Aligning Flight Simulation Software with MOSA Standards

Hung Tran
CAE USA Inc.
Tampa, FL
hung.tran@caemilusa.com

ABSTRACT

The Modular Open Systems Approach (MOSA) represents an acquisition and design strategy standardization initiative by the U.S. Department of Defense (DoD), aimed at lowering risk and accelerating innovation through modular design and interoperability. A MOSA system architecture allows for severable components to be updated, added, removed, or replaced throughout the life cycle to afford opportunities for enhanced efficiency, competition, and innovation.

Legacy simulation software—particularly in the flight simulation domain—has delivered proven accuracy and integration over decades, these systems often fall short of current adaptability and sustainability requirements, especially under MOSA mandates. This paper introduces a blueprint—a structured, repeatable framework—for reworking legacy simulation software to align with MOSA principles. The approach is exemplified through the modernization of Aircraft Survivability Equipment (ASE) simulation package originally developed in the 1980s. The framework spans six phases: system evaluation, functional decomposition, interface standardization, modular refactoring, common service extraction, and integration validation. Emphasis is placed on preserving domain-specific expertise embedded in legacy code while adopting modern software engineering practices to improve modularity, maintainability, and interoperability.

While the primary focus of this work is the modernization of legacy flight simulation software, the proposed framework is not domain-bound. Its modular design principles, interface standardization strategies, and componentization approach are equally applicable to other simulation domains such as medical training, aircraft maintenance, synthetic environments, and more. In fact, the same blueprint could be adapted to real-world software systems beyond simulation, where modularity, interoperability, and maintainability are critical. This cross-domain versatility makes the framework a valuable tool for any organization seeking to modernize complex, monolithic systems in alignment with MOSA principles.

ABOUT THE AUTHORS

Hung Tran is a Technical Authority for Software Engineering at CAE USA Inc. He joined CAE USA more than 25 years ago and has worked on the modeling and simulation of several electronic warfare systems, weapon systems, and virtual environments. Hung holds a Bachelor of Science in Electrical Engineering from Polytechnique Montréal at University de Montréal in Quebec, Canada.

Aligning Flight Simulation Software with MOSA Standards

Hung Tran
CAE USA Inc.
Tampa, FL
hung.tran@caemilusa.com

I. INTRODUCTION

The Modular Open Systems Approach (MOSA) is a Department of Defense (DoD) mandate that emphasizes open architecture design to promote interoperability, cost-effectiveness, and long-term system sustainability. As modern defense platforms become increasingly complex and interconnected, MOSA serves as a strategic foundation for rapidly integrating new capabilities, minimizing vendor lock-in, and ensuring that mission-critical systems remain adaptable and relevant over time. By enabling modular upgrades and common interfaces, MOSA not only extends the operational lifecycle of defense systems but also reduces total ownership costs and accelerates innovation through reuse and competition (NDIA 2023). Despite these advantages, a significant number of existing systems—particularly in the domain of flight simulation—remain heavily dependent on legacy software architectures. Many of these simulation packages, developed as early as the 1980s, continue to deliver accurate and reliable results and are deeply integrated with hardware and domain-specific training requirements. Based on our internal analysis and several external marketing sources, we estimate that 30% to 50% of current flight simulators continue to rely on legacy components for critical modules such as aerodynamics, avionics, aircraft systems, and system dynamics. However, the very structure that made these legacy systems robust—monolithic, tightly coupled, and highly customized—has become a critical barrier to modernization. These characteristics make legacy systems difficult to modify, costly to maintain, and fundamentally misaligned with MOSA principles. As a result, their continued use poses long-term risks to affordability, upgradeability, and innovation in training and simulation (Baldwin and Clark, 2000).

This paper addresses this challenge by presenting a case study on the modernization of a flight simulation software package originally developed in the 1980s. The effort focused on re-architecting the software to improve modularity, maintainability, and interoperability, while preserving the deep domain knowledge embedded within the legacy codebase. At the core of this work is a proposed blueprint—a structured, repeatable framework for modernizing legacy simulation systems in accordance with MOSA. This framework offers technical guidance, strategic planning considerations, and practical design patterns applicable to various legacy software environments, while also reducing acquisition costs by fostering vendor competition and minimizing proprietary lock-in, in alignment with the DoD's open systems strategy. By sharing this process and the lessons learned, the paper aims to support the broader defense software community in transforming legacy systems into flexible, future-ready platforms—thus ensuring long-term value, reducing sustainment costs, and enabling continuous innovation.

II. BACKGROUND

At its core, MOSA is an architectural and design philosophy that guides how systems—especially complex ones like flight simulators—should be structured to ensure long-term adaptability and interoperability (NDIA 2023).

MOSA is built on four foundational principles (Davendralingam et al. 2019):

1. **Modularity**: Systems should be developed as a collection of loosely coupled components or modules, each encapsulating specific functionality. This makes it easier to isolate changes, manage complexity, and facilitate system updates or replacements without disturbing the entire architecture.
2. **Open Interfaces**: Components must communicate through clearly defined, non-proprietary, and publicly documented interfaces. These interfaces serve as contracts that decouple implementation details from system integration, enabling vendors or programs to evolve components independently.
3. **Interoperability**: Components should be designed to operate seamlessly with one another, regardless of vendor, platform, or technology stack. This ensures flexibility in integrating multi-domain capabilities and allows for shared use across services.
4. **Standardization**: The use of industry and government-adopted standards—whether for data formats, communication protocols, or software interfaces—is key to achieving reuse and system-of-systems

integration goals. Standardization also supports certification and ensures compliance across different defense programs.

