. The correlation computation will be terminated, the track data will be labeled as Suspect, and Track Processing will resume polling and processing track data. The error-handling procedure terminates the correlation process and Track Processing resumes polling. The desired invariant behavior is maintained.

#### *d. Findings and Conclusions*

The assertions in Track Processing component, Weapon Assignment Processing component, and the component interfaces could support the consistency of the BMK to coordinate battle-management activities. We asserted the invariants for the BMK that will always hold true, and provided error-handling specifications for violations of the assertions by resetting the failed component to the desired operational status (i.e., fail operational). Thus, we conclude that the assertions support the consistency of the BMK as defined for this research.

### 3. Correctness.

In Chapter VI, we defined correctness as a characteristic of a system that precisely exhibits predictable behavior at all times as defined by the system specifications.

#### *a. Track*

For the Track function, we developed two assertions that support the correctness of the Track Processing component:

(1) Track Processing will continually poll Track Data Store every two seconds and pulls a single track data set from top of stack.

(2) Track Processing will poll Track Data Store every two seconds or whenever Track Processing receives *isEndDiscrim* message – whichever event occurs first. If the assertion is violated, then the developed error-handling code will reset the polling in Track Processing and resume polling every two seconds.

We chose to use temporal assertions to ensure that Track Processing will continually poll the Track Data Store and process track data. Note that Track Processing will return to poll new track data at either two-second intervals or whenever discrimination is completed. As such, the BMK will continually poll and process data as specified. If the assertion is violated, then Track Processing will reset itself and resume polling at two-second intervals.

#### *b. Weapon Assignment*

For Weapon Assignment function, we developed two assertions that support the correctness of the Weapon Assignment Processing component:

(1) Weapon Assignment Processing will continually poll Kill Data Store every two seconds and pull a single track data set from top of stack.

(2) Weapon Assignment Processing will poll Kill Data Store every two seconds or whenever Weapon Assignment Processing receives *isEndPrioritization* message – whichever event occurs first. Within two seconds of presenting track data to iPrioritize, the Track Prioritization Computation must return

*isEndPrioritization* message which iPrioritize will forward to Weapon Assignment Processing. If the temporal assertion fails to hold, then Weapon Assignment Processing will direct Track Prioritization Computation to place track data at the bottom of the priority stack in the Prioritized Threats Data Store, reset the polling in Weapon Assignment Processing, and resume polling of Kill Data Store.

We chose to use temporal assertions to ensure that Weapon Assignment Processing will continually poll the Kill Data Store and develop a weapon/target pairing for each threat in the Kill Data Store. Note that Weapon Assignment Processing will return to poll new threat data at either two-second intervals or whenever threat prioritization is completed. As such, Weapon Assignment Processing will continually pole and process data as specified. If the assertion is violated, then Weapon Assignment Processing will reset itself and resume polling at two-second intervals.

#### *c. Component Interfaces*

In each interface between an active component and a passive component (i.e., iDiscriminate, iCorrelate, iPrioritize, and iWeaponAssignment), we specified a temporal invariant for the return of output from the passive component. The intent is to ensure that the BMK would continue to operate correctly if a passive component failed to return the required output in the specified constraint of the temporal assertion.

For example, consider the following assertion in the specification for iPrioritize:

*Within two seconds of presenting track data to iPrioritize, the Track Prioritization Computation must return isEndPrioritization message which iPrioritize will forward to Weapon Assignment Processing.*

If the temporal assertion fails to hold, then Weapon Assignment Processing will direct Track Prioritization Computation to place track data at the bottom of the priority stack in the Prioritized Threats Data Store, reset the polling in Weapon Assignment Processing, and resume polling of Kill Data Store.

Consider the case in which prioritization computation does not conclude within two seconds. The invariant temporal assertion in iPrioritize will be violated. The prioritization computation will be terminated, the track data will be placed at the bottom of the priority stack of prioritized threats. Weapon Assignment Processing will resume polling and processing threat track data. The desired behavior is achieved.

#### *d. Findings and Conclusions*

The assertions in Track Processing component, Weapon Assignment Processing component, and the component interfaces could support the correctness of the BMK to coordinate battle-management activities. We asserted the desired behavior with time constraints to ensure that the BMK continues to process track data as specified and provided error-handling specifications for violations of the assertions to support continuity of operations. Thus, we conclude that the assertions support the correctness of the BMK as defined for this research.

### 4. Reliability

In Chapter VI, we defined reliability as the property that a system can operate continuously without experiencing a failure.

#### *a. Track*

For the Track function, we developed two assertions that support the reliability of the Track Processing component:

(1) Track Processing will continually poll Track Data Store every two seconds and pulls a single track data set from top of stack.

(3) Track Processing will poll Track Data Store every two seconds or whenever Track Processing receives *isEndDiscrim* message – whichever event occurs first. If the assertion is violated, then the developed error-handling code will reset the polling in Track Processing and resume polling every two seconds.

We chose to use temporal assertions to ensure that Track Processing will continually poll the Track Data Store and process track data. Note that Track Processing will return to poll new track data at either two-second intervals or whenever discrimination is completed. As such, the BMK will continually poll and process data as specified. If the assertion is violated, then Track Processing will reset itself and resume polling at two-second intervals.

### *b. Weapon Assignment*

For Weapon Assignment function, we developed two assertions that support the reliability of the Weapon Assignment Processing component:

(1) Weapon Assignment Processing will continually poll Kill Data Store every two seconds and pull a single track data set from top of stack.

(2) Weapon Assignment Processing will poll Kill Data Store every two seconds or whenever Weapon Assignment Processing receives *isEndPrioritization* message – whichever event occurs first. Within two seconds of presenting track data to iPrioritize, the Track Prioritization Computation must return *isEndPrioritization* message which iPrioritize will forward to Weapon Assignment Processing. If the temporal assertion fails to hold, then Weapon Assignment Processing will direct Track Prioritization Computation to place track data at the bottom of the priority stack in the Prioritized Threats Data Store, reset the polling in Weapon Assignment Processing, and resume polling of Kill Data Store.

We chose to use temporal assertions to ensure that Weapon Assignment Processing will continually poll the Kill Data Store and develop a weapon/target pairing for each threat in the Kill Data Store. Note that Weapon Assignment Processing will return to poll new threat data at either two-second intervals or whenever threat prioritization is completed. As such, Weapon Assignment Processing will continually pole and process data as specified. If the assertion is violated, then Weapon Assignment Processing will reset itself and resume polling at two-second intervals.

### *c. Component Interfaces*

In each interface between an active component and a passive component (i.e., iDiscriminate, iCorrelate, iPrioritize, and iWeaponAssignment), we specified a temporal invariant for the return of output from the passive component. The intent is to ensure that the BMK would continue to operate correctly if a passive component failed to return the required output in the specified constraint of the temporal assertion.

For example, consider the following assertion in the specification for iWeaponAssignment:

*After Weapon Assignment Processing presents track data to iWeaponAssignment, the correlation computation must return isEndWeaponAssignment message to iWeaponAssignment within two seconds.*

If the computation exceeds two seconds, then the track will be toggled as "Not Assigned" in Track Engagement Data Store, and iWeaponAssignment will send *isEndWeaponAssignment* message to Weapon Assignment Processing.

Consider the case in which weapon assignment computation does not conclude within two seconds. The invariant temporal assertion in iWeaponAssignment will be violated. The weapon assignment computation will be terminated and the track data will be labeled as not assigned. Weapon Assignment Processing will resume polling and processing threat track data. The BMK continues to operate without experiencing a failure.

#### *d. Findings and Conclusions*

The assertions in Track Processing component, Weapon Assignment Processing component, and the component interfaces could support the reliability of the BMK to coordinate battle-management activities. We asserted the desired behavior with time constraints to ensure that the BMK continues to process track data as specified and provided error-handling specifications for violations of the assertions to support continuity of operations. Thus, we conclude that the assertions support the reliability of the BMK as defined for this research.

### 5. Robustness

In Chapter VI, we defined robustness as a characteristic of a system that is failure and fault tolerant.

#### *a. Track*

For the Track function, we developed an assertion that supports the robustness of the Track Processing component:

*If track data from Track Data Store is not valid data, then Track Processing will discard the invalid data (e.g., null set) and poll Track Data Store in two seconds. Track Processing will not pass invalid data to Discrimination Computation.*

This assertion checks for valid track data to avoid processing invalid track data that may cause a system failure. We chose to assert that only valid data would be processed. Anything other than valid data (e.g., null set) will be discarded. If the assertion is violated, then Track Processing will reset itself and resume polling at two-second intervals. This supports the fault-tolerant behavior for Track Processing in that it will consider all track data pulled from the data store but it will only process valid track data.

#### ***b. Weapon Assignment***

For Weapon Assignment function, we developed an assertion that supports the robustness of the Weapon Assignment Processing component:

*If track data from Kill Data Store is not valid data, then Weapon Assignment Processing will discard returned track data and resume polling after two seconds has elapsed. Weapon Assignment Processing will not pass invalid track data to associated computations.*

We chose to assert that only valid data would be processed. Anything other than valid data (e.g., null set) will be discarded. If the assertion is violated, then Weapon Assignment Processing will reset itself and resume polling at two-second intervals. This supports the fault-tolerant behavior for Weapon Assignment Processing in that it will consider all threat track data pulled from the data store but it will only process valid threat track data.

#### ***c. Component Interfaces***

In each interface between an active component and a passive component (i.e., iDiscriminate, iCorrelate, iPrioritize, and iWeaponAssignment), we specified a precondition and post-condition for the passing of information through the interfaces. The intent is to ensure that the BMK would continue to operate correctly if either the active component sent invalid data to the interface or a passive component failed to return the required output in the specified constraint of the assertion.

For example, consider the following assertion in the specification for iDiscriminate:

*Within one second of presenting track data to iDiscriminate, the track data will be returned to Track Processing with one of three possible labels: Threat, Benign, or Suspect; otherwise, iDiscriminate will terminate the discrimination computation and discard the invalid output of the discrimination computation. If the computation exceeds one second, then the track data will be labeled as Suspect. If track data is not valid data, then iDiscriminate will terminate processing and return isEndDiscrim message to Track.*

(1) Precondition. Consider the case in which track data is not valid data (e.g., the null set). This is a violation of the precondition assertion for iDiscriminate. The discrimination process will be terminated, and Track Processing will resume polling and processing track data. The system continues to be available for operations.

(2) Post-condition. Consider the case in which an output from the discrimination computation is something other than Threat, Benign, or Suspect. This is a violation of the post-condition assertion for iDiscriminate. The discrimination computation will be terminated, and Track Processing will resume polling and processing track data. The system continues to be available for operations.

#### *d. Findings and Conclusions*

The assertions in Track Processing component, Weapon Assignment Processing component, and the component interfaces could support the robustness of the BMK to coordinate battle-management activities. We asserted the desired behavior with time constraints to ensure that the BMK continues to process track data as specified and provided error-handling specifications for violations of the assertions to support continuity of operations. Thus, we conclude that the assertions support the robustness of the BMK as defined for this research.

### 6. Safety

In Chapter VI, we defined safety as the property of avoiding a catastrophic outcome given a system fails to operate correctly. We developed the following eight assertions and error-handling procedures in the Safety Component that monitor the processes in the BMK:

***a. Safety Component***

(1) Threat Engagement Status. A valid threat track in the Kill Data Store will be paired with a weapon system to engage the valid threat track within thirty seconds of the first instance of that track in the Kill Data Store. If a track in the Kill Data Store has either not been assigned a weapon system to engage it or classified as Deliberate Pass within thirty seconds after entering the track into the Kill Data Store, then the Safety Component will send a warning to user display.

(2) Threat Engagement Toggle. A valid threat track can toggle between deliberate pass and a weapon system pairing only one time. If a track toggles between Deliberate Pass and weapon assigned (i.e., deliberate pass to weapon assigned is one toggle and weapon assigned to deliberate pass is one toggle) more than two times, then a warning will be sent to the user display.

(3) Unassigned Threat Track. A valid threat track in the Track Engagement Data Store will be paired with a weapon system to engage that valid threat track within thirty seconds of the first instance of that track in the Track Engagement Data Store. If an unassigned threat track is not assigned a weapon system to engage it within thirty seconds of the first instance of the Weapon Assignment Computation posting this track into the Track Engagement Data Store, then a warning will be sent to the user display.

(4) Threat Track Identification. A track in the Kill Data Store can only be identified as a Threat. Any other identification for a track in the Kill Data Store is invalid. Safety Component will pull all invalid tracks from the Kill Data Store and send an alert to the user display along with the invalid track data.

(5) Suspect Track Status. Each suspect track must be classified as either threat or benign within thirty seconds of the first instance of the track that is posted in the Suspect Track Data Store. If the track in the Suspect Track Data Store has not been classified as either threat or benign within thirty seconds after the first instance of that track appears in the Suspect Track Data Store, then the Safety Component will send a warning to user display along with the track data.

(6) Suspect Track Update. A suspect track file must be updated within thirty seconds of its previous update after the first instance of the track in the Suspect Track Data Store. If a suspect track file has not been updated within thirty seconds from its previous update, then Safety Component will send suspect track warning to user display along with the track data.

(7) Suspect Track Velocity Status. A suspect track cannot have more than two consecutive velocity updates of less than one kilometer per second. If suspect track file has three consecutive velocity updates of less than one kilometer per second, then Correlation Computation will send suspect track data to user display.

(8) Battlespace Representation Track Update. A track file must be updated within thirty seconds of its previous update after the first instance of the track in the Battlespace Representation Data Store. If a track file (benign or threat) in the Battlespace Representation Data Store has not been updated for thirty seconds or more, then the BMK Safety Component will toggle the track file to Suspect and send track data to Suspect Track Data Store.

#### *b. Findings and Conclusions*

The assertions in Track Safety Component could support the safety of the BMK to coordinate battle-management activities. We asserted the desired behavior with time constraints to ensure that the BMK continues to process track data as specified and provided error-handling specifications for violations of the assertions to support continuity of operations. Thus, we conclude that the assertions support the safety of the BMK as defined for this research.

### 7. Recoverability

In Chapter VI, we defined recoverability as the ease for which a failed system can be restored to operational use. We developed the following set of assertions and error-handling procedures in the Safety Component that monitor the processes in the BMK:

#### *a. Continuity of Operations: Track Processing*

(1) A track must be pulled from Track Data Store within fifteen seconds of its appearance in the Track Data Store.

(2) The first instance of a track file must appear in Battlespace

Representation Data Store within thirty seconds of the first appearance of that valid track data in the Track Data Store.

(3) If (1) and (2) are not true at the same time, then Safety Component will send a reset signal to Track Processing Component. This means that:

(a) Track Processing failed to poll data within fifteen seconds of a given set of track data appearing in the Track Data Store AND

(b) Track Processing failed to discriminate and correlate the track data within thirty seconds of a given set of track data appearing in the Track Data Store.

(c) If both assertion violations occur at the same time, then the Safety Component will reset Track Processing Component.

(4) If (1) and (2) are not true at the same time following a Track Processing reset that occurred within the past sixty seconds, then the Safety Component will deem the BMK as inoperable and direct the transfer of control to another BMK.

***b. Continuity of Operations: Weapon Assignment Processing***

(1) A track must be pulled from Kill Data Store within fifteen seconds of its appearance in the Kill Data Store.

(2) The first instance of a threat track file must appear in Track Engagement Data Store within thirty seconds of the first appearance of that valid threat track data in the Kill Data Store.

(3) If (1) and (2) are not true at the same time, then Safety Component will send reset signal to Weapon Assignment Processing Component.

(4) If (1) and (2) are not true at the same time following a Weapon Assignment Processing reset that occurred within the past sixty seconds, then the Safety Component will deem the BMK as inoperable and direct the transfer of control to another BMK. No other data will be processed by the BMK.

### *c. Findings and Conclusions*

The assertions in the Safety Component could support the recoverability of the BMK to coordinate battle-management activities. We asserted the desired behavior with time constraints to ensure that the BMK continues to process track data as specified and provided error-handling specifications for violations of the assertions to support continuity of operations. Thus, we conclude that the assertions support the recoverability of the BMK as defined for this research.

## I. TECHNICAL CONTRIBUTION

For the fifth technical contribution in this research, we offer that we have reduced the software complexity of the BMK by decoupling the active components in the BMK through the use of data stores. No active component has direct communications with any other active component. As other active components are added to the BMK in the future, the software complexity increases linearly in this construct as compared to the addition of a software module in a monolithic software kernel.

# XIV. DEMONSTRATION OF THE BMK ASSERTIONS

## A. INTRODUCTION

For a slice of the BMK prototype, we will develop temporal assertions from selected natural language assertions of the prototype to specify the timing constraints for the BMK. While numerous types of assertions are available to developers, we chose temporal assertions in this research to keep the scope of the dissertation manageable. We offer that a future research area could be the use of other types of assertions to support the development of controlling software in a system-of-systems.

We will enter the assertions in a model-checking tool that is called DBRover and execute several scenarios to test the assertions. DBRover is a temporal-logic monitoring tool that is based on the TemporalRover code generator. It consists of a graphical user interface for editing temporal assertions, a graphical temporal-logic simulator, and an execution engine. DBRover builds and executes temporal rules for a target program or application. In run-time, DBRover listens for messages from the target application and evaluates corresponding temporal assertions. (N.B.: A complete description of the tool and how to obtain the tool can be found at www.time-rover.com.)

## B. TRACK PROCESSING

We selected the following assertion from Track Processing:

*Track Processing will not poll Track Data Store while discriminating current track data. This assertion checks that Track Processing only tasks a single set of track data to a single instance of the Discrimination Computation and that Track Processing will not send a different set of track data to that instance of the Discrimination Computation until it completes its processing of the current set of track data.*

From the natural language assertion, we define the following terms:

**P** is defined as Track_Data which will be true if Track Processor sends a poll to Track Data Store to get another set of track data.

**Q** is defined as Discrimination Begin which is the point in time that the discrimination component begins its work on the set of track data.

**R** is defined as Discrimination End which is the point in time that the discrimination component ends its work on the set of track data.

We formally define the assertion in metric temporal logics as follows:

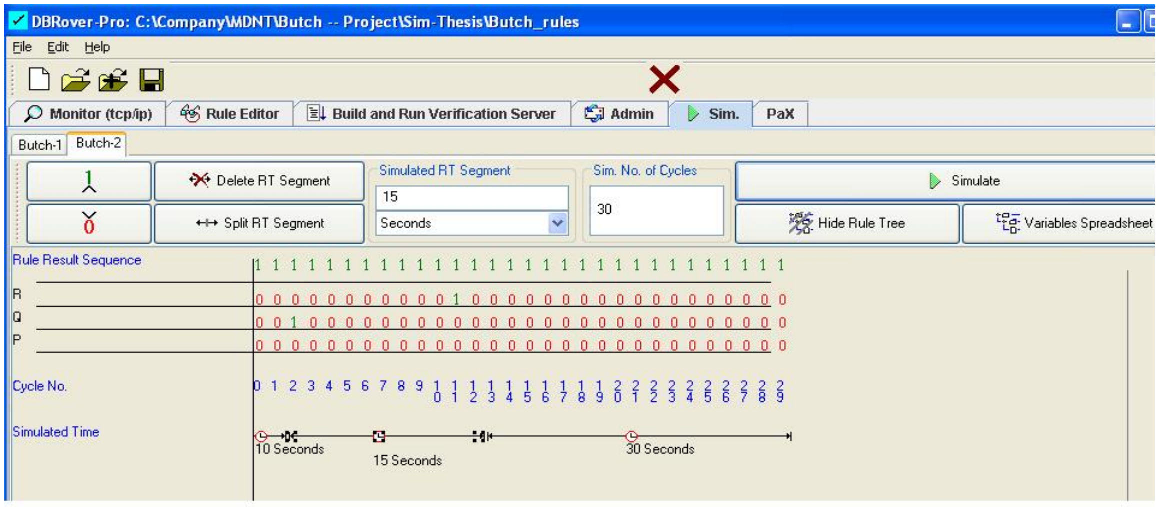
[] ((Q & !R & <>R) -> (!P U_15seconds_R))

This translates as follows:

Always Q is true at this time and R is not true at this time but R will be true at some future time implies that R will become true within fifteen seconds of Q being true and P will remain not true while R is not true; that is, we should not observe an instance of P being true between the time that Q is true and R becomes true.

The screenshots shown below captures are from two scenarios in DBRover. Observe that Q (i.e., Discrimination Begin) occurs at Cycle 2 and that R (i.e., Discrimination End) occurs at Cycle 11. P (i.e., Track_Data) is not true between Q and R. As such, this assertion holds as specified: Track Processing will not poll for new track data while Track Processing is discriminating.

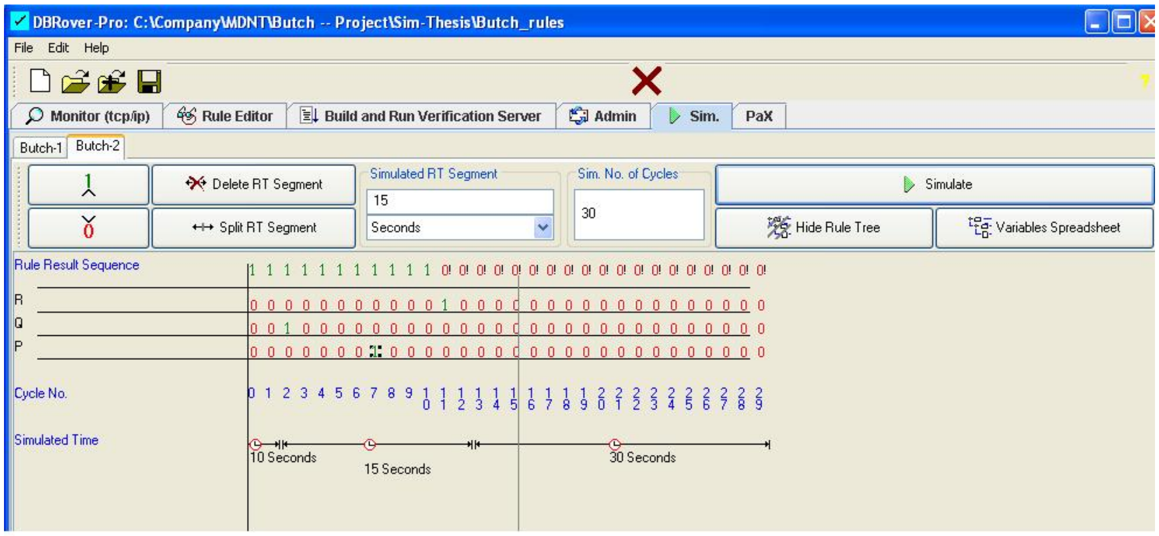
**Figure 18. Track Processing: Scenario 1**



```
Simulated RT Segment: 15 Seconds
Sim. No. of Cycles: 30

Rule Result Sequence  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
R                     0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Q                     0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
P                     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Cycle No.             0 1 2 3 4 5 6 7 8 9 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
                                          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

Simulated Time        10 Seconds   15 Seconds   30 Seconds
```

**success: absence of P between A and R and R happens 15 sec from Q**

Figure 19. Track Processing: Scenario 2



```
Simulated RT Segment: 15 Seconds
Sim. No. of Cycles: 30

Rule Result Sequence  1 1 1 1 1 1 1 1 1 1 1 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0!
R                     0 0 0 0 0 0 0 0 0 0 0 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Q                     0 0 1 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
P                     0 0 0 0 0 0 0 1 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

Cycle No.             0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Simulated Time        10 Seconds   15 Seconds   30 Seconds
```

Observe that Q (i.e., Discrimination Begin) occurs at Cycle 2 and that R (i.e., Discrimination End) occurs at Cycle 11. P (i.e., Track_Data) is true at Cycle 7 which is between Q and R. As such, this assertion does not hold as specified: Track Processing has attempted to poll for new track data before the discrimination work is completed. This is an example of trapping undesired system behavior.

## C. SAFETY COMPONENT

We selected the following natural language assertions from the Safety Component for continuity of operations of Track Processing:

(1) A track must be pulled from Track Data Store within fifteen seconds of its appearance in the Track Data Store.

(2) The first instance of a track file must appear in Battlespace Representation Data Store within thirty seconds of the first appearance of that valid track data in the Track Data Store.

(3) If (1) and (2) are not true at the same time, then Safety Component will send reset signal to Track Processing Component. This means that the Track Processing failed to poll data (1) within fifteen seconds of a given set of track data appearing in the Track Data Store and Track Processing failed to discriminate and correlate the track data within thirty seconds of a given set of track data appearing in the

Track Data Store. If both assertion violations occur at the same time, then the Safety Component will reset Track Processing Component.

(4) If (1) and (2) are not true at the same time following a Track Processing reset that occurred within the past sixty seconds, then the Safety Component will deem the BMK as inoperable and direct the transfer of control to another BMK.

We will develop the formal assertions for each of the above as follows:

***Rule 1.*** Always (TrackDataNew) Implies Eventually_15seconds (TrackPullIsTrue)

*Where:*

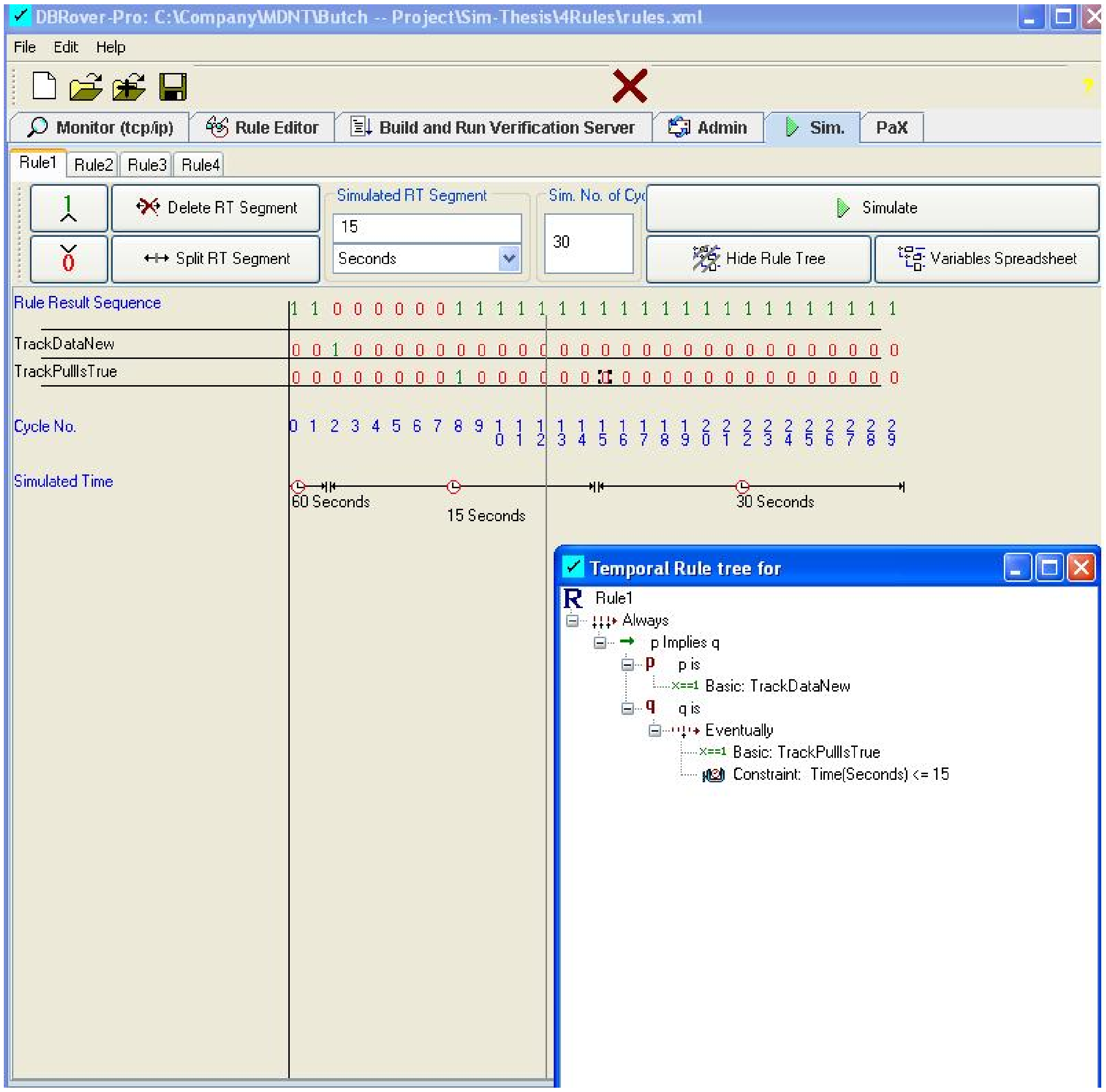
Boolean: TrackDataNew
//true if a new set of track data has been stored in Track Data Store

Boolean: TrackPullIsTrue
//true if Track Processor has pulled the new track data set from Track Data Store

*Assertion:*

[] (TrackDataNew) <>_15seconds (TrackPullIsTrue)

**Figure 20. Safety Component: Rule 1, Scenario 1**



DBRover-Pro: C:\Company\MDNT\Butch -- Project\Sim-Thesis\4Rules\rules.xml

File Edit Help

Monitor (tcp/ip) | Rule Editor | Build and Run Verification Server | Admin | Sim. | PaX

Rule1 | Rule2 | Rule3 | Rule4

Delete RT Segment | Split RT Segment

Simulated RT Segment: 15 Seconds

Sim. No. of Cyc: 30

Simulate | Hide Rule Tree | Variables Spreadsheet

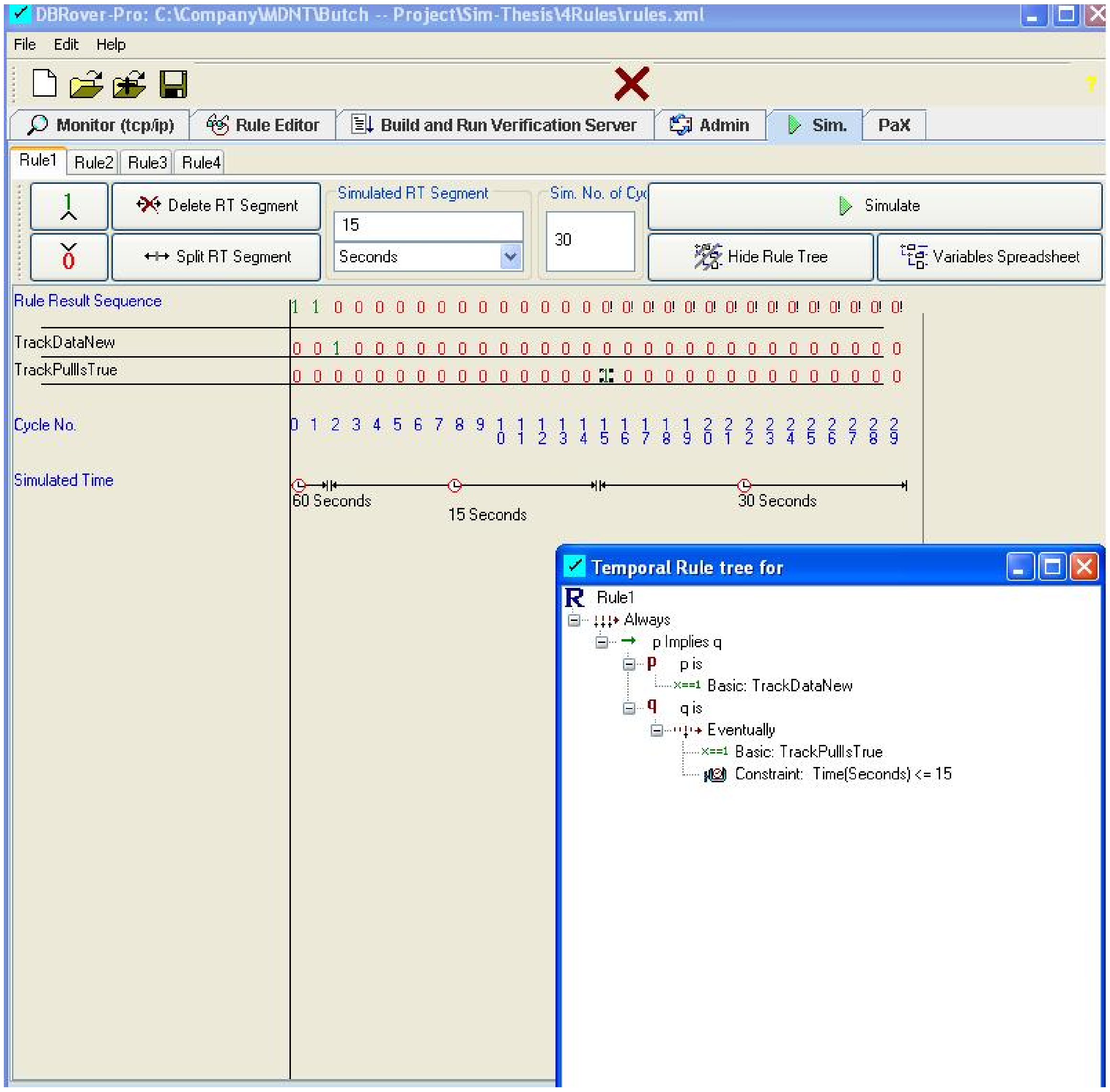
```
Rule Result Sequence  1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackPullIsTrue       0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Cycle No.             0 1 2 3 4 5 6 7 8 9 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
                                          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

Simulated Time        60 Seconds        15 Seconds            30 Seconds
```

Temporal Rule tree for

```
Rule1
  Always
    p Implies q
      p is
        Basic: TrackDataNew
      q is
        Eventually
          Basic: TrackPullIsTrue
          Constraint:  Time(Seconds) <= 15
```

Observe that the assertion (i.e., Rule 1) holds as TrackPullIsTrue becomes true within fifteen seconds of TrackDataNew becoming true; that is, Track Processor pulls the new track data set from Track Data Store within fifteen seconds of the new track data being stored in Track Data Store.

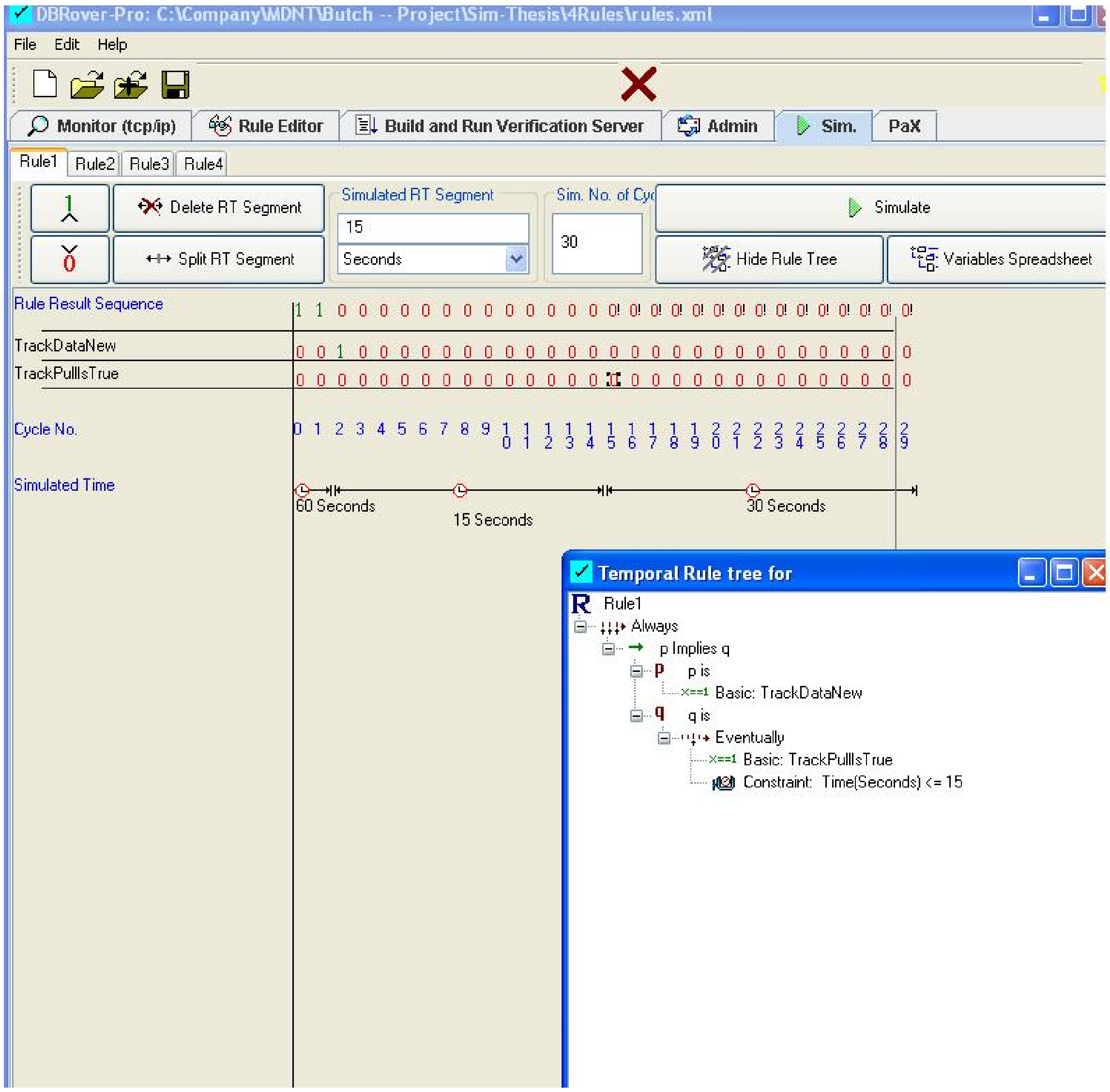
Figure 21. Safety Component: Rule 1, Scenario 2



```
Simulated RT Segment: 15 Seconds
Sim. No. of Cyc: 30

Rule Result Sequence  1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0!
TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackPullIsTrue       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 [illegible] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Cycle No.             0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
Simulated Time        60 Seconds   15 Seconds   30 Seconds

Rule1
  Always
    p Implies q
      p is
        x==1 Basic: TrackDataNew
      q is
        Eventually
          x==1 Basic: TrackPullIsTrue
          Constraint:  Time(Seconds) <= 15
```

Observe that the assertion (i.e., Rule 1) does not hold as TrackPullIsTrue does not become true within fifteen seconds of TrackDataNew becoming true; that is, Track Processor does not pull the new track data set from Track Data Store within fifteen seconds of the new track data being stored in Track Data Store. This is an example of trapping undesired behavior. Developers can devise a recovery scheme from this violation of the assertion.

Figure 22. Safety Component: Rule 1, Scenario 3


DBRover-Pro: C:\Company\MDNT\Butch -- Project\Sim-Thesis\4Rules\rules.xml

File Edit Help

Monitor (tcp/ip) | Rule Editor | Build and Run Verification Server | Admin | Sim. | PaX

Rule1 | Rule2 | Rule3 | Rule4

Delete RT Segment | Split RT Segment | Simulated RT Segment: 15, Seconds | Sim. No. of Cyc: 30 | Simulate | Hide Rule Tree | Variables Spreadsheet

```
Rule Result Sequence  1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0!
TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackPullIsTrue       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Cycle No.             0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Simulated Time        60 Seconds   15 Seconds   30 Seconds
```

Temporal Rule tree for

```
R Rule1
└─ Always
   └─ p Implies q
      ├─ p  p is
      │     └─ x==1 Basic: TrackDataNew
      └─ q  q is
            └─ Eventually
               ├─ x==1 Basic: TrackPullIsTrue
               └─ Constraint: Time(Seconds) <= 15
```

Observe that the assertion (i.e., Rule 1) does not hold as TrackPullIsTrue does not become true within fifteen seconds of TrackDataNew becoming true. (N.B.: In fact, TrackPullIsTrue does not become true in the scenario.) Track Processor does not pull the new track data set from Track Data Store within fifteen seconds of the new track data being stored in Track Data Store. This is an example of trapping undesired behavior. Developers can devise a recovery scheme from this violation of the assertion.