By following MOSA principles, systems gain key lifecycle advantages, including reduced costs, faster upgrades, and improved sustainability. Modular architectures enable independent updates or replacements, avoiding expensive system-wide overhauls. Additionally, MOSA reduces vendor lock-in and legacy system dependence, enhancing long-term adaptability to mission changes, technological advances, and obsolescence risks.

Recognizing its transformative potential, the Department of Defense (DoD) has formally embedded the Modular Open Systems Approach (MOSA) into key acquisition policies and strategies. DoD Instruction 5000.88 (DoD, 2020) mandates MOSA implementation for all new programs and major upgrades, while the approach is also foundational to the DoD Digital Engineering Strategy and emphasized by the Modular Open Systems Working Group (MOSWG, 2020). Legislative momentum further reinforced this shift: in FY 2016, the House Armed Services Committee (HASC, 2015) concluded that the conventional acquisition system was "not sufficiently agile to support warfighter demands" (HASC Report 114-102). In response, the FY 2017 National Defense Authorization Act (NDAA) amended Title 10 of the United States Code, directing DoD to implement MOSA for any Major Defense Acquisition Program (MDAP) receiving Milestone A or Milestone B approval after January 1, 2019. The legislation also introduced pivotal definitions, such as "major system component" and "major system interface," to support enforceable modularity. These policy actions reflect a department-wide pivot toward open, adaptable, and sustainable architectures—particularly for simulation, training, command and control, and embedded weapon systems. In this evolving landscape, MOSA is not merely a best practice—it is a mandated, strategic enabler for future mission success.

Despite significant advances in software development, many flight simulators still rely on legacy components from the 1980s and 1990s—not merely out of inertia, but due to practical factors such as proven reliability, tight hardware integration, and regulatory certification constraints. These systems have undergone decades of validation, delivering accurate flight dynamics and systems behavior, and are deeply embedded in certified training environments, where changes can trigger costly recertification. However, modernizing these legacy systems to align with MOSA standards presents serious challenges. Often built with monolithic architectures in procedural languages like FORTRAN or early C, these systems lack modularity, making upgrades difficult. Components are tightly coupled through undocumented data flows and proprietary interfaces, and documentation is frequently outdated or missing—particularly when the code originates from third-party vendors. Moreover, legacy systems typically use closed, non-standard protocols that hinder integration with modern modular subsystems. As defense platforms shift toward digital interoperability and agile lifecycle management, the limitations of legacy software become increasingly problematic, and without strategic modernization, the cost, complexity, and risk of obsolescence continue to grow (Alec et al., 2024; Newcomb et al., 1998).

III. CASE STUDY: AIRCRAFT SURVIVAL EQUIPMENT (ASE) SIMULATION

To ground this effort in a realistic and high-impact context, the modernization approach presented in this paper is demonstrated through the redesign of the Aircraft Survivability Equipment (ASE) simulation package, originally developed in the 1980s. The code was developed in the early C programming language, using procedural logic, global memory, and hard-coded values.

The purpose of the Aircraft Survival Equipment (ASE) is to reduce the vulnerability of aircraft by protecting it against surface-to-air and air-to-air infrared (IR) and radar-guided missile threats, airborne interceptors, and land based anti-aircraft artillery radars that may be encountered during routine and special operations. The ASE is consisting mainly of 1- Threats detection and identification systems: Radar Warning Receiver (RWR), Missile Warning System (MWS), Laser Warning System (LWS), 2- Countermeasure systems: Infrared Countermeasure (IRCM), Countermeasure Dispensing System (CMDS), and 3- Threats detection and Countermeasure: Large Aircraft Infrared Countermeasure (LAIRCM). ASE training can be conducted using flight simulators because they provide several advantages, including a safe training environment and a significant reduction in the cost of training. Additionally, ASE training within flight simulators allows the crew to practice responding to various threat scenarios in a controlled and realistic environment.

A C-130J ASE simulation software was selected as the focal case study due to its rich integration across multiple aircraft systems, its mission-critical role, and the breadth of modernization challenges it presents—making it an ideal candidate to reflect the core principles of the Modular Open Systems Approach (MOSA). First, ASE is deeply

interwoven into the aircraft's avionics architecture. It interacts with a broad range of aircraft sensors and maintains a tightly coupled interface with the aircraft's mission computer—the computing hub responsible for situational management and decision support. This interface enables ASE to receive commands from various cockpit control panels and to transmit computed outputs back to the mission computer for pilot display. The bidirectional, mission-critical communication between ASE and the mission computer highlights the importance of designing clear, modular interfaces during modernization. Second, ASE's operations depend on massive mission-specific data sets, making it an excellent candidate for data-software separation. By decoupling configuration data and operational parameters from the executable code, the modernization effort can significantly reduce maintenance complexity, simplify updates, and improve long-term adaptability—core goals of the MOSA strategy. Third, ASE has extensive interfacing requirements with other avionics systems, including communication, navigation, and flight dynamic subsystems. This necessitates a careful restructuring of software components to support interoperability across system boundaries, providing a practical example of MOSA-driven integration. Fourth, the simulation of ASE relies heavily on standardized mathematical algorithms to compute real-time situational awareness. This creates a natural opportunity to extract these algorithms into a modular library of reusable components, supporting software reuse across other simulation and mission systems, in line with MOSA's objectives of cost and time savings through common components. Finally, the ASE is composed of multiple specialized threats detection and countermeasure subsystems. While distinct in function, these components share a common processing layer that detects, evaluates, and prioritizes threats. This provides a strong use case for developing a shared software core that encapsulates common logic and can be extended by individual subsystems—supporting both modularity and system cohesion. These features make ASE a strong case study, as it reflects the complex interfaces, data-driven logic, and modular challenges common to many legacy systems. Modernizing ASE to align with MOSA not only enhances this specific simulator but also provides a reusable blueprint for upgrading other legacy simulation and mission systems.