***Rule 2.*** Always (TrackDataNew) And (TrackDataIsValid) Implies Eventually_30seconds (TrackFileIsTrue)

*Where:*

Boolean: TrackDataNew
//true if a new set of track data has been stored in Track Data Store

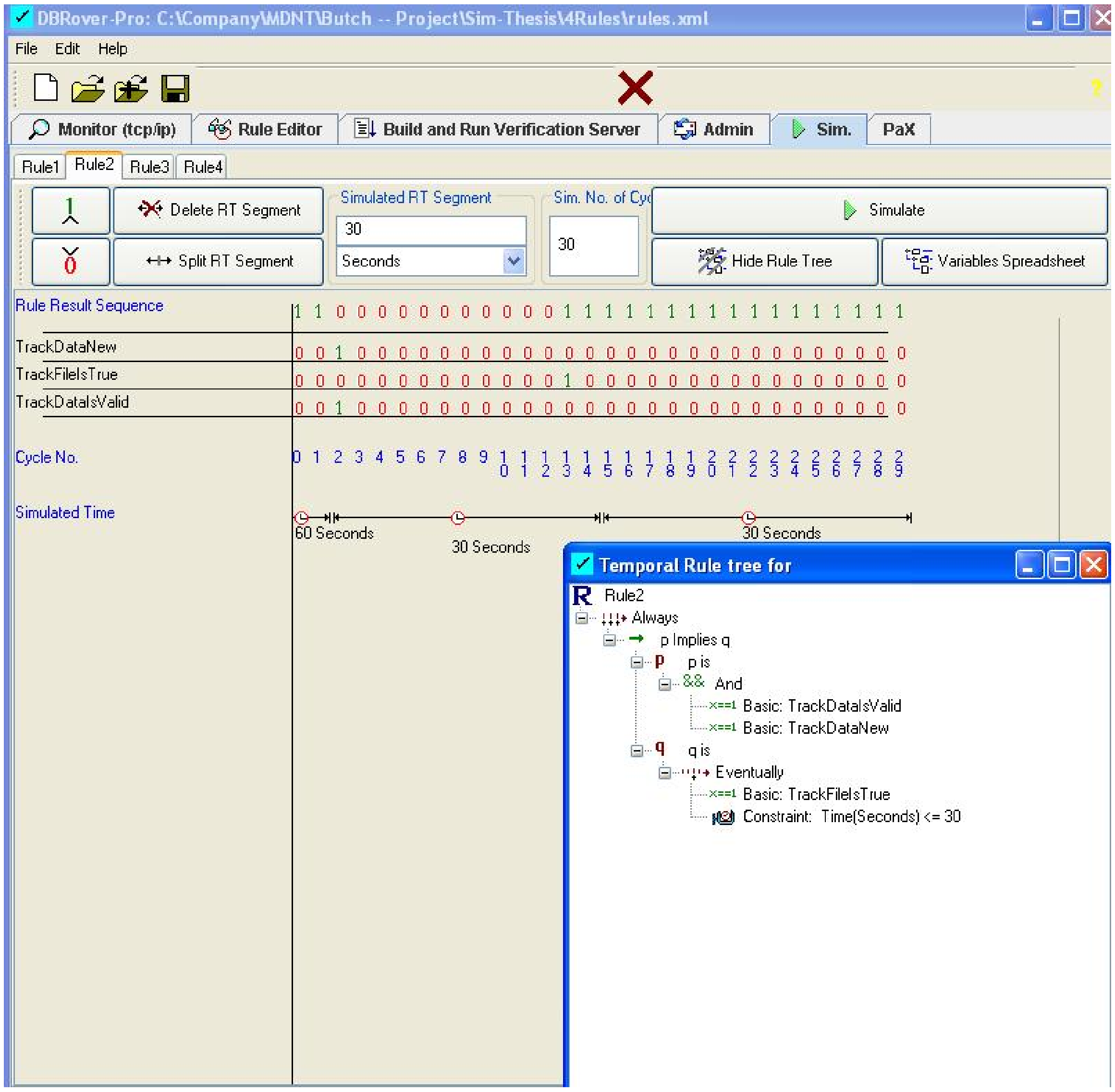
Boolean: TrackDataIsValid
//true if Track Processor has determined that the new track data set is valid

Boolean: TrackFileIsTrue
//true if Track Processor has stored the processed track data in
Battlespace Representation Data Store

*Assertion:*

[] (TrackDataNew) & (TrackDataIsValid) <>_30seconds (TrackFileIsTrue)

Figure 23. Safety Component: Rule 2, Scenario 1



DBRover-Pro: C:\Company\MDNT\Butch -- Project\Sim-Thesis\4Rules\rules.xml

File Edit Help

Monitor (tcp/ip) | Rule Editor | Build and Run Verification Server | Admin | Sim. | PaX

Rule1 | Rule2 | Rule3 | Rule4

Delete RT Segment | Split RT Segment | Simulate | Hide Rule Tree | Variables Spreadsheet

Simulated RT Segment: 30 Seconds

Sim. No. of Cyc: 30

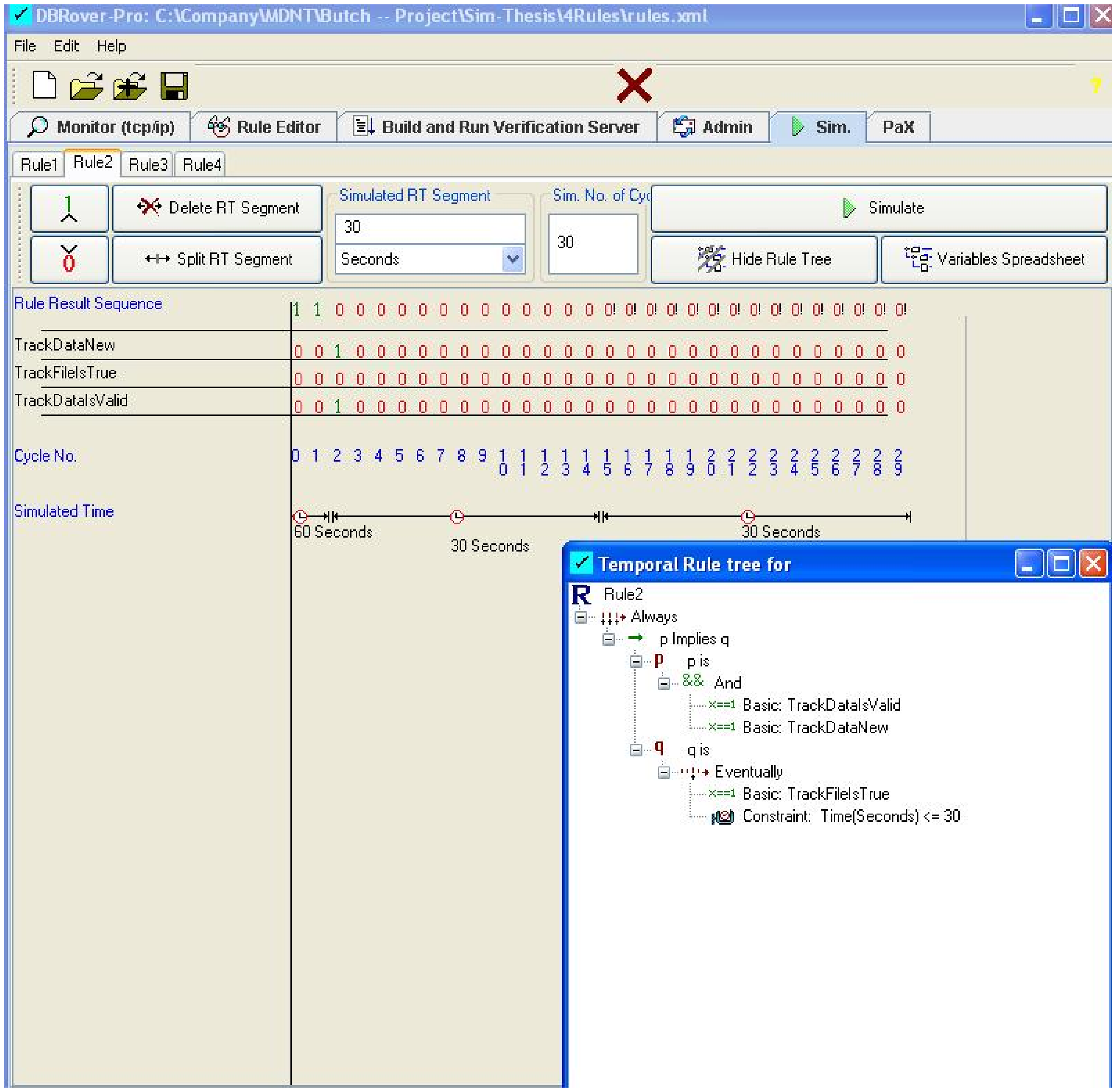
```
Rule Result Sequence  1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackFileIsTrue       0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackDataIsValid      0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Cycle No.             0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Simulated Time        60 Seconds      30 Seconds      30 Seconds
```

Temporal Rule tree for

```
R Rule2
  Always
    p Implies q
      p is
        And
          Basic: TrackDataIsValid
          Basic: TrackDataNew
      q is
        Eventually
          Basic: TrackFileIsTrue
          Constraint: Time(Seconds) <= 30
```

Observe that the assertion (i.e., Rule 2) holds as TrackFileIsTrue becomes true within thirty seconds of TrackDataNew and TrackDataIsValid becoming true; that is, Track Processor stores the processed track data in Battlespace Representation Data Store within thirty seconds of new track data being stored in Track Data Store that Track Processing determines to be valid track data. This is an example of verifying desired system behavior.

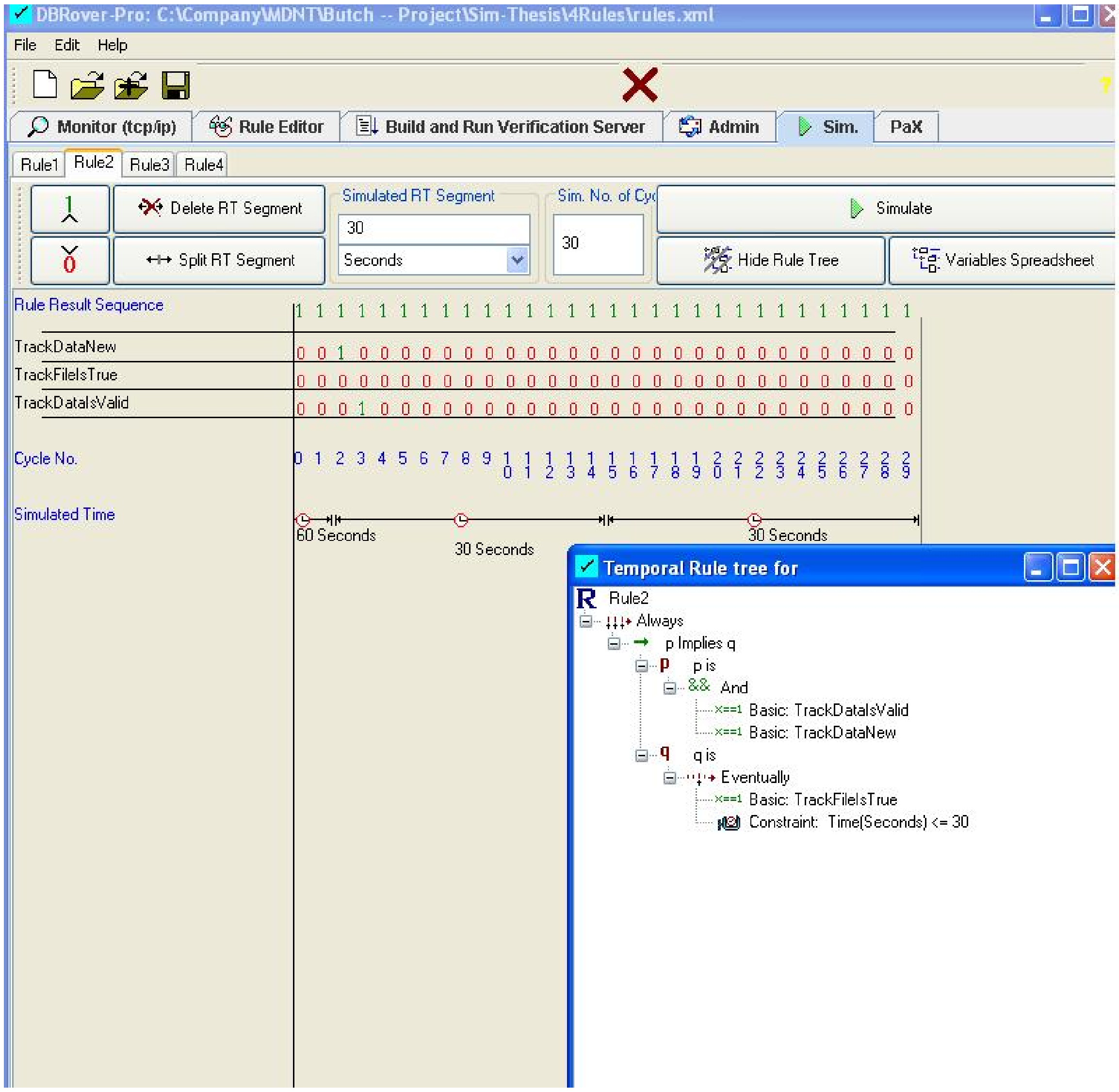
Figure 24. Safety Component: Rule 2, Scenario 2



```
Simulated RT Segment: 30 Seconds
Sim. No. of Cyc: 30

Rule Result Sequence  1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0! 0!
TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackFileIsTrue       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackDataIsValid      0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Cycle No.             0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Simulated Time        60 Seconds   30 Seconds   30 Seconds

Temporal Rule tree for
Rule2
  Always
    p Implies q
      p is
        And
          Basic: TrackDataIsValid
          Basic: TrackDataNew
      q is
        Eventually
          Basic: TrackFileIsTrue
          Constraint:  Time(Seconds) <= 30
```

Observe that the assertion (i.e., Rule 2) does not hold as TrackFileIsTrue does not become true within thirty seconds of TrackDataNew and TrackDataIsValid becoming true. (N.B.: In fact, TrackFileIsTrue does not become true in the scenario.) Track Processor does not store the processed track data in Battlespace Representation Data Store within thirty seconds of new track data being stored in Track Data Store that Track Processing determines to be valid track data. This is an example of trapping undesired behavior. Developers can devise a recovery scheme from this violation of the assertion.

Figure 25. Safety Component: Rule 2, Scenario 3



Simulated RT Segment: 30 Seconds
Sim. No. of Cycles: 30

```
Rule Result Sequence  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackFileIsTrue       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackDataIsValid      0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Cycle No.             0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
Simulated Time        60 Seconds   30 Seconds   30 Seconds
```

```
Rule2
  Always
    p Implies q
      p is
        And
          Basic: TrackDataIsValid
          Basic: TrackDataNew
      q is
        Eventually
          Basic: TrackFileIsTrue
          Constraint:  Time(Seconds) <= 30
```

Observe that the assertion (i.e., Rule 2) holds as TrackDataNew and TrackDataIsValid do not become true simultaneously as defined in the assertion. As such, a check for TrackFileIsTrue is not valid; that is, the arrival of new track data at the Track Data Store and the determination that the new track data is valid does not occur simultaneously. This is an example of trapping undesired behavior. Developers can devise a recovery scheme from this violation of the assertion.

***Rule 3.*** Always (TrackDataNew) & Eventually_15seconds (TrackPullIsTrue) & (TrackDataIsValid) & Not Eventually_30seconds (TrackFileIsTrue)

Implies Eventually_35seconds (Reset==1)

*Where:*

Boolean: TrackDataNew
//true if a new set of track data has been stored in Track Data Store

Boolean: TrackDataIsValid
//true if Track Processor has determined that the new track data set is valid

Boolean: TrackPullIsTrue
//true if Track Processor has pulled the new track data set from Track Data Store

Boolean: TrackFileIsTrue
//true if Track Processor has stored the processed track data in
Battlespace Representation Data Store

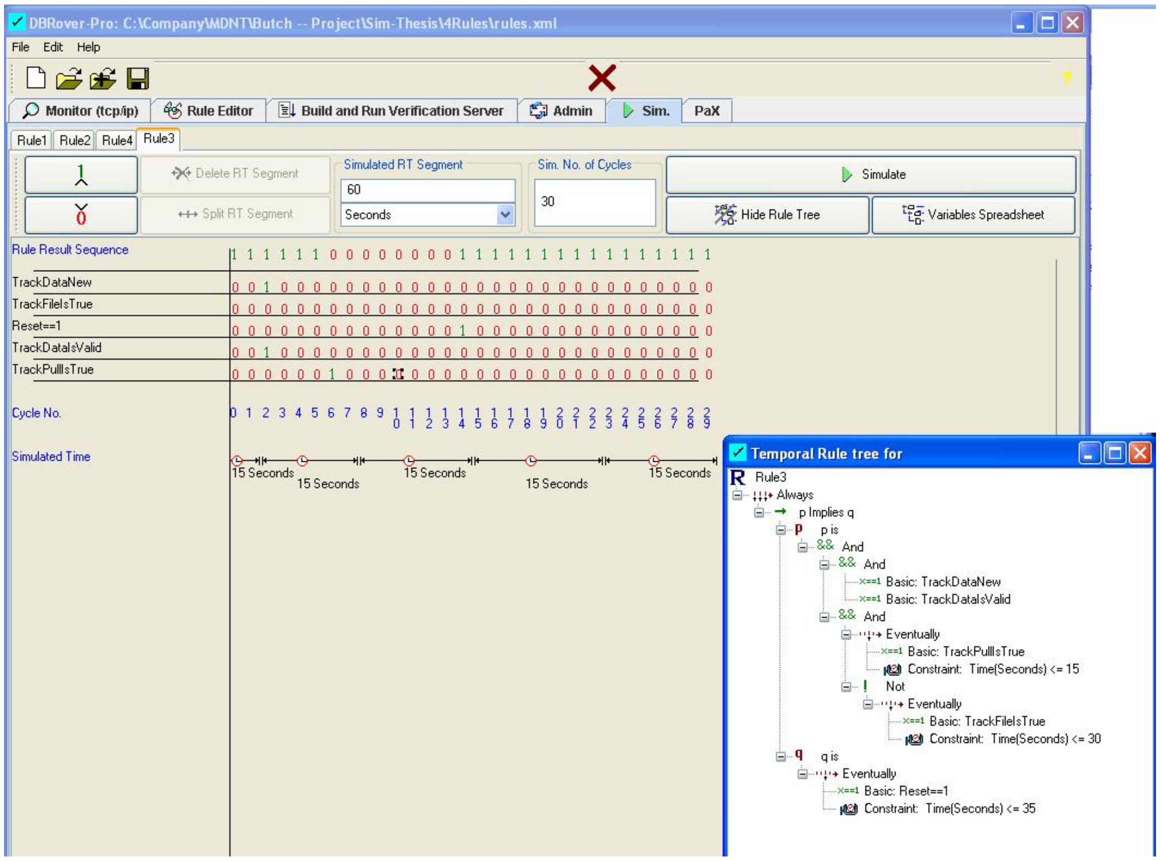
Integer: Reset
//Reset has value of zero until reset is invoked by Safety Component

*Assertion:*

[] {(TrackDataNew) & <>_15seconds (TrackPullIsTrue) & (TrackDataIsValid) &

¬(<>_30seconds (TrackFileIsTrue))} -> {(<>_35seconds (Reset==1)}

**Figure 26. Safety Component: Rule 3, Scenario 1**



DBRover-Pro: C:\Company\MDNT\Butch -- Project\Sim-Thesis\4Rules\rules.xml

File Edit Help

Monitor (tcp/ip) | Rule Editor | Build and Run Verification Server | Admin | Sim. | PaX