The original Aircraft Survivability Equipment (ASE) simulation software embodies many of the design conventions typical of its time. Key limitations hinder the ASE simulation's alignment with the Modular Open Systems Approach (MOSA) can be summarized as follows:

- **Lack of Encapsulation and Modular Design**. The codebase is highly intertwined, with minimal separation between sensor input handling, threat processing, user interface, and system state management.
- **Non-Standardized Data Formats**. Another major area of non-compliance lies in how simulation manages and interprets operational data. Threat libraries, platform-specific parameters, and environment configurations are embedded directly into the code or stored in custom binary formats lacking schema validation or version control.
- **Closed Interfaces**. The ASE simulation was originally built to interface with proprietary mission computers and custom-built I/O hardware, often using hardcoded memory mappings or non-standard communication protocols. These interfaces are tightly bound to specific hardware assumptions, limiting interoperability with newer systems or platforms that expect well-defined, open, and version-controlled APIs.
- **No Support for Component Reuse or Third-Party Insertion**. Core algorithms and interfaces are not isolated into reusable components, leading to duplication of effort across other systems that perform similar tasks. Since algorithms and subsystems are not encapsulated into modular libraries or services, the ASE simulation does not support the insertion of third-party enhancements or reuse of components across similar systems (e.g., other survivability-related simulations). This goes against MOSA's goal of promoting a vendor-agnostic, ecosystem-oriented software architecture that can accommodate a wide range of innovations without vendor lock-in.
- **Limited Scalability and Opaque Data Coupling**. The software does not readily support integration with new avionics or sensor types without extensive rework, restricting its ability to evolve alongside emerging mission needs. Operational data and configuration parameters are embedded within the code itself, rather than externalized into manageable datasets—creating barriers for mission-specific customization.

Despite these challenges, the legacy ASE simulation remains valued for its proven reliability, accurate modeling, and deep domain knowledge. These attributes made it a compelling candidate for modernization—not because it is broken, but because it must evolve to remain relevant and interoperable in a future-ready MOSA-aligned ecosystem.

IV. REDESIGN APPROACH: ALIGNING WITH MOSA STANDARDS

The process undertaken to modernize the Aircraft Survivability Equipment (ASE) simulation software serves not only as a successful case study, but also as a reusable and scalable blueprint for transforming other legacy software systems in alignment with Modular Open Systems Approach (MOSA) principles. This blueprint distills the key stages of the ASE redesign into a structured methodology that can be applied across a broad spectrum of simulation and embedded software domains. It offers a practical and adaptable framework to guide engineering teams through the complexities of decoupling tightly integrated legacy code, introducing modularity, standardizing interfaces, and ensuring long-term maintainability.

The following six-step process captures the core elements of this transformation and can serve as a foundation for similar modernization efforts seeking MOSA compliance and future-readiness.

Step 1: System Assessment
The first phase in the redesign of the ASE simulation involved a comprehensive system assessment of the original codebase. Like many legacy systems of its era, the code lacked modern software engineering practices—there were no design patterns, and it exhibited little to no structured abstraction or modular organization. While such an assessment might have been straightforward for the original developers, the task becomes significantly more challenging when the software originates from a third party, where critical contextual knowledge, development rationale, and internal conventions are often inaccessible. This lack of insight can complicate reverse engineering efforts and increase the risk of misinterpretation during the redesign process.

The current ASE simulation system has several critical shortcomings:

- Hard-coded data values were embedded throughout the code, making updates and configuration changes difficult without altering source files.
- Global memory was used as the primary method for interfacing with other simulation components, such as navigation and communication systems. This created tight coupling and reduced the system's flexibility and reusability.
- Static memory allocation was prevalent, preventing the system from handling dynamic simulation contexts or scaling for more complex scenarios.
- No structured data types or container structures (e.g., structs or linked lists) were employed, limiting the system's ability to manage complex state information in an organized way.

These characteristics presented not only architectural challenges but also risks to long-term maintainability. Any attempt to modify the code—whether to update functionality, interface with a new system, or correct a defect—required deep understanding of its logic, which was tightly entangled and sparsely documented.

For any legacy software modernization effort, system assessment is foundational. This phase not only surfaces the technical debt embedded in old code but also helps preserve domain-specific expertise, which is often locked inside undocumented logic and interface behaviors. Skipping or rushing through this phase can result in breaking essential functionality or overlooking critical constraints that affect future integration.

Step 2: Functional Decomposition
After completing a thorough system assessment, the next step in modernizing the ASE simulation involved functional decomposition—breaking down the tightly coupled, monolithic codebase into logically distinct modules that align with operational behavior and facilitate modular design. These groupings were reorganized into modular components with clearly defined responsibilities. Examples of ASE modules are:

- Sensor Module: Simulated the behavior and characteristics of ASE sensors.
- Data Fusion Module: Consolidated sensor inputs and mission data to create a coherent situational awareness picture. This module became central to threat correlation and priority assignment.
- Threat Detection: Contained logic to identify, locate, and warn pilots of potential hostile threats.
- Warning Tones Generation Module: Contains algorithms to generate Warning Tones.

By encapsulating each functional group into objects or software components, dependencies between modules were minimized. For instance, the sensor modules could now be modified or extended without impacting the threat

evaluation logic, and vice versa. To further enhance modularity and cohesion, the Facade Pattern was used to expose simplified interfaces to complex subsystems such as sensor emulation, reducing external dependency on internal implementation details. The Adapter Pattern was employed to bridge legacy input/output formats with newer modular interfaces, enabling the system to integrate with both modern and older avionics simulations during transition. These patterns contributed to a cleaner architecture where modules could be independently developed, tested, and maintained—leading to faster upgrades and reduced integration risks.

Functional decomposition is essential for transforming legacy software into modular, standards-compliant systems. The key is to identify logical boundaries within the system—often aligned with physical subsystems, or user interactions—and use those boundaries to define modules. Even in tightly coupled legacy systems, patterns will emerge that reflect real-world operational flows. Capturing these flows and formalizing them into modules provides a natural path to encapsulation. Introducing design patterns can help enforce clean module interaction and prepare the software for evolution and reuse across programs, supporting long-term MOSA goals.

Step 3: Interface Standardization

With modular components defined, the next critical step in modernizing the ASE simulation involved standardizing the interfaces between those components and with external systems. The legacy ASE simulation relied heavily on proprietary protocols and global memory mappings to exchange data with other flight simulation subsystems such as navigation, communication, and mission computing. These methods, while functional, created tight coupling and fragile dependencies. In the redesign, these interactions were re-architected using open, well-defined Application Programming Interfaces (APIs). Each major module—sensor simulation, threat evaluation, cockpit I/O—now communicated via explicit interface contracts, with standardized input/output formats:

- Data Exchange: Mission data, threat tracks, and sensor signals were restructured to conform to a common data model derived from DoD Interface Standards (e.g., DIS, or HLA).
- API Definitions: Each module exposed a set of documented APIs, specifying allowed function calls, parameters, and expected responses—promoting clarity, extensibility, and third-party integration.

Interface standardization is a universal and indispensable step. It transforms a closed, tightly coupled system into one that is interoperable, maintainable, and ready for integration into a broader ecosystem of tools and simulators. The key takeaway is that standardizing interfaces isn't just a technical task, it is a strategic enabler. It allows organizations to decouple development timelines, reduce vendor lock-in, and prepare for evolving mission needs.

Step 4: Incremental Refactoring

After standardizing interfaces, the modernization effort focused on encapsulating legacy functionality and refactoring modules to improve structure, readability, and maintainability—while preserving existing behaviors critical to simulation fidelity. Given the system's age and operational significance, a carefully staged, iterative approach was essential to manage risk. Rather than rewriting the system from scratch—an approach prone to introducing bugs—we applied incremental refactoring, encapsulating functionality one module at a time into self-contained components. This included:

- Wrapping legacy code blocks in object-like structures or libraries with defined APIs.
- Gradually replacing global variables with well-scoped, modular data handling.
- Refactoring duplicated logic (e.g., sensor math, threat evaluation) into shared utility modules.

To reduce the risk of cascading failures, we used a "wrap-and-replace" strategy, allowing each software change to be tested in isolation. To ensure simulation accuracy was maintained throughout the modernization, we implemented a comprehensive validation process. Automated regression tests compared outputs from legacy and refactored modules to confirm identical behavior. Real-world scenarios were replayed in the updated system to validate consistent aircraft responses, and the new APIs underwent stress testing to ensure interface stability under real-time data rates and edge conditions.

One of the key strengths of this approach was that the ASE simulation remained operational throughout the refactoring process. At no point was the entire system "down for overhaul." Instead, refactored modules were gradually integrated and validated within the existing framework.

Encapsulation and refactoring are often the most time-consuming parts of modernization, but also the most essential for achieving long-term modularity and maintainability.

Step 5: Common Services & Libraries
A critical part of the ASE simulation redesign involved identifying and extracting shared functionality across the software package—particularly mathematical computations and support services—and reorganizing them into reusable, modular libraries. This step reduced code duplication, improved maintainability, and laid the foundation for interoperability with other simulation systems. The legacy ASE codebase contained a range of mathematical algorithms used for sensor modeling, vector transformations, coordinate conversions, and threat evaluation logic. These were often repeated across multiple modules, written inconsistently, and tightly coupled with unrelated logic.

As part of the redesign:

- These algorithms were identified, abstracted, and consolidated into standalone utility functions.
- They were then wrapped into well-documented libraries using consistent interfaces and naming conventions.
- Functions were parameterized to enable reuse by multiple ASE subcomponents.

By wrapping these routines as shared modules, the simulation architecture gained both modularity and traceability—two pillars of MOSA alignment.

Beyond math utilities, we created common service layers to handle:

- Input/output operations (e.g., data acquisition, time stamping, signal injection).
- Data formatting and transformation, especially converting mission data into usable runtime forms.
- Logging, timing, and simulation state management, which were previously implemented inconsistently across the codebase.

These services were moved outside of the ASE-specific codebase into a shared simulation service layer. This design made them accessible not only to ASE components but also to other simulation packages—such as navigation systems, electronic warfare modules, or radar simulations—maximizing reuse and interoperability across domains.

Identifying shared logic and services is a key turning point toward modernization. This step supports several MOSA objectives—modularity, reusability, and separation of concerns.

Step 6: Integration & Validation
The final step in the ASE simulation redesign was the systematic reintegration and validation of all modernized components within the larger flight simulation environment. After modularizing, refactoring, and standardizing individual parts of the ASE, ensuring these parts could function as a cohesive whole was essential—not only within ASE itself but also in its interactions with other avionics subsystems such as navigation, communication, and mission control. To mitigate risk and control complexity, the integration was conducted in phases:

- Module-level validation occurred first, verifying that each newly refactored module (e.g., threat detection, sensor emulation) operated correctly in isolation.
- Next, modules were integrated incrementally, using simulation stubs and test harnesses to replicate inter-module communication and timing.
- Finally, the full ASE package was tested in the context of the entire simulation suite, ensuring end-to-end functionality, correct data exchanges with the mission computer, and accurate situational awareness output on cockpit displays.