Rule1 | Rule2 | Rule4 | Rule3

Delete RT Segment | Split RT Segment

Simulated RT Segment: 60 Seconds

Sim. No. of Cycles: 30

Simulate | Hide Rule Tree | Variables Spreadsheet

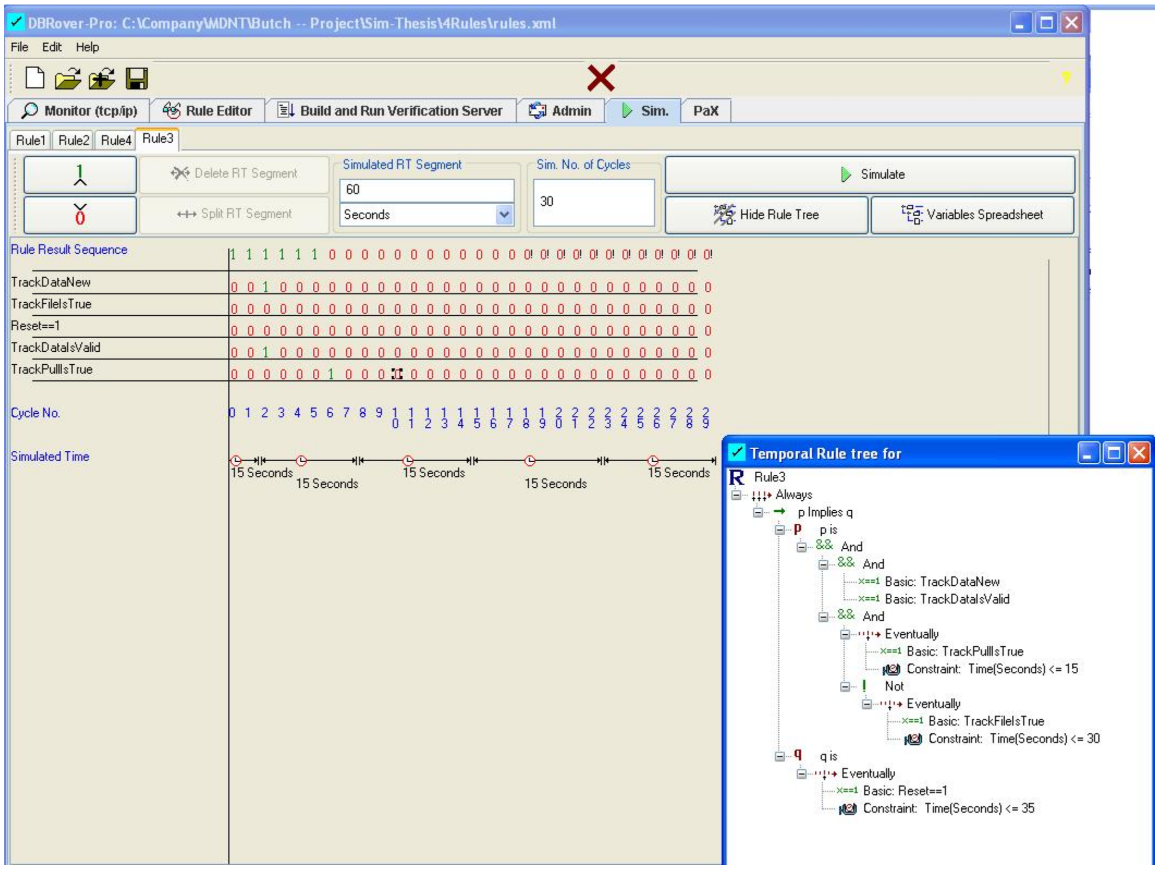
```
Rule Result Sequence  1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackFileIsTrue       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Reset==1              0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackDataIsValid      0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackPullIsTrue       0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Cycle No.             0 1 2 3 4 5 6 7 8 9 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
                                          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

Simulated Time        15 Seconds  15 Seconds  15 Seconds  15 Seconds  15 Seconds
```

Temporal Rule tree for

```
R Rule3
  Always
    p Implies q
      p  p is
        && And
          && And
            x==1 Basic: TrackDataNew
            x==1 Basic: TrackDataIsValid
          && And
            Eventually
              x==1 Basic: TrackPullIsTrue
              Constraint:  Time(Seconds) <= 15
            ! Not
              Eventually
                x==1 Basic: TrackFileIsTrue
                Constraint:  Time(Seconds) <= 30
      q  q is
        Eventually
          x==1 Basic: Reset==1
          Constraint:  Time(Seconds) <= 35
```

Observe that the assertion (i.e., Rule 3) holds as Reset==1 given that TrackFileIsTrue does not become true although the following variables are true: (1) TrackDataNew, (2) eventually (within fifteen seconds) TrackPullIsTrue, and (3) TrackDataIsValid; that is, the Safety Component will observe that Track Processor will reset itself given that Track Processor does not store the processed track data in Battlespace Representation Data Store within thirty seconds of new track data being stored in Track Data Store that Track Processing pulls from Track Data Store and determines to be valid track data. This is an example of verified system behavior.

**Figure 27. Safety Component: Rule 3, Scenario 2**



Simulated RT Segment: 60 Seconds

Sim. No. of Cycles: 30

| Cycle No. | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rule Result Sequence | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0! | 0! | 0! | 0! | 0! | 0! | 0! | 0! | 0! | 0! | 0! | 0! |
| TrackDataNew | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TrackFileIsTrue | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Reset==1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TrackDataIsValid | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TrackPullIsTrue | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | [illegible] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

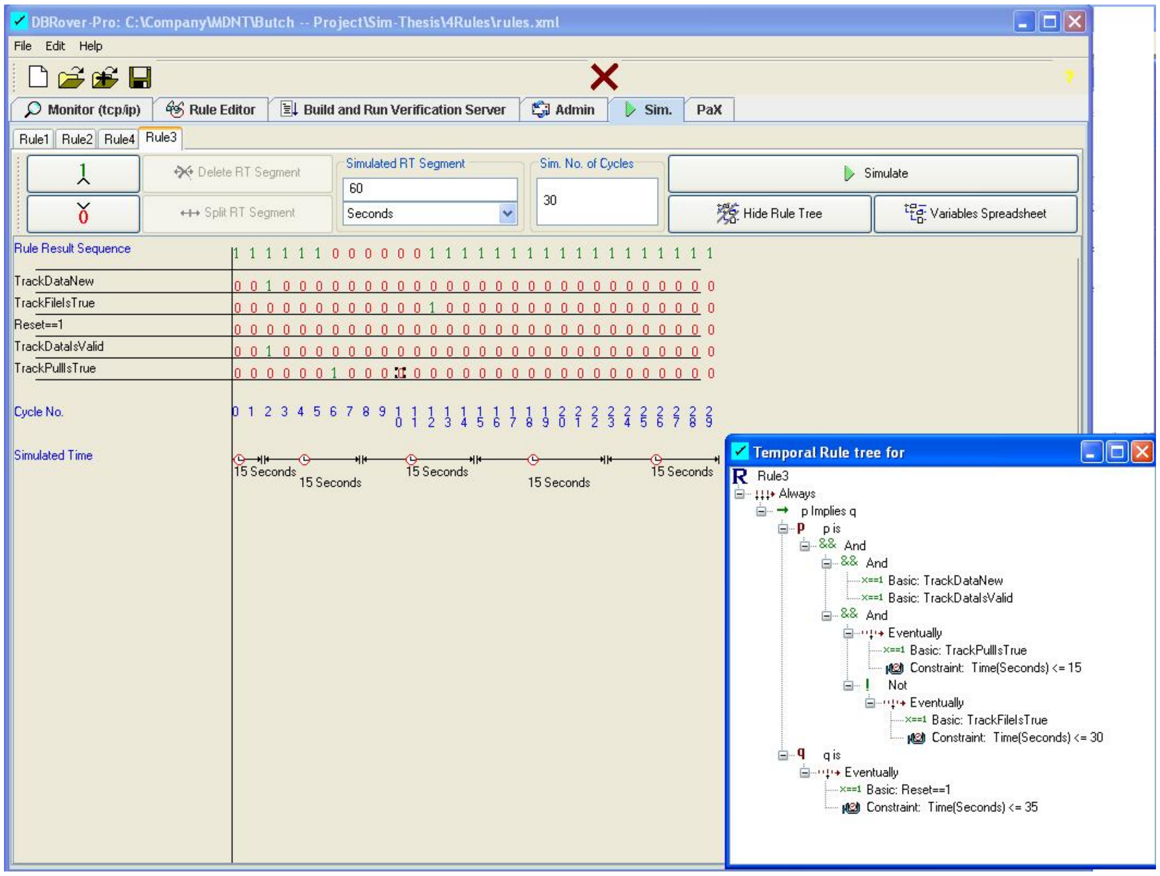
Simulated Time: 15 Seconds, 15 Seconds, 15 Seconds, 15 Seconds, 15 Seconds

Temporal Rule tree for

```
R Rule3
  Always
    p Implies q
      p  p is
        && And
          && And
            Basic: TrackDataNew
            Basic: TrackDataIsValid
          && And
            Eventually
              Basic: TrackPullIsTrue
              Constraint: Time(Seconds) <= 15
            ! Not
              Eventually
                Basic: TrackFileIsTrue
                Constraint: Time(Seconds) <= 30
      q  q is
        Eventually
          Basic: Reset==1
          Constraint: Time(Seconds) <= 35
```

Observe that the assertion (i.e., Rule 3) does not hold as Reset is not set to “1” given that TrackFileIsTrue does not become true and the following variables are true: (1) TrackDataNew, (2) eventually (within fifteen seconds) TrackPullIsTrue, and (3) TrackDataIsValid; that is, the Safety Component will observe Track Processor should have reset itself given that Track Processor does not store the processed track data in Battlespace Representation Data Store within thirty seconds of new track data being stored in Track Data Store that Track Processing pulls from Track Data Store and determines to be valid track data. This is an example of trapping undesired behavior. Developers can devise a recovery scheme from this violation of the assertion.

Figure 28.  Safety Component:  Rule 3, Scenario 3



Simulated RT Segment: 60 Seconds

Sim. No. of Cycles: 30

| Cycle No. | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rule Result Sequence | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| TrackDataNew | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TrackFileIsTrue | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Reset==1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TrackDataIsValid | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TrackPullIsTrue | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | [illegible] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

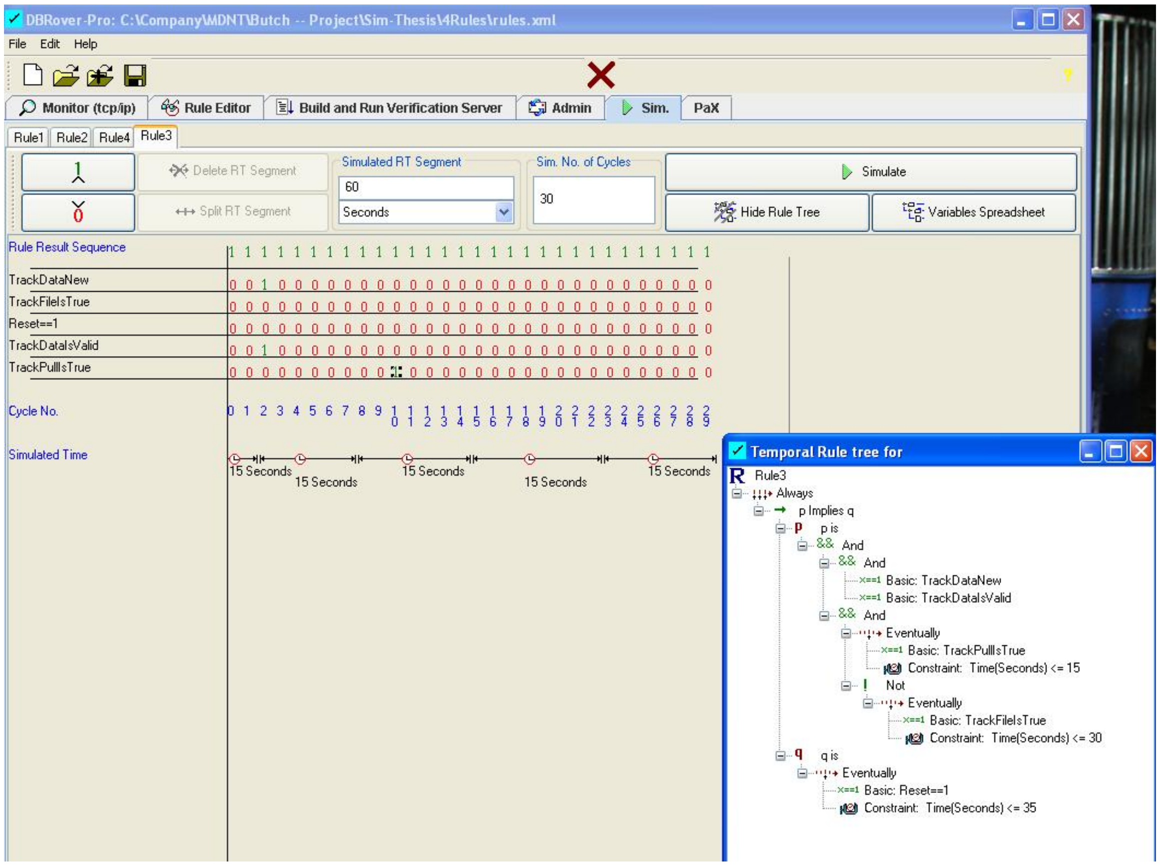
Simulated Time: 15 Seconds, 15 Seconds, 15 Seconds, 15 Seconds, 15 Seconds

Temporal Rule tree for

```
R Rule3
  Always
    p Implies q
      p is
        And
          And
            Basic: TrackDataNew
            Basic: TrackDataIsValid
          And
            Eventually
              Basic: TrackPullIsTrue
              Constraint:  Time(Seconds) <= 15
            Not
              Eventually
                Basic: TrackFileIsTrue
                Constraint:  Time(Seconds) <= 30
      q is
        Eventually
          Basic: Reset==1
          Constraint:  Time(Seconds) <= 35
```

Observe that the assertion (i.e., Rule 3) holds as Reset is not set to "1" given that the following variables are true: (1) TrackDataNew, (2) eventually (within fifteen seconds) TrackPullIsTrue, (3) TrackDataIsValid, and (4) eventually (within 30 seconds) TrackFileIsTrue; that is, the Safety Component will observe Track Processor behaved according to the specification and should not have reset itself given that Track Processor stores the processed track data in Battlespace Representation Data Store within thirty seconds of new track data being stored in Track Data Store that Track Processing pulls from Track Data Store and determines to be valid track data.  This is an example of trapping undesired behavior.  Developers can devise a recovery scheme from this violation of the assertion.

Figure 29. Safety Component: Rule 3, Scenario 4


DBRover-Pro: C:\Company\MDNT\Butch -- Project\Sim-Thesis\4Rules\rules.xml

Simulated RT Segment: 60 Seconds

Sim. No. of Cycles: 30

```
Rule Result Sequence  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackFileIsTrue       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Reset==1              0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackDataIsValid      0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackPullIsTrue       0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Cycle No.             0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Simulated Time        15 Seconds | 15 Seconds | 15 Seconds | 15 Seconds | 15 Seconds
```

Temporal Rule tree for

```
Rule3
  Always
    p Implies q
      p is
        And
          And
            Basic: TrackDataNew
            Basic: TrackDataIsValid
          And
            Eventually
              Basic: TrackPullIsTrue
              Constraint:  Time(Seconds) <= 15
            Not
              Eventually
                Basic: TrackFileIsTrue
                Constraint:  Time(Seconds) <= 30
      q is
        Eventually
          Basic: Reset==1
          Constraint:  Time(Seconds) <= 35
```

Observe that the assertion (i.e., Rule 3) holds as Reset is not set to "1" given that TrackPullIsTrue becomes true in a time frame greater than or equal to fifteen seconds; that is, the Safety Component will observe Track Processor should not have reset itself based on this assertion that requires Track Processor to eventually pull (within fifteen seconds of TrackDataNew becoming true) new track data being stored in Track Data Store. This is an example of verifying desired system behavior.

***Rule 4.*** Always (Reset==0) And sometime in the past (Reset==1) And (TrackDataNew) And (TrackDataIsValid) And Eventually_15seconds (TrackPullIsTrue) & Not Eventually_40seconds (TrackFileIsTrue) Implies Eventually_10seconds (ReplaceBMK)

*Where:*

Boolean: TrackDataNew

//true if a new set of track data has been stored in Track Data Store

Boolean:  TrackDataIsValid
//true if Track Processor has determined that the new track data set is valid

Boolean:  TrackPullIsTrue
//true if Track Processor has pulled the new track data set from Track Data Store

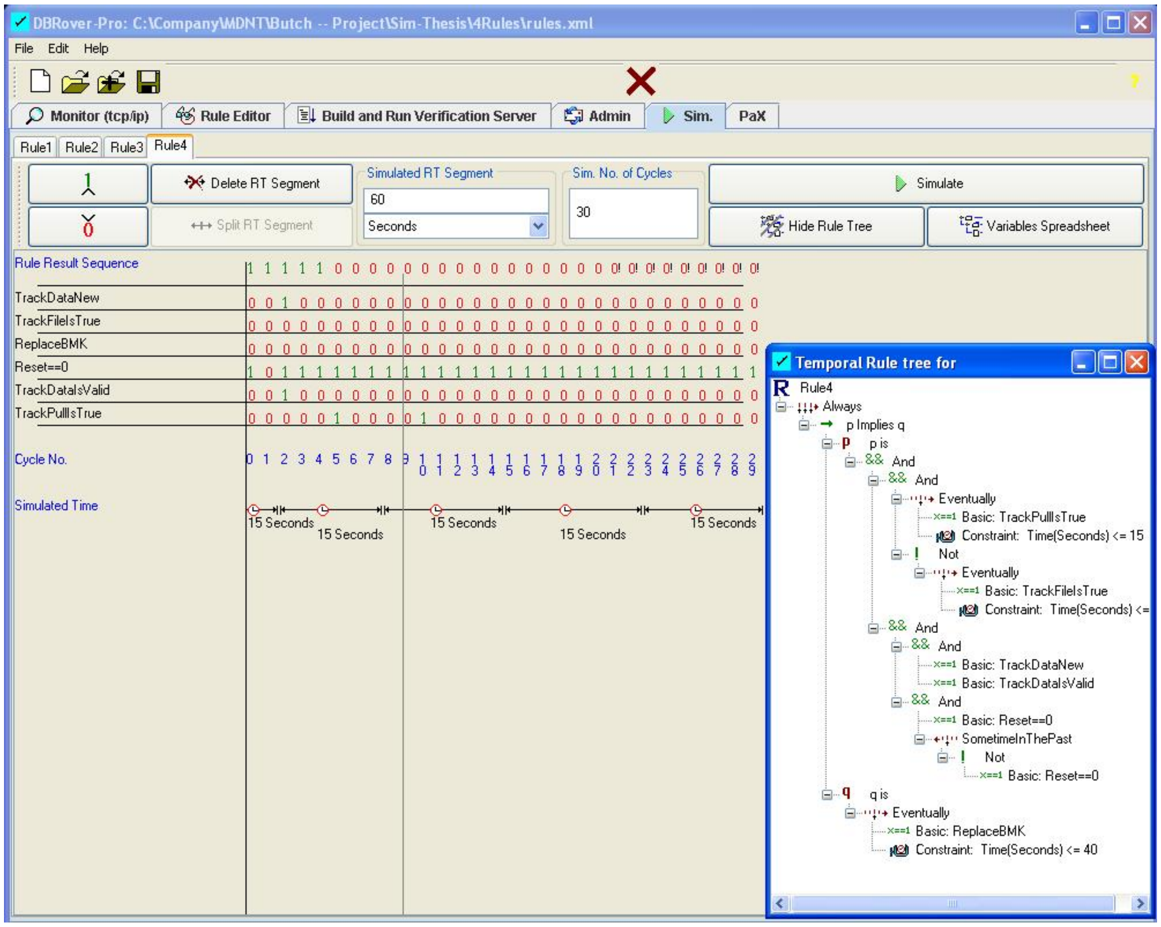
Boolean:  TrackFileIsTrue
//true if Track Processor has stored the processed track data in
  Battlespace Representation Data Store

Integer:  Reset
//Reset has value of zero until reset is invoked by Safety Component

*Assertion:*

[] (Reset==0) & <-> (Reset==1) & (TrackDataNew) & <>_15seconds (TrackPullIsTrue) & (TrackDataIsValid) & ¬(<>_40seconds (TrackFileIsTrue) -> <>_10seconds (ReplaceBMK)