Robust validation techniques were employed at each stage:

- Regression testing confirmed that updated modules maintained expected outputs and did not introduce new errors.
- Behavioral testing was used to verify situational awareness logic under realistic mission scenarios.
- Timing and performance analysis ensured the modernized ASE met real-time constraints and responded predictably under load.

These validation efforts were informed by domain expertise and comparison with established behaviors of the legacy ASE model, creating confidence in both technical accuracy and operational fidelity.

Integration and validation are critical for ensuring that modernization efforts yield operationally viable systems. For legacy systems, this step must go beyond simple code verification—it must ensure that the refactored components function reliably in context, under the full scope of mission demands. This final step ensures that the re-structured simulation software moves from technical restructuring to mission readiness, aligning legacy systems with MOSA principles and paving the way for scalable, interoperable, and sustainable solutions across the defense simulation enterprise.

Figure 1 illustrates the redesigned ASE System Architecture, which reflects the modularized and standards-aligned framework introduced in this paper. At the foundation are the common libraries, which were extracted from the legacy ASE code and now serve as reusable software core assets supporting all ASE components. Above this layer, each ASE subsystem—RWR, MW, LWS, IRCM, CMDS, and LAIRCM—is simulated independently as a modular component. These modules interact with an Audio Threat Warning system via a dedicated API, enabling coordinated aural alerts. The architecture supports open simulation interoperability through interfaces for Distributed Interactive Simulation (DIS) and High-Level Architecture (HLA) standards. Configuration flexibility is achieved through ASE Configuration Files, which allows tailoring for specific customers and flight simulator platforms. Each module draws from its relevant data sources, including the RWR Mission Data Table, MWS User Defined Module, CMDS Mission Data Files, and LAIRCM User Defined Module. Collectively, the figure encapsulates a modernized ASE simulation ecosystem designed for reusability, scalability, and alignment with MOSA principles.

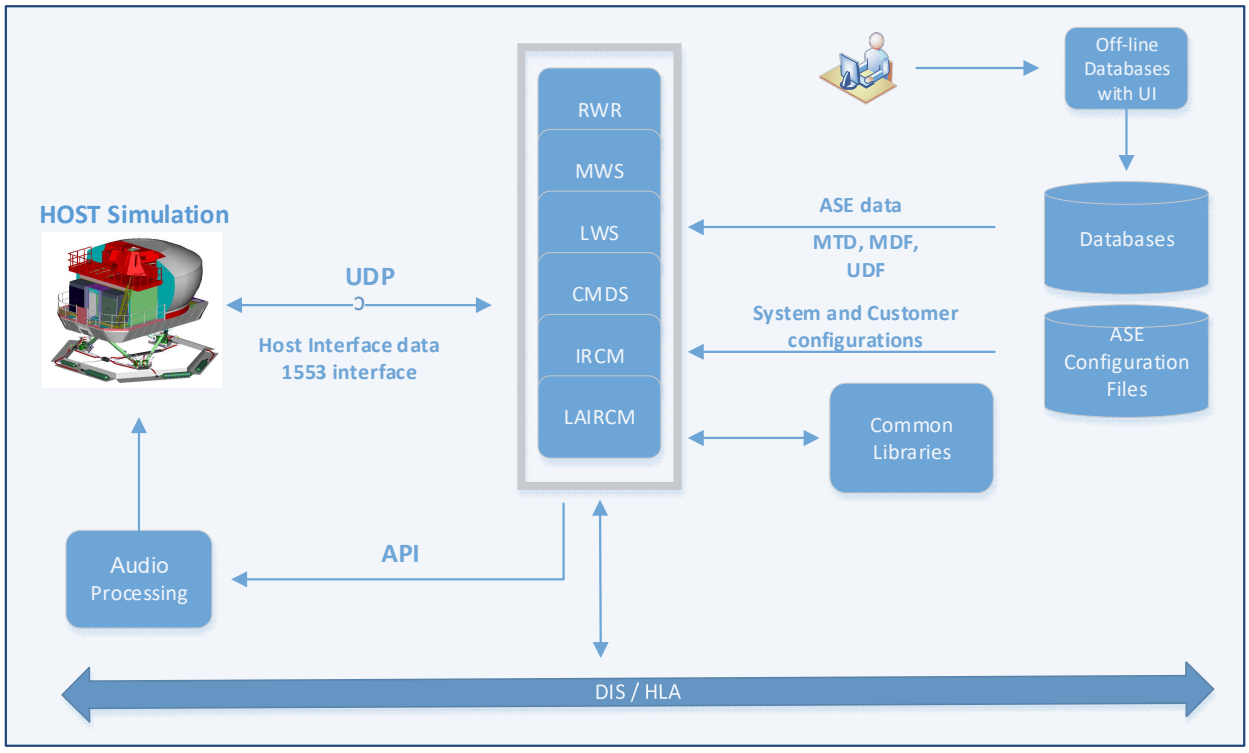

Fig 1. Redesigned ASE System Architecture illustrating modular simulation components.