**Figure 30. Safety Component: Rule 4, Scenario 1**



Simulated RT Segment: 60 Seconds

Sim. No. of Cycles: 30

| Cycle No. | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rule Result Sequence | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0! | 0! | 0! | 0! | 0! | 0! | 0! | 0! | 0! |
| TrackDataNew | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TrackFileIsTrue | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ReplaceBMK | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Reset==0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| TrackDataIsValid | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TrackPullIsTrue | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Simulated Time: 15 Seconds, 15 Seconds, 15 Seconds, 15 Seconds, 15 Seconds

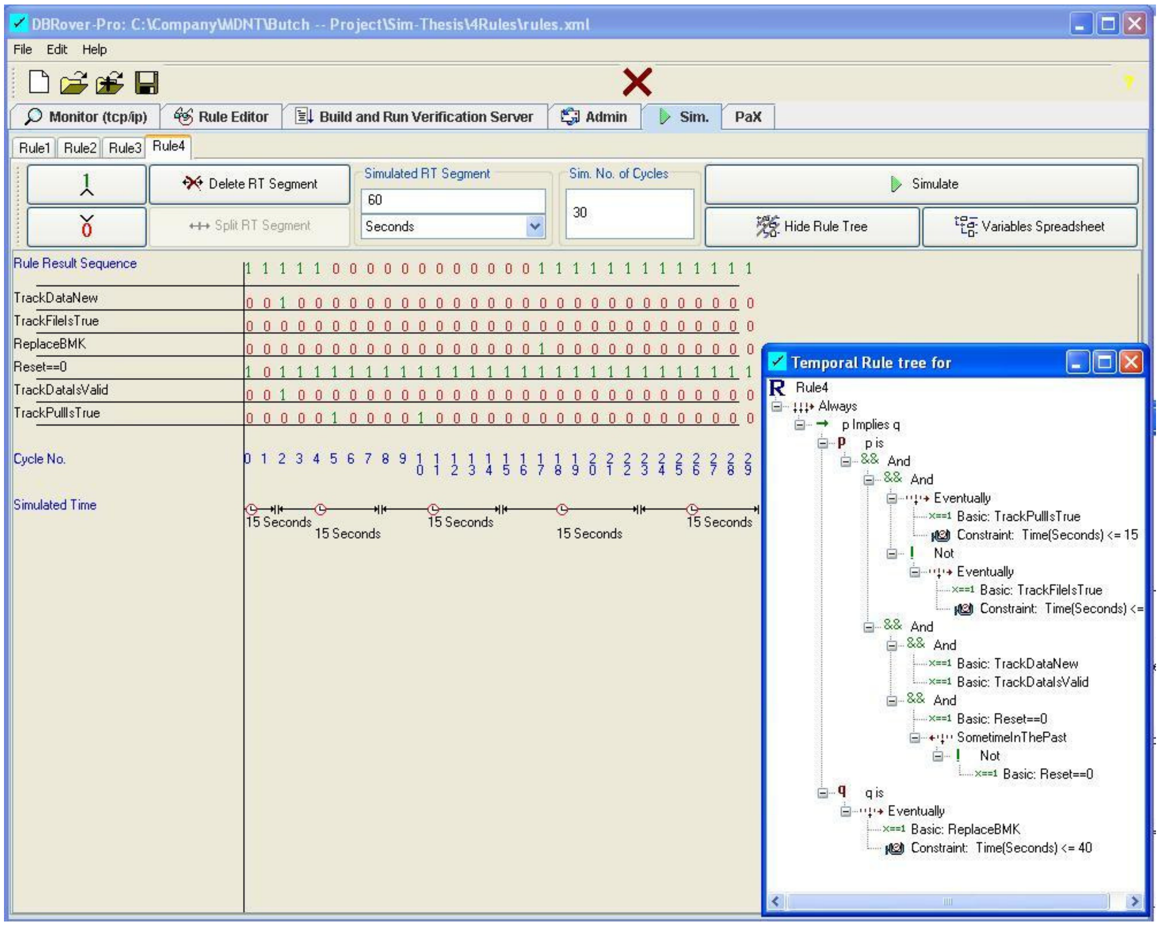
Temporal Rule tree for

```
R Rule4
  Always
    p Implies q
      p  p is
        && And
          && And
            Eventually
              x==1 Basic: TrackPullIsTrue
              Constraint:  Time(Seconds) <= 15
            ! Not
              Eventually
                x==1 Basic: TrackFileIsTrue
                Constraint:  Time(Seconds) <=
          && And
            && And
              x==1 Basic: TrackDataNew
              x==1 Basic: TrackDataIsValid
            && And
              x==1 Basic: Reset==0
              SometimeInThePast
                ! Not
                  x==1 Basic: Reset==0
      q  q is
        Eventually
          x==1 Basic: ReplaceBMK
          Constraint:  Time(Seconds) <= 40
```

Observe that the assertion (i.e., Rule 4) does not hold as ReplaceBMK does not become true although the conditions in the assertion should have driven the BMK to identify itself as not functioning correctly, notifying the other BMKs that is will be transferring control, and shutting down the nonfunctioning BMK. In this scenario, Reset has a previous history of being set to "1" (i.e., reset condition). Although Track Processor pulls the track data set from Track Data Store as specified and determines that the track data set is valid, Track Processor does not store the track data set in Battlespace Representation Data Store. As this is a second violation of the prescribed rules for Track Processor (i.e., Reset was set to "1" in the past), the Safety Component will observe that the BMK failed to shut itself down and transfer control to another BMK. This is an example of trapping undesired behavior

.

**Figure 31. Safety Component: Rule 4, Scenario 2**



DBRover-Pro: C:\Company\MDNT\Butch -- Project\Sim-Thesis\4Rules\rules.xml

Simulated RT Segment: 60 Seconds

Sim. No. of Cycles: 30

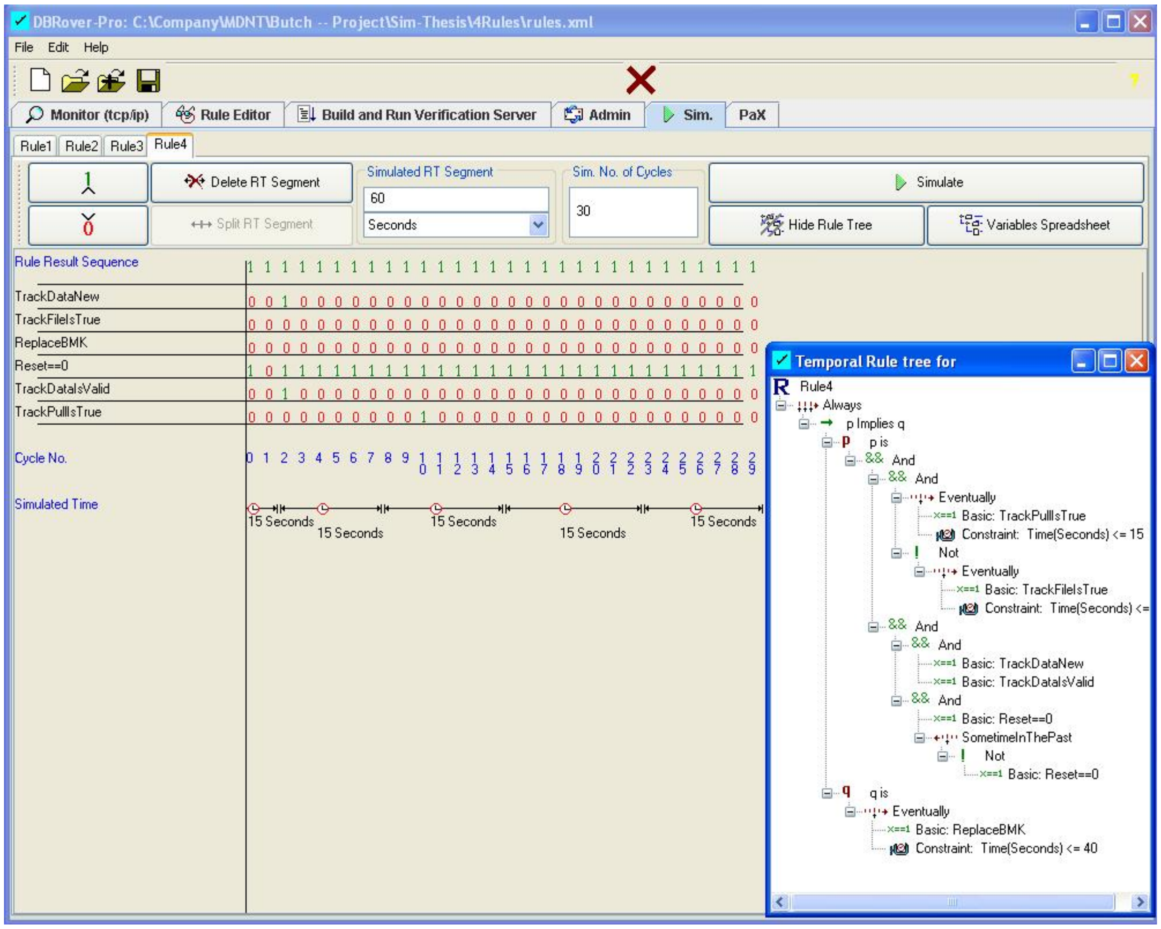
```
Rule Result Sequence  1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackFileIsTrue       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ReplaceBMK            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
Reset==0              1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
TrackDataIsValid      0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackPullIsTrue       0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Cycle No.             0 1 2 3 4 5 6 7 8 9 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
                                          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

Simulated Time        15 Seconds  15 Seconds  15 Seconds  15 Seconds  15 Seconds
```

Temporal Rule tree for

```
R Rule4
  Always
    p Implies q
      p is
        And
          And
            Eventually
              Basic: TrackPullIsTrue
              Constraint: Time(Seconds) <= 15
            Not
              Eventually
                Basic: TrackFileIsTrue
                Constraint: Time(Seconds) <=
          And
            And
              Basic: TrackDataNew
              Basic: TrackDataIsValid
            And
              Basic: Reset==0
              SometimeInThePast
                Not
                  Basic: Reset==0
      q is
        Eventually
          Basic: ReplaceBMK
          Constraint: Time(Seconds) <= 40
```

Observe that the assertion (i.e., Rule 4) holds as ReplaceBMK becomes true as the conditions in the assertion have are telling the BMK to identify itself as not functioning correctly, notify the other BMKs that it will be transferring control, and shutting down. In this scenario, Reset has a previous history of being set to "1" (i.e., reset condition). Although Track Processor pulls the track data set from Track Data Store as specified and determines that the track data set is valid, Track Processor does not store the track data set in Battlespace Representation Data Store. As this is a second violation of the prescribed rules for Track Processor (i.e., Reset was set to "1" in the past), the Safety Component will observe that the BMK has identified its failure to operate as specified, shut itself down, and transfer control to another BMK. This is an example of verifying desired system behavior.

Figure 32. Safety Component: Rule 4, Scenario 3



```
DBRover-Pro: C:\Company\MDNT\Butch -- Project\Sim-Thesis\4Rules\rules.xml
File  Edit  Help
Monitor (tcp/ip) | Rule Editor | Build and Run Verification Server | Admin | Sim. | PaX
Rule1 | Rule2 | Rule3 | Rule4

Delete RT Segment    Simulated RT Segment: 60  Seconds    Sim. No. of Cycles: 30    Simulate
Split RT Segment                                                                    Hide Rule Tree   Variables Spreadsheet

Rule Result Sequence  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
TrackDataNew          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackFileIsTrue       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ReplaceBMK            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Reset==0              1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
TrackDataIsValid      0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TrackPullIsTrue       0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Cycle No.             0 1 2 3 4 5 6 7 8 9 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
                                          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

Simulated Time        15 Seconds  15 Seconds  15 Seconds  15 Seconds  15 Seconds

Temporal Rule tree for
R Rule4
  Always
    p Implies q
      p  p is
        && And
          && And
            Eventually
              x==1 Basic: TrackPullIsTrue
              Constraint:  Time(Seconds) <= 15
            ! Not
              Eventually
                x==1 Basic: TrackFileIsTrue
                Constraint:  Time(Seconds) <=
          && And
            && And
              x==1 Basic: TrackDataNew
              x==1 Basic: TrackDataIsValid
            && And
              x==1 Basic: Reset==0
              SometimeInThePast
                ! Not
                  x==1 Basic: Reset==0
      q  q is
        Eventually
          x==1 Basic: ReplaceBMK
          Constraint:  Time(Seconds) <= 40
```

1. Observe that the assertion (i.e., Rule 4) holds as TrackPullIsTrue becomes true after the time constraint of being true within fifteen seconds of TrackDataNew and TrackDataIsValid being true. ReplaceBMK does not become true since the conditions in the assertion are not present for shutting down the BMK. This is an example of verifying desired system behavior.

## D. TECHNICAL CONTRIBUTIONS

For the sixth contribution of this research, we offered a systematic method for deriving assertions from collaboration diagrams. First, we developed the natural language assertions from reasoning about the collaboration diagram. Second, we transformed the natural language assertions to temporal assertions that we input to a model checker.

For the seventh contribution of this research, we extended component-based software engineering by the advanced use of assertions in the contracts for component interfaces to assert the protocols surrounding the components in a reactive system.

For the eighth contribution of this research, we provided evidence that formal methods can be applied to large, complex system-of-systems developments as follows:

1. Develop an architecture that decouples major portions of the software

2. Separate the software into components that will experience limited modifications over time from those components that might experience significant and frequent modifications

3. Specify the requirements for components and their interface contracts in the form of natural language assertions

4. Transform the natural language assertions of the components and the interface contracts into temporal assertions that can be tested in a model checker

The concepts offered in the BMK prototype and the demonstration address the third of the three research questions from Chapter V. We demonstrated that formal methods can be useful in the design and development of the controlling software in a system-of-systems.

THIS PAGE INTENTIONALLY LEFT BLANK

# XV. CONCLUSIONS

## A. TECHNICAL CONTRIBUTIONS

The following is a summary of the technical contributions in this research:

1. Identification of distributed-system attributes for controlling software in a system-of-systems

2. Identification of real-time attributes for real-time controlling software in a reactive system-of-systems

3. Development of system-of-systems architecture views from system-of-systems view to component view in controlling software

4. Use of kernel in controlling software for system-of-systems to shape dependable behavior of system-of-systems

5. Reduction of software complexity from an exponential factor for a monolithic software program to a component-based construct in which the active components are decoupled by data stores

6. Development of assertions from collaboration diagrams

7. Adoption of CBSE by advanced use of assertions in interface contracts between components to assert protocols surrounding the components in reactive systems

8. Demonstration that formal methods can be applied to large, complex system-of-systems developments

## B. PARNAS' ISSUES

Recall from Chapter I that we cited David Parnas' six major issues with the battle-management software in the SDI program. Recall that we stated that these six issues are not unique to the SDI program but could be extended to other systems-of-systems for which the users desire controlling software (e.g., battle manager) for the system-of-systems. We will review the technical contributions of this research with respect to Parnas' six issues.

Figure 33. Technical Contributions of this Research that Address Parnas' Issues

| PARNAS' ISSUES | TECHNICAL CONTRIBUTIONS OF THIS RESEARCH | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | | | 🟦 | | 🟦 | | 🟦 | 🟦 |
| 2 | 🟦 | | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 |
| 3 | | | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 |
| 4 | | | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 |
| 5 | | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 |
| 6 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 |

🟦 Addresses Parnas' Issue

### 1. Issue One

#### *a. Statement of the Issue*

The battle-management software must identify, track, and direct weapons towards targets whose characteristics may not be known with certainty until the moment of battle. The battle-management software must discriminate the threat objects from decoys and debris.

#### *b. Contributions of this Research*

In this research, we did not address the specific issue of discrimination given that discrimination was not within the scope of this research; however, we offer that Technical Contributions 3, 5, 7, and 8 address supporting aspects to this issue.

Recall that we desired a physical separation of the active components in the BMK that should be stable over time from the passive components in the component layer that are computations, and may be upgraded or replaced frequently. In this research, we defined an architecture that featured a BMK with active components (e.g., Track Processing, Weapon Assignment) and a Component Layer that contained passive components (e.g., Discrimination, Correlation). We proposed the design of the interfaces between active and passive components in accordance with the concept of design-by-contract. We discussed the testing benefits of such a construct and developed an interface with assertions to define the pre-conditions, post-conditions, and invariants of the interface. These assertions can support the development and verification of the desired interface behavior as well as support the development and verification of error-handling procedures for assertions that do not hold true.

This construct can increase the confidence that the computations are correct for planned and unplanned inputs to the passive component along with its interface and test oracle can provide the basis for the development of a test suite for that passive component. This construct can increase that the desired functionality and behavior are correct in a kernel such as the BMK given that an active component connected to its associated passive components with the respective interfaces that are defined with pre-conditions, post-conditions, and invariants along with the test oracle for the active component can provide the basis for the development of a test suite for the kernel.

Developers and maintainers of systems-of-systems can develop and verify the functionality and behavior of the controlling software as outlined in this research. Complex computations can be verified as discussed in the passive component construct. Complex computations can be upgraded and replaced as discussed in the passive component construct.

### 2. Issue Two

#### *a. Statement of the Issue*

Battle-management computing will be accomplished through a network of computers that are connected to sensors and weapons as well as other battle-management computers. The behavior of the battle-management software cannot be predicted with confidence given the actual configuration of weapons, sensors, and battle managers at the moment of battle.

***b. Contribution of this Research***

We offer that Technical Contributions 1,3,4,5,6,7, and 8 address this issue.

We identified attributes to consider for distributed systems that developers might consider in the design of a system-of-systems that could result in predictable distributed behavior of the system-of-systems. We developed architectural views of the system-of-systems that could serve as tools for reasoning about the design of a predictable, dependable system-of-systems.

We defined an architecture that featured a BMK with active components (e.g., Track Processing, Weapon Assignment) and a Component Layer that contained passive components (e.g., Discrimination, Correlation). We proposed the design of the interfaces between active and passive components in accordance with the concept of design-by-contract. We discussed the testing benefits of such a construct and developed an interface with assertions to define the pre-conditions, post-conditions, and invariants of the interface. These assertions can support the development and verification of the desired interface behavior as well as support the development and verification of error-handling procedures for assertions that do not hold true.

This construct can increase the confidence that the computations are correct for planned and unplanned inputs to the passive component, along with its interface and test oracle, can provide the basis for the development of a test suite for that passive component. This construct can increase the level of confidence that the desired functionality and behavior are correct in a kernel such as the BMK given that an active component connected to its associated passive components with the respective interfaces that are defined with pre-conditions, post-conditions, and invariants along with the test oracle for the active component, can provide the basis for the development of a test suite for the kernel.

Developers and maintainers of system-of-systems can develop and verify the functionality and behavior of the controlling software as outlined in this research. Complex computations can be verified as discussed in the passive component construct.

Complex computations can be upgraded and replaced as discussed in the passive component construct.

We discussed the specification of the battle manager with assertions. Additionally, we discussed the verification of the assertions through model checking. We discussed the futility of exhaustive testing of all but the most trivial of systems. We discussed defining, developing, and testing the functionality and behavior of the battle manager with the use of assertions and test oracles.

We constructed a prototype of the BMK by developing natural language assertions to specify the desired functionality and behavior of the battle manager. Included with the prototype was the specification of contracts between the active components of the BMK and the passive components.

We used temporal assertions to specify the timing constraints of the selected natural language assertions that we developed in the prototype. While numerous types of assertions are available to developers, we chose temporal assertions to keep the scope of the dissertation manageable. We defined several scenarios for each assertion and ran the assertions through a temporal-logic model checker. We determined that we had captured the desired functionality and behavior through the use of the assertions. We observed that we could trap undesired behavior through the use of assertions.

In this research, we demonstrated that developers can increase the level of predictable behavior in the controlling software of a system-of-systems by specifying functionality and behavior through the use of assertions, testing of components and interfaces that contain assertions, and verifying the assertions through the use of a model checker. Furthermore, we demonstrated that developers could trap undesired behavior as specified by assertions.

### 3. Issue Three

#### *a. Statement of the Issue*

Developers cannot test the battle-management software under realistic conditions prior to actual use of the software.

### *b. Contribution of this Research*

As the complexity and scope of the system-of-systems increases, it seems that the testing of these large, complex systems becomes increasingly limited with respect to exhaustively testing the system.