V. ASSET HIERARCHY AND CONFIGURATION MANAGEMENT STRATEGY FOR ASE SOFTWARE

While the six-step blueprint outlines a process for technical redesign, long-term sustainability and configurability also require a clear organizational structure for managing software components. To support scalable development, ensure traceability, and enable reuse across multiple aircraft platforms, the redesigned ASE software adopts a **Modular Asset Hierarchy**. This hierarchy distinguishes between software core assets, system assets, and aircraft component assets, each maintained and configured under separate configuration management (CM) baselines. This asset structure not only reinforces MOSA principles of modularity and separation of concerns but also provides a practical mechanism for enabling specialized teams to collaborate across product lines while maintaining software integrity and minimizing duplication. The following section introduces this asset model and its role in supporting reuse, rapid integration, and platform-specific customization.

To enable modularity, reusability, and configuration control in alignment with MOSA principles, the redesigned ASE simulation introduces a **three-tiered software asset** structure:

1. **Software Core Assets**
   Core assets are foundational software modules that implement common functionality required by all ASE subsystems, regardless of sensor type or platform. Examples include sensor modeling, detection algorithm, threats track file, threats identification and categorization. These assets are developed and maintained in a dedicated configuration-managed (CM) repository. They are designed to be platform-agnostic and reusable and are versioned separately to allow controlled evolution over time.
2. **System Assets**
   System assets are software modules specific to a class of ASE subsystem, such as the RWR, LWR, MWS, LAIRCM, IRCM and CMDS. These assets extend and depend on the core assets and implement subsystem-specific logic, interfaces, and behaviors. System assets are also maintained in a separate CM-controlled repository from core assets, allowing system-level evolution without tightly coupling all platforms or forcing core changes.
3. **Aircraft Component Assets**
   Aircraft component assets are platform-specific ASE modules derived from system assets and configured for integration into a particular aircraft model. These assets reflect the actual hardware (e.g., AN/ALR-69 RWR), aircraft-specific integration needs (e.g., mission computer commands, control panel mapping) and custom tuning and data formats. These assets are created and maintained by product teams responsible for individual aircraft simulation packages. Teams pull from the CM-managed core and system asset repositories, integrating them into the platform's overall simulation framework.

To support seamless integration across the asset hierarchy, dedicated software interfaces were developed: a middleware API layer enables standardized communication between core assets and system assets, while an application-level API connects system assets to aircraft component assets, ensuring modular composition and adherence to MOSA integration principles.

Beyond aligning with MOSA and achieving modularity, the redesign introduced measurable improvements in sustainment and supportability. By implementing metrics to monitor update frequency, defect density, and integration stability, we were able to assess and manage software health more proactively. The separation of core, systems and aircraft assets enabled targeted updates and reduced regression risks, minimizing the need for broad revalidation. Most notably, this modular approach reduced the cost of non-quality—defined as the engineering effort spent without delivering additional functional or quality improvements (i.e. rework effort)—by over 50%. This gain translates to more focused development, reduced rework, and lower lifecycle costs.

Figure 2 illustrates the hierarchical asset structure of the RWR sub-system, showcasing how simulation software assets are logically organized into tiers of reusability and specificity. At the top level, core assets represent reusable modules and libraries that are common across multiple platforms and sub-systems—such as sensor modeling, threat detection, threats track files, and threat identification and categorization. These components are independent of any specific platform or mission context and are intended to serve as foundational building blocks for the ASE simulation. The second tier, system assets, includes those components that are unique to a specific system (e.g. RWR)—such as RWR Sensors, Radar Detection, RWR threats prioritization, RWR Threat Display Rules, etc. Finally, aircraft component assets form the most specialized layer, containing ASE platform-specific configurations (e.g., AN/ALR-69), mission data sets, and interface adapters tailored to a particular aircraft or simulator implementation. While Figure 2 shows only a few representative building blocks for each asset class for illustration purposes, it is important to note that each class may encompass numerous components depending on the system's complexity and simulation requirements. This hierarchical approach not only improves software modularity and maintainability but also enables more efficient integration across different aircraft models and simulation environments. Figure 2 only focuses on the RWR simulation system, but this structure is replicated across other ASE sub-systems (e.g., MWS, LWS, IRCM, CMDS, LAIRCM) and is extensible to broader simulation ecosystems, supporting scalable development and reuse across the flight training domain.