We defined an architecture that featured a BMK with active components and a Component Layer that contained passive components. We proposed the design of the interfaces between active and passive components in accordance with the concept of design-by-contract. We discussed the testing benefits of such a construct and developed an interface with assertions to define the pre-conditions, post-conditions, and invariants of the interface. These assertions can support the development and verification of the desired interface behavior as well as support the development and verification of error-handling procedures for assertions that do not hold true.

This construct can increase the confidence that the computations are correct for planned and unplanned inputs to the passive component, along with its interface and test oracle, can provide the basis for the development of a test suite for that passive component. This construct can increase that the desired functionality and behavior are correct in a kernel such as the BMK given that an active component connected to its associated passive components with the respective interfaces that are defined with pre-conditions, post-conditions, and invariants along with the test oracle for the active component can provide the basis for the development of a test suite for the kernel.

We discussed the specification of the battle manager with assertions. Additionally, we discussed the verification of the assertions through model checking. We discussed the futility of exhaustive testing of all but the most trivial of systems. We discussed defining, developing, and testing the functionality and behavior of the battle manager with the use of assertions and test oracles.

We constructed a prototype of the BMK by developing natural language assertions to specify the desired functionality and behavior of the battle manager. Included with the prototype was the specification of contracts between the active components of the BMK and the passive components. To reduce the impact of undesired state behavior of any given active component on any other active component, we decoupled each active component from all other active components in the prototype. We defined data stores to connect the active components and use pulled data from continual polling of specific data stores as a trigger for activities in active components. The decoupling of the active components can enhance the testability of the BMK given that developers can test the active components independently with increased confidence that the composition of the active components will exhibit the desired behavior. Furthermore, we discussed the testing of active components include both black-box testing to test the appropriate outputs for the inputs from a test oracle and white-box testing to determine whether the active components may have exhibited coincidental correctness during the black-box testing.

We used temporal assertions to specify the time constraints of the selected natural language assertions that we developed in the prototype. We defined several scenarios for each assertion and ran the assertions through a temporal-logic model checker. We determined that we had captured the desired functionality and behavior through the use of the assertions. We observed that we could trap undesired behavior through the use of assertions.

In this research, we defined and demonstrated a slice of the test paradigm that can instill increasing levels of confidence as developers test individual components for all possible conditions and continue to compose the system while using the assertions and test oracles to assess system behavior.

### 4. Issue Four

#### *a. Statement of the Issue*

The duration of the defense engagement will be short: it will not allow for either human intervention or debugging the software to overcome software faults at runtime.

### *b. Contribution of this Research*

In this research, we offered techniques for architecture, design, and specification that could produce a dependable system-of-systems at runtime.

We defined an architecture that featured a BMK with active components and a Component Layer that contained passive components. We proposed the design of the interfaces between active and passive components in accordance with the concept of the design-by-contract. We discussed the testing benefits of such a construct and developed an interface with assertions to define the pre-conditions, post-conditions, and invariants of the interface. These assertions can support the development and verification of the desired interface behavior as well as support the development and verification of error-handling procedures for assertions that do not hold true.

This construct can increase the confidence that the computations are correct for planned and unplanned inputs to the passive component along with its interface and test oracle can provide the basis for the development of a test suite for that passive component. This construct can increase confidence level that the desired functionality and behavior are correct in a kernel such as the BMK given that an active component connected to its associated passive components with the respective interfaces that are defined with pre-conditions, post-conditions, and invariants along with the test oracle for the active component can provide the basis for the development of a test suite for the kernel.

We discussed the specification of the battle manager with assertions. Additionally, we discussed the verification of the assertions through model checking. We discussed the futility of exhaustive testing of all but the most trivial of systems. We discussed defining, developing, and testing the functionality and behavior of the battle manager with the use of assertions and test oracles.

We constructed a prototype of the BMK by developing natural language assertions to specify the desired functionality and behavior of the battle manager. Included with the prototype was the specification of contracts between the active components of the BMK and the passive components. To reduce the impact of undesired state behavior of any given active component on any other active component, we decoupled each active component from all other active components in the prototype. We defined data stores to connect the active components and use pulled data from continual polling of specific data stores as a trigger for activities in active components. The decoupling of the active components can enhance the testability of the BMK given that developers can test the active components independently with increased confidence that the composition of the active components will exhibit the desired behavior. Furthermore, we discussed the testing of active components include both black-box testing to test the appropriate outputs for the inputs from a test oracle and white-box testing to determine whether the active components may have exhibited coincidental correctness during the black-box testing.

We used temporal assertions to specify the time constraints of the selected natural language assertions that we developed in the prototype. We defined several scenarios for each assertion and ran the assertions through a temporal-logic model checker. We determined that we had captured the desired functionality and behavior through the use of the assertions. We observed that we could trap undesired behavior through the use of assertions.

We offer that developers and maintainers of system-of-systems can develop and verify the functionality and behavior of the controlling software as outlined in this research. While the techniques that are offered in this research will not eliminate all design and implementation errors, these techniques can reduce the number of critical software faults at runtime and support the handling of unknown conditions at runtime while maintaining continuity of operations.

### 5. Issue Five

#### *a. Statement of the Issue*

The battle-management software will have absolute real-time deadlines for the computation that will consist of periodic processes to include detecting and identifying potential threat missiles, assigning a weapon to engage the threat missile, and providing an assessment of the interceptor-threat missile engagement. Because of the unpredictability of the computational requirements of each process, developers cannot predict the required resources for computation.

#### *b. Contribution of this Research*

While not all system-of-systems will have real-time requirements, developers might consider that non-real-time solutions may not be applicable to reactive systems with periodic deadlines. While the development of a real-time solution for a reactive system-of-systems was beyond the scope of this research, the use of assertions can increase the developer's awareness of the deadlines in the controlling software of a system-of-systems as well as assert specific behavior that correspond to the periodic deadlines of the controlling software.

We defined an architecture that featured a BMK with active components and a Component Layer that contained passive components. We proposed the design of the interfaces between active and passive components in accordance with the concept of the design-by-contract. We discussed the testing benefits of such a construct and developed an interface with assertions to define the pre-conditions, post-conditions, and invariants of the interface. These assertions can support the development and verification of the desired interface behavior as well as support the development and verification of error-handling procedures for assertions that do not hold true.

This construct can increase the confidence that the computations are correct for planned and unplanned inputs to the passive component along with its interface and test oracle can provide the basis for the development of a test suite for that passive component. This construct can increase confidence level that the desired functionality and behavior are correct in a kernel such as the BMK, given that an active component connected to its associated passive components with the respective interfaces that are defined with pre-conditions, post-conditions, and invariants along with the test oracle for the active component, can provide the basis for the development of a test suite for the kernel.

We discussed the specification of the battle manager with assertions. Additionally, we discussed the verification of the assertions through model checking. We discussed the futility of exhaustive testing of all but the most trivial of systems. We discussed defining, developing, and testing the functionality and behavior of the battle manager with the use of assertions and test oracles.

We constructed a prototype of the BMK by developing natural language assertions to specify the desired functionality and behavior of the battle manager. Included with the prototype was the specification of contracts between the active components of the BMK and the passive components. To reduce the impact of undesired state behavior of any given active component on any other active component, we decoupled each active component from all other active components in the prototype. We defined data stores to connect the active components and use pulled data from continual polling of specific data stores as a trigger for activities in active components. The decoupling of the active components can enhance the testability of the BMK given that developers can test the active components independently with increased confidence that the composition of the active components will exhibit the desired behavior. Furthermore, we discussed the testing of active components include both black-box testing to test the appropriate outputs for the inputs from a test oracle and white-box testing to determine whether the active components may have exhibited coincidental correctness during the black-box testing.

We used temporal assertions to specify the time constraints of the selected natural language assertions that we developed in the prototype. We defined several scenarios for each assertion and ran the assertions through a temporal-logic model checker. We determined that we had captured the desired functionality and behavior through the use of the assertions. We observed that we could trap undesired behavior through the use of assertions.

While the scope of this research did not include a solution for a real-time system-of-systems, we recognized the real-time nature of reactive controlling software in a system-of-systems. We demonstrated that we could specify timing constraints with temporal assertions and test these assertions in a model checker.

### 6. Issue Six

#### *a. Statement of the Issue*

The missile defense system will include a large variety of sensors, weapons, and battle-management components for which all will be large, complex software systems. The suite of weapons and sensors will increase in number as the development progresses. The characteristics of these future weapons and sensors are not well defined and will likely remain fluid for many years. Additionally, all weapons and sensors will be subject to change independently of each other. As such, the battle-management software must integrate numerous dynamic software systems to the extent that has never before been achieved.

#### *b. Contribution of this Research*

While the scope of this research did not include interfaces to external systems in a system-of-systems, the BMK architecture described in this research can reduce the impacts of the integration of new systems with the controlling software in a system-of-systems. Rather than direct messaging from a sensor to the BMK, we defined an asynchronous solution that separates incoming sensor data from the BMK through a data store so the BMK is not cognizant of specific sensors. Additionally, we defined an asynchronous solution that separates the BMK from weapon systems by the same data store methodology.

We defined an architecture that featured a BMK with active components and a Component Layer that contained passive components. We proposed the design of the interfaces between active and passive components in accordance with the concept of the design-by-contract. We discussed the testing benefits of such a construct and developed an interface with assertions to define the pre-conditions, post-conditions, and invariants of the interface. These assertions can support the development and verification of the desired interface behavior as well as support the development and verification of error-handling procedures for assertions that do not hold true.

This construct can increase the confidence that the computations are correct for planned and unplanned inputs to the passive component along with its interface and test oracle can provide the basis for the development of a test suite for that passive component. This construct can increase the confidence level that the desired functionality and behavior are correct in a kernel such as the BMK given that an active component connected to its associated passive components with the respective interfaces that are defined with pre-conditions, post-conditions, and invariants along with the test oracle for the active component can provide the basis for the development of a test suite for the kernel.

We discussed the specification of the battle manager with assertions. Additionally, we discussed the verification of the assertions through model checking. We discussed the futility of exhaustive testing of all but the most trivial of systems. We discussed defining, developing, and testing the functionality and behavior of the battle manager with the use of assertions and test oracles.

We constructed a prototype of the BMK by developing natural language assertions to specify the desired functionality and behavior of the battle manager. Included with the prototype was the specification of contracts between the active components of the BMK and the passive components. To reduce the impact of undesired state behavior of any given active component on any other active component, we decoupled each active component from all other active components in the prototype. We defined data stores to connect the active components and use pulled data from continual polling of specific data stores as a trigger for activities in active components. The decoupling of the active components can enhance the testability of the BMK given that developers can test the active components independently with increased confidence that the composition of the active components will exhibit the desired behavior. Furthermore, we discussed the testing of active components include both black-box testing to test the appropriate outputs for the inputs from a test oracle and white-box testing to determine whether the active components may have exhibited coincidental correctness during the black-box testing.

We used temporal assertions to specify the time constraints of the selected natural language assertions that we developed in the prototype. We defined several scenarios for each assertion and ran the assertions through a temporal-logic model checker. We determined that we had captured the desired functionality and behavior through the use of the assertions. We observed that we could trap undesired behavior through the use of assertions.

With respect to data that originates or is sent to these external sources, our research demonstrated that the BMK could have a high level of immunity from the impacts of new sensors and weapons systems in the BMDS. Controlling software in a system-of-systems could realize the same benefit of a high level of immunity from the impacts of integration with new systems. We demonstrated that developers can specify and verify the desired behavior of controlling software through the use of assertions to include the trapping of undesired behavior in the controlling software for a system-of-systems.

## C. CONCLUSIONS

Based on the assessment of the prototype demonstration of a slice of the prototype, the assertions employed in the BMK prototype could support the dependable behavior in the BMK for the seven dependability properties identified in this research: availability, correctness, consistency, reliability, robustness, safety, and recoverability. We specified the desired behavior in the active components of the BMK and we developed assertions that would check the desired behavior at runtime. We assessed the assertions and associated error-handling procedures to determine whether the assertions contributed to the achievement of the desired availability, correctness, consistency, reliability, robustness, safety, and recoverability.

The adherence to design-by-contract concepts could further enhance the dependability of controlling software. Through the development concepts of component-based engineering of the battle manager, we could design the battle manager in such a way that the battle-manager capabilities might be extended by adding components and extending each capability by improving the components associated with that capability.

With respect to Parnas' six issues for the battle manager, the results of this research could contribute to defining, developing, and building a battle manager for the BMDS that is available for operations at any time, operates correctly at all times, traps system faults and returns to operations without impacting the mission of the BMDS, and performs its missions in such a way that no unintended harm to people and protected assets will come from its operations.

Thus, the technical contributions of this research offer evidence that lead us to conclude the following about the questions posed for this research:

1. It is possible to develop a system-of-systems architecture from which we can reason about the controlling software for a system-of-systems.

2. We can realize the controlling software from a system-of-systems architecture through the concepts of component-based software engineering.

3. We can apply formal methods in the design and development of the controlling software for a system-of-systems by specifying the requirements for the software components with assertions and employing a runtime verification tool to verify the desired behavior specified in the assertions.

THIS PAGE INTENTIONALLY LEFT BLANK

# XVI. FUTURE RESEARCH RECOMMENDATIONS

## A. BACKGROUND

As suggested in this research, the system-of-systems problem space is largely unknown to software developers. By all indications, developers of systems-of-systems seem to be experiencing limited success. This research is a first step towards our mastery of the development of dependable systems-of-systems. There are many paths that researchers might travel in the study of the system-of-systems problem. We offer ten recommendations for future research in this area.

## B. FUTURE RESEARCH RECOMMENDATIONS

### 1. Safety Kernel for a System-of-Systems

Each system in the system-of-systems should consider the system-safety concerns for that system; however, the system-of-systems developer should consider the system-safety aspects for the entire system-of-systems. This may require the design of safety-monitoring software as well as safety-related fault-handling software. The considerations in this area might include the desired functionality, specifications, control, distributed system behavior, and real-time system behavior. A future research consideration might be the design and development of a safety kernel for the system-of-systems. A further consideration might be the application of formal methods in the specification of the safety kernel.

### 2. System-of-Systems Operation in Multiple Configurations

System-of-systems can operate in multiple configurations at runtime. A future research consideration might be the development of architecture, design, and specification techniques to ensure the desired level of dependability in all configurations and reducing the potential negative impacts of configuration modifications at runtime.

### 3. Trade-Offs of Assertions Left in System-of-Systems at Runtime Versus Assertions Used in Development

Assertions can be used for testing software, debugging code, and armor-plating software. A future research consideration might be the study of tradeoffs to support the determination of whether to leave the assertions in the software of a system-of-systems for runtime execution.

#### 4. Asserting Control in System-of-Systems

Developers can design and specify the control over a system-of-systems in various ways. A future research topic might be development of techniques for asserting control in a system-of-systems to include the integration of traditional control theory with software engineering.

#### 5. Development Metrics for a System-of-Systems

It is difficult to measure and monitor progress towards achieving architecture and design goals in a system-of-systems given the independent development and life-cycle activities of each system in the system-of-systems. A future research topic might be the development of metrics that reflect the progress towards achieving the functional goals as well as the dependability goals of a system-of-systems development.

#### 6. Properties of a Dependable System-of-Systems and Network-Centric Warfare Solution

For this research, we defined a dependable system as one that provides the appropriate levels of correctness and robustness in accomplishing its mission while demonstrating the appropriate levels of availability, consistency, reliability, safety, and recoverability. We selected these seven properties as these seven properties could be a minimum set of properties for a dependable system-of-systems. Other properties may be applicable to a system-of-systems and a network-centric warfare construct. We would recommend that future research be conducted in the area of dependable system-of-systems and dependable network-centric warfare solutions. Development organizations sometimes focus on the functional aspects of a solution while neglecting to consider the dependability properties of systems that are critical to successful operations.

#### 7. Distributed Considerations for a Dependable System-of-Systems and Network-Centric Warfare Solution

The Department of Defense has stated initiatives for system-of-systems and network-centric warfare solutions. The initiatives seem to have noble goals; however, it is not clear that acquisition organizations have considered the distributed properties that these systems must exhibit from a dependability perspective. Much seems to be left to trust with respect to the degree of dependability in the system-of-systems and network-centric solution. We propose the specific design and development of a distributed system for a system-of-systems and network-centric solutions as a future research topic.

### 8. Real-Time In Distributed Environment

In the system-of-systems and network-centric warfare environments, there will be real-time constraints in the operational battlespace. Acquisition organizations continue to purchase more and faster processors that can mask the real-time problem that has yet to be solved for systems-of-systems and network-centric warfare solutions. We propose the real-time nature of system-of-systems and network-centric warfare solutions as a future research topic.

### 9. Interface Considerations For A Dependable System-Of-Systems And Network-Centric Warfare Solution

To support dependable system-of-systems and network-centric warfare solutions, interfaces should provide the appropriate services required for functionality while maintaining the properties of dependability. For example, the BMDS Battle Manager services include: (1) sending observed tracks and features from the sensors to the battle manager, (2) sending weapon assignments from the battle manager to the weapons, (3) sending health-and-status information from the sensors and weapons to the battle manger, (4) sending command-and-control parameters from the C2 function to the battle manager, and (5) sending situational awareness information from the battle manager to C2 displays. We recommend external interfaces as a potential research topic for dependable systems-of-systems and network-centric solutions with respect to developing and achieving the desired level of dependability.

### 10. Testing Considerations For A Dependable System-Of-Systems And Network-Centric Warfare Solution

The Department of Defense has stated initiatives for systems-of-systems and network-centric warfare solutions. Testing of these solutions has focused on a subset of the entire functionality of the system-of-systems and network-centric warfare solutions. Test solutions as these systems evolve are limited and do not consider the dependability aspects. Testing the entire system can be expensive. For example, operational flight testing of these systems can exceed $100,000,000 for live flight tests such as in the

BMDS development. We propose the testing for the evolving functionality and dependability of a system-of-systems and network-centric warfare solution as a future research topic.

# APPENDIX A. GLOSSARY

**Acquisition:** The process in which the Department of Defense obtains materiel solutions to identified problems in mission need statements.

**Active component:** A component that will execute based on external conditions and a defined set of rules.

**Architecture:** the collection of logical and physical views, constraints, and decisions that define the external properties of a system and provide a shared understanding of the system design to the development team and the intended user of the system**.**

**Architectural style:** a defined grouping of subsystems and connector types along with the defined constraints that are used to realize message transport.

**Assertion:** A predicate expression whose value is either true or false.

**Availability:** The probability that a system is operating correctly and is ready to perform its desired functions.

**Battle management:** The decisions and actions executed in direct response to the activities of enemy forces in support of the Joint Chiefs of Staff's precision engagement concept.

**Battlespace:** All aspects of air, surface, and subsurface, land, space, and the electromagnetic spectrum that encompass the area of influence and area of interest. (NWP 1-02)

**Battlespace constraints:** The forces, facilities, and other features that serve to restrain, restrict, or prevent the implementation of proposed military improvements in the defined battlespace. Constraints may include natural and physical forces, doctrine, potential adversary threats, and environmental features.

**Bistatic radar:** A radar that radar operates with separated transmitting and receiving antennas.

**Black box testing:** A software testing technique whereby explicit knowledge of the internal workings of the component being tested are not known and the outputs are examined with respect to the inputs.

**Capability:** The ability to perform a course of action or sequence of activities leading to a desired outcome.

**Capability-based acquisition:** The process of identifying system capabilities in terms of specifications and acquiring the software applications, hardware, and information services to support these desired capabilities in an integrated environment.

**Chain of command:** The succession of commanding officers from a superior to a subordinate through which command is exercised. (Joint Pub 1-02)

**Coalition:** An ad hoc arrangement between two or more nations for common action.

**Coincidental correctness:** A characteristic of a system that can produce the correct outputs for a specific inputs as defined by the system specifications; however, incorrectly implemented software in the system does not always impact the final output of the system. That is, a system that is said to demonstrate coincidental correctness does the right thing some of the time.

**Combatant command:** One of the unified or specified combatant commands established by the President. (Joint Pub 1-02)

**Combatant command (command authority):** Non-transferable command authority established by title 10, United States Code, section 164, exercised only by commanders of unified or specified combatant commands unless otherwise directed by the President or the Secretary of Defense. Combatant command (command authority) is the authority of a combatant commander to perform those functions of command over assigned forces involving organizing and employing commands and forces, assigning tasks, designating objectives, and giving authoritative direction over all aspects of military operations, joint training, and logistics necessary to accomplish the missions assigned to the command. Also called **COCOM**. (Joint Pub 1-02)

**Combatant commander:** A commander in chief of one of the unified or specified combatant commands established by the President. (Joint Pub 1-02)

**Combat information:** Unevaluated data gathered by or provided directly to the tactical commander which, due to its highly perishable nature or the criticality of the situation, cannot be processed into tactical intelligence in time to satisfy the user's tactical intelligence requirements. (Joint Pub 1-02)

**Command:** 1. The authority that a commander in the Armed Forces lawfully exercises over subordinates by virtue of rank or assignment. Command includes the authority and responsibility for effectively using available resources and for planning the employment of, organizing, directing, coordinating, and controlling military forces for the accomplishment of assigned missions. It also includes responsibility for health, welfare, morale, and discipline of assigned personnel. 2. An order given by a commander; that is, the will of the commander expressed for the purpose of bringing about a particular action. 3. A unit or units, an organization, or an area under the command of one individual. (Joint Pub 1-02)

**Command and control:** The exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission. Command and control functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission. (JCS/J7/Joint Doctrine Division memo dated 20 Oct 94)

**Command and control system:** The facilities, equipment, communications, procedures, and personnel essential to a commander for planning, directing, and controlling operations of assigned forces pursuant to the missions assigned. (Joint Pub 1-02)

**Command, Control, Communications, and Computer Systems** (C4 Systems). Integrated systems of doctrine, procedures, organizational structures, personnel, equipment, facilities, and communications designed to support a commander’s exercise of command and control through all phases of the operational continuum.

**Command and control warfare:** The integrated use of operations security (OPSEC), military deception, psychological operations (PSYOP), electronic warfare (EW), and physical destruction, mutually supported by intelligence, to deny information to, influence, degrade, or destroy adversary command and control capabilities, while protecting friendly command and control capabilities against such actions. Command and control warfare applies across the operational continuum and at all levels of conflict. Also called **C2W**. C2W is both offensive and defensive: a. counter-C2-To prevent effective C2 of adversary forces by denying information to, influencing, degrading, or destroying the adversary C2 system. b. C2-protection-To maintain effective command and control of own forces by turning to friendly advantage or negating adversary efforts to deny information to, influence, degrade, or destroy the friendly C2 system. (Joint Pub 1-02)

**Component:** A software unit of composition with contractually specified interfaces and explicit context dependencies.

**Component-based engineering:** The design and development of a system through the assembly of components which can be developed independently of the system.

**Configurable component:** A component which can accept parameters from an external source such as a sensor or user.

**Control:** Authority which may be less than full command exercised by a commander over part of the activities of subordinate or other organizations. (Joint Pub 1-02).

**Correctness:** A characteristic of a system that precisely exhibits predictable behavior at all times as defined by the system specifications. That is, a system that is said to demonstrate correctness does the right thing all the time.

**Correlation:** The capability to associate one track with one sensed object

**Consistency:** The property that invariants will always hold true in the system.

**Crisis Action Planning**: The time-sensitive planning for the deployment, employment, and sustainment of assigned and allocated forces and resources that occurs in response to a situation that may result in actual military operations. Crisis action planners base their plan on the circumstances that exist at the time planning occurs. Also called CAP (Joint Pub 1-02)

**Critical Section:** Shared resources for which multiple processes can access during runtime. The software code of a critical section must execute without interruption. Otherwise, the system software could experience deadlock and race conditions.

**Data:** A representation of individual facts, concepts, or instructions in a manner suitable for communication, interpretation, or processing by humans or by automatic means. [IEEE]

**Deadlock:** The condition in which a process waits indefinitely for conditions that will never be satisfied. For deadlock to occur, all of the following four conditions must be true: (1) processes claim exclusive control of shared resources, (2) processes hold shared resources while waiting for other shared resources to be released, (3) processes cannot be directed to release shared resources, and (4) a circular waiting condition exists for the release of shared resources. [30]

**Deliberate Planning:** A planning process for the deployment and employment of apportioned forces and resources that occurs in response to a hypothetical situation. Deliberate planners rely heavily on assumptions regarding the circumstances that will exist when the plan is executed. (Joint Pub 1-02)

**Dependable system:** One that provides the appropriate levels of correctness and robustness in accomplishing its mission while demonstrating the appropriate levels of availability, consistency, reliability, safety, and recoverability.

**Design:** The details of planned implementation which are defined, structured, and constrained by the architecture

**Discrimination:** The capability to distinguish a threat object from benign objects such as debris, chaff, countermeasures, and satellites.

**Distributed component:** A component that executes across multiple processors.
**Distributed system:** A system that has multiple processors that are connected by a communications structure.

**Domain analysis:** The process of identifying and formalizing constraints on input, state, and output values.

**Dominant maneuver:** The ability of joint forces to gain positional advantage with decisive speed and overwhelming operational tempo in the achievement of assigned military tasks.

**Fail hard:** A system condition in which either a hardware or software failure causes the entire system to stop working.

**Fail soft:** A system condition in which either a hardware or software failure causes the termination of nonessential processing. Systems in fail-soft mode can still provide partial operational-capability.

**Failure:** The inability of a system or component to perform a required function within specified limits.

**Fire-control solution:** The collection of calculations by a weapon system to determine the point of intercept, launch angle, and time of launch of an interceptor.

**Fault:** An incorrect statement, step, process, or data definition in a software program.

**Fly-out time:** The time difference from the time of launch of the interceptor to the time of engagement of the ballistic missile threat

**Focused logistics:** The ability to provide the joint force the right personnel, equipment, and supplies in the right place, at the right time, and in the right quantity, across the full range of military operations.

**Formal method:** A method that precisely describes a specification in mathematical terms to make possible the verification of the specification in the requirements phase as well as the testing phase of system development.

**Formal specification:** The precise definition of a system behavior that is typically expressed in mathematical terms.

**Framework:** an abstracted view of a complex entity or process

**Full dimensional protection:** The ability of the joint force to protect its personnel and other assets required to decisively execute assigned tasks.

**Functional model:** A system abstraction that contains the set of observations, modeling data, pre-conditions, post-conditions, invariants, boundary conditions, and algorithms that describe the physical system.

**Information:** The meaning that a human assigns to data by means of the known conventions used in their representation. (Joint Pub 1-02)

**Intelligence:** The product resulting from the collection, processing, integration, analysis, evaluation, and interpretation of available information concerning foreign countries or areas. (Joint Pub 1-02)

**Interoperability:** The ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces and to use the services so exchanged to enable them to operate effectively together. (Joint Pub 1-02)

**Invariant:** A property that holds true under any transformation in the system.

**Joint:** Connotes activities, operations, organizations, etc., in which elements of two or more Military Departments participate. (Joint Pub 1-02)

**Joint force:** A general term applied to a force composed of significant elements, assigned or attached, of two or more Military Departments, operating under a single joint force commander. (JCS/J7/Joint Doctrine Division memo dated 20 Oct 94)

**Joint task force:** A joint force that is constituted and so designated by the Secretary of Defense, a combatant commander, a sub-unified commander, or an existing joint task force commander. (JCS/J7/Joint Doctrine Division memo dated 20 Oct 94)

**Keep-out altitude:** The keep-out altitude for ballistic missile defense is the lowest altitude above an area on the surface of the Earth for which an engagement must occur to minimize the ground effects of debris from the engagement. The keep-out altitude is important because the debris from the resultant engagement will fall back to Earth, and it may contain nuclear, chemical, or biological agents that can negatively impact humans and assets in the volume of the debris fallout.

**Kill Chain:** The sequence of activities that must occur to complete a mission goal. For this dissertation, the elements of the kill chain are: Detect, Track, Assign Weapon, Engage, and Kill Assessment.

**Recoverability:** The ease for which a failed system can be restored to operational use.

**Memory leak:** An error in a program's dynamic-store allocation logic that causes it to fail to reclaim discarded memory and can result in a system crash due to memory exhaustion.

**Mission:** The task, together with the purpose, that clearly indicates the action to be taken and the reason therefore. (Joint Pub 1-02)

**Mission type order:** Order to a unit to perform a mission without specifying how it is to be accomplished. (Joint Pub 1-02)

**Model:** A representation of a physical system or process intended to enhance the software engineer's ability to understand, predict, or control its behavior.

**Model checking:** The systematic approach for testing functional assertions and substantiating the desired system behavior in the model. Model checking is not a proof of correctness; however, model checking involves creating functional models of a system and analyzing the model against the formal representations of the desired behavior.

**Operational control:** Transferable command authority that may be exercised by commanders at any echelon at or below the level of combatant command. Operational control is inherent in Combatant Command (command authority) and is the authority to perform those functions of command over subordinate forces involving organizing and employing commands and forces, assigning tasks, designating objectives, and giving authoritative direction necessary to accomplish the mission. Also called **OPCON**. (Joint Pub 1-02)

**Operations tempo:** The rate of military actions or missions.

**Oracle**: A tool to evaluate the results of a test case as either pass or not passed. The oracle is the test key that contains the inputs for a system and the associated required output for each input.

**Passive component:** A component that executes only when called upon to do so by an active component.

**Pre-condition:** A fact that must always be true just prior to execution of a specific section of code.

**Post-condition:** A fact that must always be true just after the execution of a specific section of code.

**Precision engagement:** The ability of joint forces to locate, surveil, discern, and track objectives or targets; select, organize, and use the correct systems; generate desired effects, assess results; and reengage with decisive speed and overwhelming operational tempo as required, throughout the full range of military operations.

**Predicate:** A function that represents the truth or falsehood of some condition.

**Race condition:** A condition in which the state of a resource depends on timing conditions that are not predictable. A race condition occurs if the final result of a computation that requires access to a critical section is executed by two or more processes, and the final result of the computation depends on the order in which those processes execute. For example, if two processes ($P_A$ and $P_B$) write different values $V_A$ and $V_B$ to the same variable in a critical section, then the final value of the variable is determined by the order in which $P_A$ and $P_B$ execute.

**Reactive system:** A system for which its behavior is primarily caused by reactions to external events as opposed to being internally generated stimuli.

**Real-time system:** A real-time system is one for which producing correct computations as a result of an external event is equally as critical as meeting the deadlines for those computations.

**Reliability:** The property that a system can operate continuously without experiencing a failure.

**Requirement:** A criterion that a system must meet. A requirement may define what a system must do, characteristics it must have, and levels of performance it must attain.

**Robustness:** A characteristic of a system that is failure and fault tolerant. Such a system handles unexpected states in a manner that minimizes performance degradation, data corruption, and incorrect output.

**Rules of engagement:** Directives issued by competent military authority that delineate the circumstances and limitations under which United States forces will initiate and/or continue combat engagement with other forces encountered. Also called ROE. (Joint Pub 1-02)

**Safety**: The property of avoiding a catastrophic outcome given a system fails to operate correctly.

**Schedulability:** The determination of whether a group of tasks, whose individual CPU utilization is known, will meet their deadlines.

**Situational awareness:** Perception of available facts, comprehension of the facts in relation to the individual's expert knowledge, and projecting how the situation is likely to develop in the future.

**Software reuse:** The act of selecting and employing a chunk of software that was designed and implemented for use in other systems without modification to that chunk of software.

**Software salvage:** The act of selecting a chunk of software and modifying it for use in another system.

**Specification:** An articulation of either: (1) desired system behavior that is expressed as a feature, function, property, or capability, or (2) an undesired system behavior that can be expressed as a limitation, constraint, negative (e.g., the system shall not…), or condition.

**Specified command:** A command that has broad continuing missions and that is established by the President through the Secretary of Defense with the advice and assistance of the Chairman of the Joint Chiefs of Staff. It normally is composed of forces from a single Military Department. Also called specified combatant command. (Joint Pub 1-02)

**State:** A recognizable situation that exists over an interval of time.

**State explosion:** The condition in which the size of the state space exceeds the memory capacity of the automated tool to check every trace in the model.

**State transition:** A change in state that is caused by an input event.

**Subsystem:** A testable collection of classes, objects, components, and modules that typically share a common attribute or contribute to a common goal. (Binder)

**Surveillance**: The systematic observation of aerospace, surface or subsurface areas, places, persons, or things, by visual, aural, electronic, photographic, or other means. (Joint Pub 1-02)

**System-of-Systems:** An amalgamation of legacy systems and developing systems that provide an enhanced military capability greater than that of any of the individual systems within the system-of-systems.

**System behavior:** The collective responses of a system as it reacts to stimuli such as sensory information, a clock, or a received transaction.

**Tactical control:** The detailed and, usually, local direction and control of movements or maneuvers necessary to accomplish missions or tasks assigned. Also called TACON. (Joint Pub 1-02)

**Targeting**: 1. The process of selecting targets and matching the appropriate response to them taking account of operational requirements and capabilities. 2. The analysis of enemy situations relative to the commander's mission, objectives, and capabilities at the commander's disposal, to identify and nominate specific vulnerabilities that, if exploited, will accomplish the commander's purpose through delaying, disrupting, disabling, or destroying enemy forces or resources critical to the enemy. (Joint Pub 1-02)

**Temporal logic:** An extension of propositional logic that incorporates special operators that cater for time. With temporal logic one can specify how components, protocols, objects, modules, procedures and functions behave as time progresses. The specification is done with temporal logic statements that make assertions about properties and relationships in the past, present, and the future.

**Test-ready model:** One that contains sufficient information for which to automatically produce test cases for its implementation.

**Time-critical task:** A task that needs to meet a hard deadline.

**Transparent:** A distributed system that appears to be a single system to the users that operate the distributed system, and the applications that reside and execute on the distributed system. (Tannenbaum and van Steen)

**Trustworthy system:** One that provides the appropriate levels of correctness and robustness in accomplishing its mission while demonstrating the appropriate levels of availability, consistency, reliability, safety, and recoverability to the degree that justifies a user's confidence that the system will behave as expected.

**Unified command:** A command with broad continuing missions under a single commander and composed of forces from two or more Military Departments, and which is established by the President, through the Secretary of Defense with the advice and assistance of the Chairman of the Joint Chiefs of Staff. Also called unified combatant command. (Joint Pub 1-02)

**Validation:** The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. (IEEE Std. 610.12-1990)

**Verification:** The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (IEEE Std. 610.12-1990)

**White-box testing:** A software testing technique whereby explicit knowledge of the internal workings of the component being tested is used to examine the outputs.

# APPENDIX B. ACRONYMS

| | |
|---|---|
| AADC | Area air defense commander |
| ABL | Airborne laser |
| ABM | Anti-ballistic missile |
| ACTD | Advanced concept technology demonstration |
| AD | Air defense |
| ADA | Air defense artillery |
| ADCP | Air defense communications platform |
| ADG | Active defense group |
| ALERT | Attack and launch early report to theater |
| AO | Area of Operations |
| AOA | Amphibious objective area |
| AOC | Air Operations Center |
| AOR | Area of responsibility |
| ATACMS | Army tactical missile system |
| ATO | Air tasking order |
| AWACS | Airborne warning and control system |
| BDA | Battle damage assessment |
| BMC4I intelligence | Battle management command, control, communications, computers, and |
| BMD | Ballistic missile defense |
| BMDO | Ballistic Missile Defense Organization |
| BMDS | Ballistic Missile Defense System |
| BPI | Boost-phase intercept |
| CAP | Crisis action planning |
| C2 | Command and control CAP Combat air patrol |
| C3I | Command, Control, Communications, and intelligence |
| CEC | Cooperative engagement capability |
| CENTCOM | United States Central Command |
| CEP | Circular error probable |
| CIC | Combat information center |
| CJCS | Chairman, Joint Chiefs of Staff |

| | |
|---|---|
| CM | Configuration management |
| CO | Commanding officer |
| COA | Course of action |
| COCOM | Combatant Commander |
| COEA | Cost and operational effectiveness analysis |
| CONOPS | Concept of operations |
| CONPLAN | Operations plan in concept format |
| CONUS | Continental United States (excluding Alaska and Hawaii) |
| COP | Common operational picture |
| COTS | Commercial off the shelf |
| CRC | Control and reporting center |
| DAL | Defended asset list |
| DE | Directed energy |
| DIA | Defense Intelligence Agency |
| DISA | Defense Information Systems Agency |
| DoD | Department of Defense |
| DSP | Defense Support Program |
| EO | Electrical-optical |
| EUCOM | United States European Command |
| EW | Early warning |
| EXORD | Execute order |
| GBI | Ground-based interceptor |
| GBR | Ground-based radar |
| GCCS | Global command and control system |
| GEM | Guidance enhanced missile (PATRIOT) |
| GGIG | Global information grid |
| GMD | Ground-based Missile Defense |
| GPS | Global Positioning System |
| HQ | Headquarters |
| IA | Information assurance |
| ICBM | Intercontinental ballistic missile |
| ICC | Information Coordination Central (PATRIOT) |
| IER | Information exchange requirement |
| IRS | Interface Requirements Specification |

| | |
|---|---|
| IOC | Initial operational capability |
| IPB | Intelligence preparation of the battle space |
| IR | Infrared |
| IRBM | Intermediate-range ballistic missile |
| IRST | Infrared search and track |
| ITW/AA | Integrated tactical warning/attack assessment |
| JCS | Joint Chiefs of Staff |
| JCTN | Joint composite tracking network |
| JDN | Joint data network |
| JEZ | Joint engagement zone |
| JFACC | Joint force air component commander |
| JFC | Joint force commander |
| JFCOM | Joint Forces Command |
| JFMCC | Joint force maritime component commander |
| JIC | Joint intelligence center |
| JMCIS | Joint maritime command information system |
| JP | Joint publication |
| JPN | Joint planning network |
| JS | Joint staff |
| JSOC | Joint Special Operations Command |
| JSTARS | Joint surveillance and target attack radar system |
| JTA | Joint technical architecture |
| JTAGS | Joint tactical ground station |
| JTF | Joint task force |
| JTIDS | Joint tactical information distribution system |
| JTMD | Joint theater missile defense |
| KE | Kinetic energy |
| KV | Kill vehicle |
| KW | Kinetic warhead |
| MDA | Missile Defense Agency |
| MEADS | Medium extended air defense system |
| MEZ | Missile engagement zone |
| MNS | Mission need statement |
| MLRS | Multiple launch rocket system |

| | |
|---|---|
| MRBM | Medium-range ballistic missile |
| NATO | North Atlantic Treaty Organization |
| NBC | nuclear, biological, and chemical |
| NASA | National Aeronautics and Space Administration |
| NCA | National Command Authority |
| NMCC | National Military Command Center |
| NMD | National missile defense |
| NORTHCOM | United States Northern Command |
| OCONUS | Outside the continental United States |
| OOB | Operational order of battle |
| OODA | Observe, orient, decide, act |
| OPLAN | Operations plan |
| OPORD | Operations order |
| OPTEMPO | Operations tempo |
| ORD | Operational Requirements Document |
| OSD | Office of the Secretary of Defense |
| PAC | Patriot advanced capability |
| PACOM | Pacific Command |
| PATRIOT | phased array tracking radar intercept on target |
| PDAL | Prioritized defended asset list |
| $P_k$ | Probability of kill |
| POM | Program objective memorandum |
| R&D | Research and development |
| RCS | Radar cross-section |
| R&D | Research and development |
| RDT&E | Research, development, test, and evaluation |
| RF | Radio frequency |
| ROE | Rules of engagement |
| RV | Reentry vehicle |
| SAM | Surface-to-air missile |
| SATCOM | Satellite communications |
| SBIRS-LOW | Space-based infrared system-low earth orbit |
| SBWS | Space-based warning system (DSP + TES) |
| SDI | Strategic Defense Initiative |

| | |
|---|---|
| SDIO | Strategic Defense Initiative Organization |
| SMTS | Space and missile tracking system |
| SOCOM | United States Special Operations Command |
| SOF | Special operations forces |
| STRATCOM | United States Strategic Command |
| SRBM | Short-range ballistic missile |
| SRS | System Requirements Specifications |
| TAOC | Tactical air operations center |
| TBM | Theater ballistic missile |
| TBM-WMD | Theater ballistic missile—weapons of mass destruction |
| TBMD | Theater ballistic missile defense |
| TCT | Time critical target |
| TDDS | Tactical data distribution system |
| TEL | Transporter-erector-launcher (for TBM) |
| THAAD | Theater high-altitude area defense |
| TIBS | Tactical information broadcast service |
| TLAM | Tomahawk land attack missile |
| TM | Theater missile |
| TMD | Theater missile defense |
| TOC | Tactical operations center |
| TPFDD | Time-phased force and deployment data |
| TPFDL | Time-phased force and deployment list |
| TRAP | TRE and related applications (now TDDS) |
| TRE | Tactical receive equipment |
| UCP | Unified Command Plan |
| UOES | User operational evaluation system |
| USA | United States Army |
| USAF | United States Air Force |
| USMC | United States Marine Corps |
| USN | United States Navy |
| VCJCS | Vice-Chairman, Joint Chiefs of Staff |
| WMD | Weapons of mass destruction |

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF REFERENCES

[1] Bachmann, F., Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., Seacord, R., and Wallnau, K. Volume II: Technical Concepts of Component-Based Software Engineering. Technical Report CMU/SEI-2000-TR-008, Carnegie-Mellon University/Software Engineering Institute, Pittsburg, Penn., May 2000.