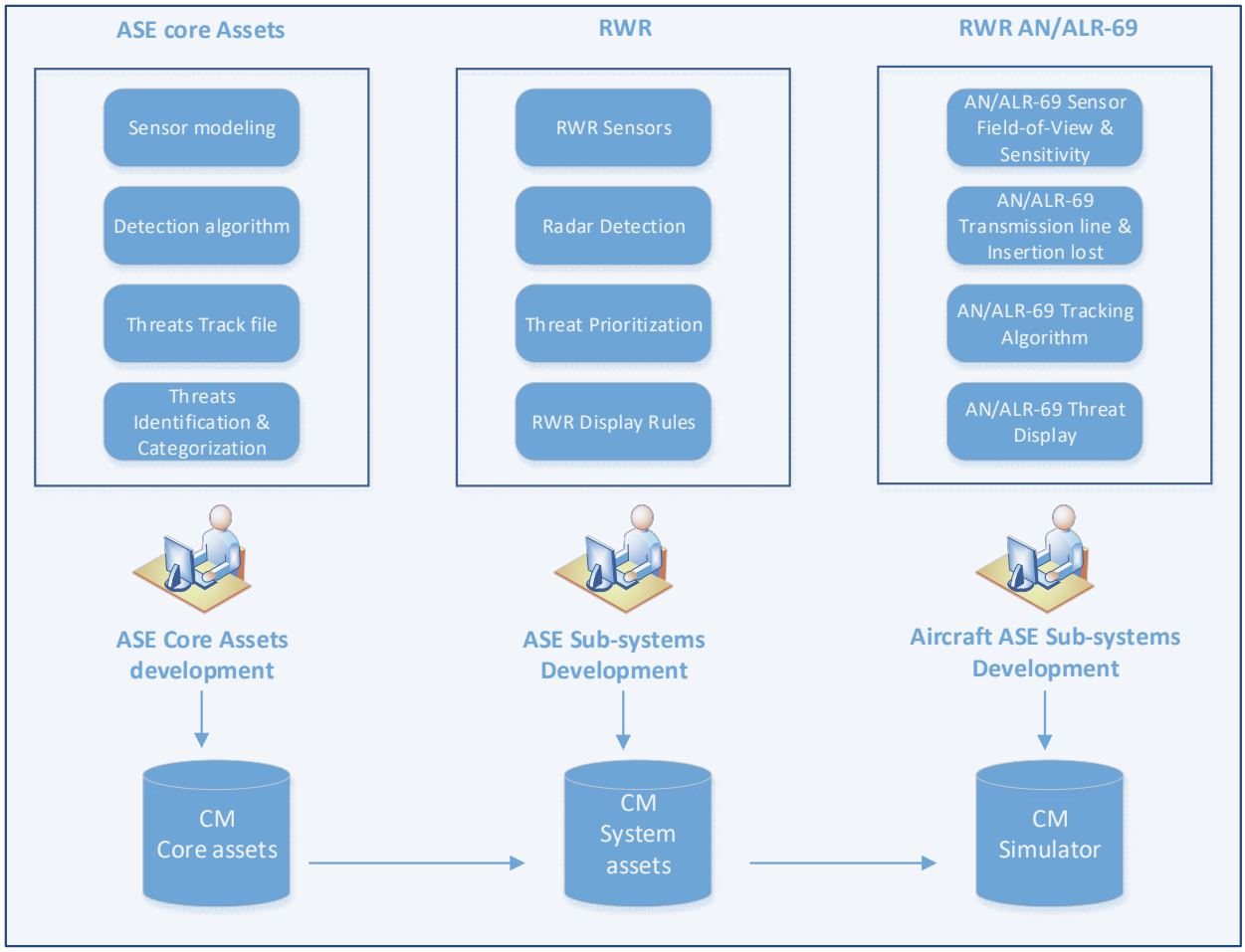

Figure 2. RWR Simulation Hierarchy Software Assets for enabling specialized teams to collaborate across product lines while maintaining software integrity and minimizing duplication

VI. LESSONS LEARNED

The modernization of the ASE simulation provided critical insights into the complexities and rewards of aligning legacy flight simulation software with MOSA principles. The following lessons highlight both challenges and effective strategies that emerged from this effort.

Balancing Legacy and Modernization

One of the primary challenges was balancing the need to introduce modularity without compromising the integrity of proven legacy functionality. The original ASE simulation, though monolithic and outdated in structure, had undergone years of validation and delivered consistent performance. To preserve this reliability while enhancing modularity, key algorithms—especially those related to sensor processing and threat detection—were carefully isolated and reused within new modular components. Reusable modules were built around these validated pieces, minimizing risk while transitioning to a modern architecture.

Domain Expertise Preservation

Throughout the redesign process, close collaboration with subject-matter experts (SMEs) was crucial. Much of the ASE simulation's domain knowledge—particularly the logic behind sensor behavior and mission-level interactions—was embedded directly in the original code. Without active SME involvement, there was a significant risk of losing critical nuances that influence simulation fidelity. To mitigate this, structured knowledge transfer sessions were held between software engineers and domain experts, ensuring the transition preserved both technical accuracy and operational realism.

MOSA Compliance Challenges
Achieving compliance with MOSA standards proved to be both a technical and interpretive challenge. Some legacy hardware dependencies and closed, proprietary interfaces were not easily abstracted or replaced. Additionally, interpretations of data standardization varied, complicating integration efforts. To address these issues, wrapper interfaces were developed to bridge incompatible protocols, and compliance was approached in phased increments—beginning with high-impact areas like interface standardization before moving into deeper architectural adjustments. However, the introduction of wrapper interfaces introduced the potential for execution performance degradation, making it critical to ensure that the simulation continued to meet real-time requirements.

Scalability and Adaptability
The redesign demonstrated measurable improvements in the system's scalability and adaptability. By separating core, system, and component-specific assets, the ASE simulation can now be tailored to various aircraft configurations with minimal duplication of effort. This adaptability supports faster deployment of new system variants and reduces long-term technical debt. While the modular approach required significant up-front investment in architecture design and refactoring, the long-term benefits in maintainability and reuse are expected to outweigh initial costs.

Performance and Fidelity Trade-offs
The shift to a modular, MOSA-compliant architecture introduced initial latency due to added abstraction layers. This was mitigated through performance tuning techniques such as optimized data access, streamlined middleware communication, and efficient memory management. While modularization required some generalization of code, fidelity-critical logic was retained in system-specific layers to preserve simulation accuracy. Ultimately, the redesign maintained real-time performance and high fidelity while enabling long-term scalability and maintainability.

Configuration Management (CM) Evolution
Modernizing the ASE simulation required rethinking traditional configuration management practices. The introduction of software core assets, system assets, and aircraft component assets necessitated separate CM repositories, each with tailored version control and change tracking. This modular CM structure enabled better traceability, facilitated parallel development across teams, and supported reuse across multiple platforms. Tools and processes were updated to manage dependencies and enforce alignment with MOSA principles, ensuring each asset could evolve independently while maintaining system integrity. This evolution in CM proved essential for sustaining long-term maintainability and scalability.