[2] Bass, L., Clements, P. and Kazman, R. *Software Architecture in Practice: 2nd ed.*, Reading, Mass.: Addison-Wesley, 2003.

[3] Binder, R. V. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Reading, Mass.: Addison-Wesley, June 2001.

[4] Boehm, B. and Basili, V. R. Software defect reduction top 10 list. *IEEE Computer*, Jan. 2001, pp. 135-137.

[5] Bowen, J. P. and Hinchey, M. G. Seven more myths of formal methods. *IEEE Software*, July 1995, pp. 34-41.

[6] Boyd, J. R. A Discourse on Winning and Losing: Patterns of Conflict. Unpublished lecture notes, Dec. 1986. (Typewritten)

[7] Brown, M. L., The Application of Safety Kernels in Weapon-Related Systems, Technical Report, NOSSA-TR-2004-003, National Ordance Safety and Security Activity of the Naval Sea Systems Command, Indian Head, Md., May 2004.

[8] Butler, R.W., Caldwell, J., Carreno, V., Holloway, M., Miner, P., and Di Vito, B. NASA Langley's Research and Technology-Transfer Program in Formal Methods, http://shemesh.larc.nasa.gov/fm/NASA-over.pdf, May 2002.

[9] Caffall, D. S. and Michael, J. B. A new paradigm for requirements specification and analysis of system-of-systems. In Wirsing, M., Knapp, A., and Balsamo, S., eds., *Lecture Notes in Computer Science*, Berlin: Springer-Verlag, No. 2941, pp. 108-121.

[10] Caffall, D. S., Michael, J. B., and Shing, M.-T. Developing highly predictable system behavior in real-time battle-management software. In Savoie, M. J., Chu, H.-W., Michael, J., and Pace, P., eds., *Proc. Int. Conf. on Computing, Commun. and Control Technologies*, Orlando, Fla.: Int. Inst. of Informatics and Systemics (Austin, Tex., Aug. 2004), vol. 6, pp. 7-12.

[11] Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. Progress on the state explosion problem in model checking. In Wilhelm, R., ed., *Lecture Notes in Computer Science*, Vol. 2000, Heidelberg, Ger.: Springer-Verlag, 2001, pp. 176-194.

[12] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., and Stafford, J. *Documenting Software Architectures: Views and Beyond*. Reading, Mass.: Addison Wesley, 2003.

[13] U.S. Congress, Office of Technology Assessment, *Ballistic Missile Defense Technologies*, OTA-ISC-254, Washington, D.C., U.S. Government Printing Office, Sept. 1985.

[14] Coram, R. *Boyd: The Fighter Pilot Who Changed the Art of War*. New York: Little Brown and Co., 2002.

[15] Crnkovic, I. and Larsson, M., eds. *Building Reliable Component-Based Software Systems*. Norwood, Mass.: Artech House, 2002.

[16] del Mar Gallardo, M., Martínez, J., Merino, P., and Pimentel, E. Abstract model checking and refinement of temporal logic in αSPIN. In *Proc. Third Int. Conf. on Application of Concurrency to System Design*, IEEE (Guimarães, Port., June 2003), pp. 245-246.

[17] Dijkstra, E. W. The structure of the multiprogramming system. *Communications of the ACM*, Vol. 11, No. 5 (May 1968), pp. 341-346.

[18] U.S. Department of the Army. *Tactics, Techniques, and Procedures for the Targeting Process*. Field Manual 6-20-10, May 1996.

[19] U.S. Department of Commerce. The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards and Technology Planning Report 02-03, May 2002.

[20] U.S. Department of Defense. *Joint Capabilities Integration and Development System*. Chairman of the Joint Chiefs of Staff Instruction 3170.01D, Mar. 12, 2004.

[21] U.S. Department of Defense. *Department of Defense Dictionary of Military and Associated Terms*. Joint Pub. 1-02, Apr. 12, 2001 (as amended through May 23, 2003).

[22] US Department of Defense. *Doctrine for Joint Theater Missile Defense*. Joint Pub. 3-01.5, Joint Chiefs of Staff, Feb. 1996.

[23] U.S. Department of Defense. *Joint Vision 2020*. Washington, D.C.: U.S. Government Printing Office, June 2000.

[24] U.S. Department of Defense. *Joint Force Command and Control Concept to Guide Standing Joint Force Headquarters Development by 2005*. Joint Forces Command, Mar. 5, 2003.

[25] U.S. Department of Defense. *Network Centric Warfare: Department of Defense Report to Congress*, Department of Defense, 2001.

[26] U.S. Department of Defense. Operation of the Defense Acquisition System. Department of Defense Instruction 5000.2, May 12, 2003.

[27] U.S. Department of Defense. *Tactical Digital Information Link (TADIL) J Message Standard.* MIL-STD-6016A, Joint Technical Architecture - Version 4.0, July 2002.

[28] U.S. Department of Defense. *The Defense Acquisition System*. Department of Defense Directive 5000.1, May 12, 2003.

[29] U.S. Department of Defense. *The Joint Staff Officer's Guide 2000*. JFSC Pub 1, National Defense University, Joint Forces Staff College, Norfolk, Va., 2000.

[30] Douglass, B. P. *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Reading, Mass.: Addison-Wesley, 1999.

[31] Fowler, M. *Refactoring: Improving the Design of Existing Code,* Addison Wesley, 1999.

[32] Fowler, M. Who needs an architect? *IEEE Software*, Sept. 2003, pp. 11-13.

[33] French, M. Network-centric warfare progressing, *Federal Computer Week*, May 1, 2003.

[34] Friedman, M. A. and Voas, J. M. *Software Assessment: Reliability, Safety, Testability*. New York: John Wiley & Sons, 1995.

[35] Ganssle, J.G. Disaster. Embedded.com, *Embedded Systems Programming J.*, Vol. 11, No. 5, May 1998.

[36] Garg, V. K. *Elements of Distributed Computing*, New York: John Wiley & Sons, 2002.

[37] Erwin, S. I. General Jumper. Time to change traditional program advocacy, *National Defense*, July 2002, pp. 14-15.

[38] Garland, J. and Anthony R. Large-Scale Software Architecture, West Sussex, England: John Wiley & Sons, 2003.

[39] National Missile Defense: Schedule and technical risks represent significant development challenges. Report GAO/NSIAD-98-28, U.S. General Accounting Office, Washington, D.C., Dec. 1997.

[40] Gnesi, S., Latella, D., and Massink, M. In *Proc. Fourth Int. Symposium on High Assurance Systems Engin.*, IEEE, (Washington, D.C., Nov. 1999), pp 46-55.

[41] Gomaa, H. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Reading, Mass.: Addison-Wesley, 2000.

[42] Hall, A. Seven myths of formal methods, *IEEE Software*, Sept. 1990, pp. 11-19.

[43] He, W. and Goddard, S. Capturing an application's temporal properties with UML for real-time. In *Proc. Fifth Int. Symposium on High Assurance Systems Engin.*, IEEE, (Albuquerque, N.M., Nov. 2000), pp. 65-74.

[44] IEEE. Standard Glossary of Software Engineering Terminology, IEEE Std. 610.12-1990.

[45] Kaufman, P. R. Sensors Emerge as More Crucial Weapons Than Shooters, IEEE Spectrum Online*,* News Analysis, July 2002.

[46] Kenne, L. F. Tightening the kill chain: Broadening information access, *Intercom*, Jan. 2003, pp. 6-9.

[47] Lang, B. Finding Errors Using Model-Based Verification, www.sei.cmu.edu/news-at-sei/features/2001/1q01/feature-2-1q01.htm, 2001.

[48] Leckie, R. *None Died In Vain*. New York: Harpers Collins, 1990.

[49] Leishman, T.R. and Cook, D.A. Requirements risks can drown software projects, *CrossTalk*, Apr. 2002, pp. 4-8.

[50] Leffingwell, D. and Widrig, D. *Managing Software Requirements: A Unified Approach.* Reading, Mass.: Addison-Wesley, 2000.

[51] Lewis, G. A., Comella-Dorda, S., Gluch, D. P., Hudak, J., and Weinstock, C. Model-based verification: Analysis guidelines. Technical Note CMU/SEI-2001-TN-028, Software Engineering Institute, Pittsburgh, Penn., Dec. 2001.

[52] Liu, J. W. S. *Real-Time Systems*. Upper Saddle River, N.J.: Prentice Hall, 2000.

[53] Loeb, V. Friendly fire deaths traced to dead battery, *Washington Post*, Mar. 24, 2002, p. A21.

[54] Lutz, R.R. Analyzing software requirements errors in safety-critical, embedded systems. In *Proc. Fourth Int. Symposium on Requirements Engin.*, IEEE, (San Diego, Calif., Jan. 1993), pp. 126-133.

[55] Mantle, P. J. *The Missile Defense Equation: Factors for Decision Making*. Reston, Va.: American Institute of Aeronautics and Astronautics, Inc., 2004.

[56] Meyers, C. B., Feiler, P. H., Marz, T. Proc. Real-Time Systems Engin. Workshop. Special Report CMU/SEI-2001-SR-022, Software Engineering Institute, Pittsburgh, Penn., Aug. 2001.

[57] Michael, J.B. Gaining the Trust of Stakeholders in Systems-of-Systems: A Brief Look at the Ballistic Missile Defense System. Technical Report NPS CS-04-006, Naval Postgraduate School, Monterey, Calif., May 2004.

[58] Michael, J.B. and Lawler, G. M. Classification paradigms for comparing distributed systems. Unpublished manuscript, Nov. 2002.

[59] Musa, J.D., Iannino, A., and Okumoto, K. *Software Reliability Measurement Prediction Application*. New York: McGraw-Hill, 1997.

[60] National Aeronautics and Space Administration Formal Methods Specification and Verification Guidebook for Software and Computer Systems, Volume I: Planning and Technology Insertion. Office of Safety and Mission Assurance, NASA/TP-98-208193, 1998.

[61] Parnas, D. L. *Software Fundamentals: Collected Papers by David L. Parnas*. Reading, Mass.: Addison-Wesley, 2001.

[62] Patriot Missile Software Problem. Report GAO/IMTEC-92-26, U.S. General Accounting Office, Washington, D.C., Feb. 1992.

[63] Paul, R. Rapid and adaptive end-to-end T&E of joint systems of systems. Presentation at the Fifteenth Annual Software Technology Conf., Salt Lake City, Utah, Apr. 2003.

[64] Preckshot, G. G., A kernel approach to safety systems. Technical Report UCRL-ID-120075, Lawrence Livermore National Laboratory, Nov. 14, 1994.

[65] Rempt, R. The Navy in the twenty-first century, Part II: Theater Air and Missile Defense, *Johns Hopkins APL Technical Digest*, Vol. 22, No. 1 (2001), pp. 21-28.

[66] Selic, B. The pragmatics of model-driven development, *IEEE Software*, Sept./Oct. 2003, pp. 19-25.

[67] Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline.* Saddle River, NJ: Prentice Hall, 1996.

[68] Sitaraman, M. and Gandi, G. Design-time error detection using assertions. In Cook, J. E. and Ernst, M. D., eds., *Proc. ICSE Workshop on Dynamic Analysis*, (Portland, May 2003).

[69] Standish Group, CHAOS: A Recipe for Success, The Standish Group International, 1999.

[70] Standish Group, The Chaos Report, West Yarmouth, MA.: The Standish Group, 1994.

[71] Standish Group, CHAOS Chronicles Version 3.0, West Yarmouth, Mass.: The Standish Group, 2003.

[72] Stenbit, J. P. Horizontal fusion: Enabling net-centric operations and warfare, *Crosstalk*, Jan. 2004.

[73] Storey, N. *Safety-Critical Computer Systems*. New York: Addison Wesley, 1996.

[74] Szyperski, C. *Component Software: Beyond Object-Oriented Programming*. Reading, Mass.: Addison-Wesley, 2nd ed., 2002.

[75] Tanenbaum, A. S. and van Steen, M. *Distributed Systems: Principles and Paradigms*. Upper Saddle River, N.J.: Prentice Hall, 2002.

[76] Tenet, G. J. The Worldwide Threat 2004: Challenges in a Changing Global Context, *Testimony of Director of Central Intelligence before Senate Select Committee on Intelligence*, Feb. 2004.

[77] Theisen, E. E. Ground-aided precision strike: heavy bomber activity in operation enduring freedom. Maxwell Paper No. 31, Air War College, Air University Press, Maxwell Air Force Base, Alabama, July 2003.

[78] Voas, J. Mitigating the potential for damage caused by COTS and third-party software failures. In Ghezzi, C. and Fusani, M., eds., *Proc. 4th Int. Conf. on Achieving Quality in Software*, Pisa, Italy: Consorzio Universitario in Ingegneria della Qualita (Venice, Mar. 1998).

[79] Vinu, G. and Vaughn, R. Application of lightweight formal methods in requirements engineering. *CrossTalk,* Jan. 2003.

[80] Walker, R.W. DoD's Grand Plan, *Government Computer News*, Feb. 24, 2003.

[81] Wallace, D. R. and Kuhn, D. R. Lessons from 342 medical device failures. In *Proc. Fourth International Symposium on High Assurance Systems Engin.*, IEEE, (Washington, D.C., Nov. 1999), pp. 123-131.

[82] Young, R. R. *Effective Requirements Practices.* Reading, Mass.: Addison-Wesley, 2001.

[83] Popper, S. W., Bankes, S. C., Callaway, R. and DeLaurenetis, D. System of systems symposium: report on a summer conversation. Potomac Institute for Policy Studies, Arlington, Va., Nov. 2004.

[84] Smith, D., Morris, E., and Carney, D. Adoption centric problems in the context of system of systems interoperability. In *Proc. 4th Int. ICSE Workshop on Adoption-Centric Software Engineering*, IEEE (Edinburgh, Scotland, May 2004), pp. 63-68.

[85] Jones, C. and Randell, B. Dependable pervasive systems, CS-TR-839 School of Computing Science, University of Newcastle upon Tyne, Apr. 2004.

[86] Knight, J. C. An introduction to computing system dependability. In *Proc. 26th Int. Conf. on Software Engineering*, IEEE (Edinburgh, Scotland, May 2004), pp. 730-731.

[87] Keating, C., Rogers, R., Unal, R., Dryer, D., Sousa-Poza, A., Safford, R., Peterson, W., and Rabadi, G. System of systems engineering. *Engineering Management J.*, Vol. 15, No. 3 (Sept. 2003) , pp. 36-45.

[88] Mustapic, G., Wall, A., Norstrom, C., Ivica Crnkovic, Sandstrom, K., Froberg, J., and Andersson, J. Real world influences on software architecture – interviews with industrial system experts. In *Proc. 4th Working Conf. on Software Architecture*, IEEE (Oslo, Norway, June 2004), pp. 101-111.

[89] Maier, M. Architecting principles for systems of systems. In *Proc. 6th Annual Int. INCOSE Symposium*, Seattle, Wash.: Int. Council on Systems Engineering (Boston, Mass., July 1996), pp. 567-574.

[90] Fisher, D. A., and Smith, D. Emergent issues in interoperability. *news@sei*, Software Engineering Institute, No. 3, 2004, pp. 1-5.

[91] Rhodes, D. and Hastings, D. The case for evolving systems engineering as a field within engineering systems, Paper presented at Massachusetts Institute of Technology Engineering Systems Symposium, Cambridge, Mass., esd.mit.edu/symposium/pdfs/papers/rhodes.pdf, Mar. 2004.

[92] Chen, P. and Han, J. Facilitating system-of-systems evolution with architecture support. In *Proc. 4th Int. ICSE Workshop on Principles of Software Evolution*, IEEE (Vienna, Austria, Sept. 2001), pp. 130-133.

[93] Hooks, I. Managing requirements for a system of systems, *CrossTalk*, Aug. 2004.

[94] Sage, A. P. Conflict and risk management in complex system of systems issues. In Proc.Int. Conf. on Systems, Man, and Cybernetics, Vol. 4, Oct. 2003, pp. 3296-3301.

[95] Kasser, J. The acquisition of a system of systems is just a simple multi-phased parallel-processing paradigm. In *Proc. Int. Engineering Management Conf.*, Vol. 2, IEEE (Cambridge, England, 2002), pp. 510-514.

[96] Crossley, W. A. System of systems: An introduction of Purdue University's schools of engineering's signature area. Paper presented at Massachusetts Institute of Technology Engineering Systems Symposium, Cambridge, Mass., esd.mit.edu/symposium/pdfs/papers/crossley.pdf, Mar. 2004.

[97] Christian, E. The architecture of GEOSS (global earth observation system of systems. Briefing presented at Industry Workshop on GEOSS Architecture, Federal Geographic Data Committee, United States Geological Survey, iwgeo.ssc.nasa.gov/docs/GEOSSArchitecture.ppt, Reston, Va., May 2004.

[98] Stoudt, C. A. A systems perspective on current ATC trends. *IEEE Aerospace and Electronic Systems Magazine*, Sept. 2002, pp. 28-32.

[99] Greaves, M. Stavridou-Coleman, V., and Laddaga, R., Dependable agent systems. *IEEE Intelligent Systems*, Sept./Oct. 2004, pp. 20-23.

[100] Schaefer, R. Systems of systems and coordinated atomic actions, *ACM SIGSOFT Software Engineering Notes*, Vol. 30, Issue 1, abstract on p. 6 with full article online, Jan. 2005.

[101] Kokar, M. M., Baclawski, K., and Eracar, Y. A. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, May/June 1999, pp. 37-45.

[102] Kleinmann, K., Lazarus, R., and Tomlinson, R. An infrastructure for adaptive control of multi-agent systems. In *Proc.Int Conf. on Integration of Knowledge Intensive Multi-Agent Systems*, IEEE (Boston, Mass., Sept. 2003), pp. 230-236.

[103] U.S. Congress. Office of Technology Assessment. Automatic Train Control in Rail Rapid Transit. Washington, D.C.: Government Printing Office, 1976.

THIS PAGE INTENTIONALLY LEFT BLANK

# INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Bret Michael
   Department of Computer Science
   Naval Postgraduate School

4. Professor Man-tak Shing
   Department of Computer Science
   Naval Postgraduate School

5. Professor Doron Drunsinsky
   Department of Computer Science
   Naval Postgraduate School

6. Professor Dan Boger
   Chairman, Department of Information Sciences
   Naval Postgraduate School

7. Dr. Kevin Greaney
   Senior Software Engineer
   DB Data Systems

8. General Robert Dehnert
   Program Director for C2BMC
   Missile Defense Agency

9. Mr. Richard Ritter.
   Deputy Program Director for C2BMC
   Missile Defense Agency

10. LTC Tom Cook
    Chief, Advanced Battle Manager Development Team
    Missile Defense Agency

11. Ms. Paula Lynn Jones
Human Resources
Joint Interoperability Test Command

12. Professor David Parnas
Software Quality Research Laboratory
University of Limerick

13. Professor John Knight
Department of Computer Science
University of Virginia

14. Mr. R. K. Callaway
Potomac Institute for Policy Studies

15. Professor Duminda Wijesekera
Department of Information and Software Engineering
George Mason University

16. Professor Phillip Pace
Department of Electrical & Computer Engineering
Naval Postgraduate School

17. Professor Murali Tummala
Department of Electrical & Computer Engineering
Naval Postgraduate School