Performance Metrics for Development, Integration, and Sustainment
The modernization of the ASE simulation enabled the collection of concrete performance metrics across the software lifecycle. Preliminary data indicated a 20% increase in development velocity, attributed to modular design and reusable components. Integration metrics showed a 25% reduction in integration time, thanks to standardized interfaces and encapsulated modules. Most notably, sustainment metrics revealed a 30% decrease in maintenance costs, as clearer architecture and decoupled components simplified troubleshooting and upgrades. These metrics were tracked using development dashboards integrated with version control and issue-tracking systems, and visualized through burndown charts, integration timelines, and maintenance effort trendlines. Additionally, the adoption of a three-tiered software asset structure significantly enhanced our CI/CD pipeline, enabling more flexible and reliable integration of the ASE across multiple simulation platforms. Data showed that deployment effort was reduced by approximately 50%, as platform-specific adjustments could be isolated and managed independently within the tiered structure. This data-driven approach reinforced the tangible value of MOSA-aligned modernization in improving engineering efficiency and lifecycle performance.

Blueprint Validation
This effort validated the six-step blueprint proposed in this paper. Notably, the modularization and interface standardization directly improved interoperability with other simulation domains, while the separation of data and logic enabled faster updates and reduced maintenance overhead. These results affirm the blueprint's potential as a generalized framework for modernizing other legacy systems, especially those used in high-fidelity training environments.

VII. CONCLUSION

The persistent use of legacy flight simulation software presents a formidable challenge in the context of the Department of Defense's (DoD) push toward Modular Open Systems Approach (MOSA) compliance (NDIA, 2020). While these systems offer proven performance and reliability, their monolithic design, tightly coupled architectures, and proprietary interfaces hinder adaptability, scalability, and integration with modern tools and technologies.

This paper presented a case study on the redesign of the Aircraft Survivability Equipment (ASE) simulation package—originally developed in the 1980s—as a representative example of the broader issues encountered in modernizing legacy simulation systems. Through a systematic and disciplined redesign process, the legacy codebase was decomposed into modular components, standardized interfaces were introduced, and reusable libraries were created to encapsulate common algorithms. These steps aligned the system with key MOSA principles, including modularity, interoperability, and maintainability, while preserving the domain-specific expertise embedded in the original software. Importantly, the methods applied in this effort serve as a blueprint—a repeatable and adaptable framework that can guide similar modernization initiatives across a wide variety of simulation systems and legacy software packages. While this work focuses on the flight simulation domain, exemplified by the ASE software, the principles and strategies outlined here are not limited to flight simulation applications. The same framework can be extended to other simulation disciplines such as medical training, aircraft maintenance, synthetic environments, and more. Moreover, its modular and scalable structure holds promise for real-world software systems beyond simulation, where maintainability, modularity, and interoperability are equally vital. This structural approach enables defense organizations and contractors to transform entrenched systems without discarding valuable historical knowledge or starting from scratch. This blueprint is offered as a foundational guide to help the defense software community systematically modernize legacy systems in alignment with MOSA. As defense software evolves, such structured approaches will become essential to scaling innovation and sustaining technical superiority. By translating strategic MOSA principles into practical software engineering actions, this blueprint offers the defense software community a reliable path forward for extending the relevance of legacy assets while preparing for future system evolution.

Beyond technical alignment, the broader implications of this transformation are significant. Systems modernized under MOSA principles enable cost-effective sustainment, support rapid upgrades, and foster technological innovation—factors critical for maintaining readiness in rapidly evolving defense environments. By standardizing interfaces and clearly delineating software asset boundaries, the framework also enables vendor-neutral procurement, reducing proprietary lock-in and promoting competitive acquisition strategies—core objectives of MOSA-driven training system acquisitions. As defense systems increasingly demand agility, interoperability, and sustainability, the need to modernize legacy software becomes not only a technical imperative but a strategic one. Organizations are encouraged to adopt and tailor this blueprint as they prepare their simulation environments to meet the demands of future operations and compliance with evolving standards. Doing so ensures continued operational relevance while unlocking the benefits of modular, future-ready software architectures.

REFERENCES

Alec K., David B., Kyle G., David J., Stewart H. (2024). Modernizing Legacy Defense Systems to be Compliant with Open Architecture Standards. 2024 NDIA Michigan Chapter Ground Vehicle Systems Engineering and Technology Symposium. 1-8.

Baldwin C, Clark K. B. (2000). Design rules: The power of modularity. Cambridge, MA: MIT Press, 2000.

Davendralingam N., Guariniello C., Tamaskar S., DeLaurentis D., Kerman M. (2019). Modularity research to guide MOSA implementation. The Journal of Defense Modeling and Simulation, Vol. 16(4), pages 389-401.

Department of Defense (2020). DOD Instruction 5000.88. Engineering of Defense Systems.

House Committee on Armed Services (2016). Report 114-102 (2016). Accompanying National Defense Authorization Act (NDAA) FY 2016. House Committee on Armed Services (HASC), December 2015.

Modular Open Systems Working Group (MOSWG) Meeting. Slides 25-34. February 2022.

National Defense Industrial Association (2020). "Modular Open System Architecture Considerations Impacting Both Acquirer and Supplier Adoption." White paper. Arlington, Va.: NDIA, 2020.

National Defense Industrial Association (2023). Modular Open Systems Approach - Implementation Challenges and Opportunities. National Defense Industrial Association Systems Engineering Architecture Committee.

Newcomb PJ, Bras B, Rosen DW (1998). Implications of modularity on product design for the life cycle. J Mech Des 1998; 120(3): 483–490.