DOT/FAA/TC-20/42

Federal Aviation Administration
William J. Hughes Technical Center
Aviation Research Division
Atlantic City International Airport
New Jersey 08405

Model-Based Systems Engineering and Model-Based Safety Analysis: Final Report

January 2021

Final report

U.S. Department of Transportation
Federal Aviation Administration

Technical Report Documentation Page

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

1. Report No. DOT/FAA/TC-20/42	2. Government Accession No.	3. Recipient's Catalog No.	
4. Title and Subtitle Model-Based Systems Engineering and Model-Based Safety Analysis -Final Report		5. Report Date January 2021	
		6. Performing Organization Code	
7. Author(s) Daniel Fogarty, Pete De Salvo and Edward DeMello		8. Performing Organization Report No.	
9. Performing Organization Name and Address BOEING AEROSPACE OPERATIONS, INC. 6001 S AIR DEPOT OKLAHOMA CITY, OK 73135- 6601		10. Work Unit No. (TRAIS)	
		11. Contract or Grant No. 692M15-18-C-00015	
12. Sponsoring Agency Name and Address U.S. Department of Transportation, Federal Aviation Administration Air Traffic Organization Operations Planning Office of Aviation Research and Development Washington, DC 20591		13. Type of Report and Period Covered Final Report	
		14. Sponsoring Agency Code AIR-134	
15. Supplementary Notes			

16. Abstract

Aircraft manufactures and modifiers are developing highly integrated and distributed system architectures with multiple aircraft functions. System architectures and associated requirements for aerospace digital avionics systems have experienced acceleration in complexity and integration over the last two decades. Where initial generations of digital avionics automated individual functions that were often stand-alone or limited in integration with other airplane-level functions, today's complex avionics architectures can be highly integrated across complex systems.

The purpose of this report is to develop recommendations that assist in the certification and assurance of highly integrated and distributed digital aircraft systems, including systems development processes and architectures, requirements validation and integration, new and novel electronic hardware and software implementation techniques, tools, methods, and processes, continued operational safety and streamlining approaches for safety assurance and aircraft certification.

The research identified the potential benefits of using Model-Based Systems Engineering (MBSE) and Model-Based Safety Analysis (MBSA) for highly integrated systems architectures. It determined that the existing guidance/standards, while they could potentially be improved, are acceptable.

The major impediment to successful large-scale implementation of MBSE and MBSA is not related to improving existing industry guidance/standards. The biggest impediments are the lack of common data standards and tool capabilities. Data standards are the rules by which data is described and recorded. In order to share, exchange, and understand data, the format, as well as the meaning, must be standardized. Data standards are the foundation for data interchange/exchange/language consistency.
The research included a recommended path to achieve data standards, which in turn would help with tool development and compatibility. The research also included recommended training curriculum to implement MBSE and MBSA.

17. Key Words Requirements, validation, verification, safety, development assurance, ARP4754A, ARP4761, DO-178B/C, DO-254, DO-297.		18. Distribution Statement	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 214	22. Price

ACKNOWLEDGEMENTS

The authors would like to acknowledge the following Federal Aviation Administration Review Team individuals for providing support to the project:

- Barbara Lingberg (Sponsor)
- Srini Mandalapu (Project Manager)
- George Romanski (Chief Scientific Technical Advisor for aircraft computer software)
- Mike Vukas
- Richard Spencer

Contents

Figures

Tables

Acronyms

Acronym	**Definition**
3D	Three-dimensional
AC	Advisory Circular
ADM	Air Data Module
AEH	Airborne Electronic Hardware
AFHA	Airplane Functional Hazard Assessment
AIMS	Aircraft Information Management Systems
AIR	Aerospace Information Report
AR	Authorized Representative
ARINC	Aeronautical Radio, Incorporated
ARP	Aerospace Recommended Practice
ASIC	Application Specific Integrated Circuits
ASME	The American Society of Mechanical Engineers
ATC	Amended Type Certificate
BCA	Boeing Commercial Airplanes
BITE	Built-In Test Equipment
CAN	Controller Area Network
CDN	Common Data Network
CEA	Cascade Effects Analysis
CMA	Common Mode Analysis
CONOPS	Concept of Operations
DAL	Design Assurance Level
DFD	Data Flow Diagram
DIMA	Distributed Integrated Modular Avionics
DoD	Department of Defense
DOE	Design of Experiments
ECL	Electronic checklist
ECS	Environmental Control System
ESR	Engineering Safety Review
FAA	Federal Aviation Administration
FDAL	Functional Development Assurance Level
FFBD	Functional Flow Block Diagram
FHA	Functional Hazard Assessment

Acronym	**Definition**
FMEA	Failure Modes and Effects Analysis
FOX	Fiber Optic Translator
FPM	Failure Propagation Model
FTA	Fault Tree Analysis
GG	Graphic Generator
GPM	General Processing Module
HF	High Frequency
HIRF	High Intensity Radiated Field
ICD	Interface Control Document
IDAL	Item Development Assurance Level
IMA	Integrated Modular Avionics
INCOSE	International Council on Systems Engineering
LRM	Line-Replaceable Module
LRU	Line-Replaceable Unit
LVC	Live, Virtual, Constructive
MBD	Model-Based Design
MBSA	Model-Based Safety Analysis
MBSE	Model-Based Systems Engineering
MBE	Model-Based Engineering
MEL	Minimum Equipment List
NASA	National Aeronautics and Space Administration
NextGen	Next Generation Air Transportation System
NIST	National Institute of Standards and Technology
O&M	Operations and Maintenance
OMG	Object Management Group
PASA	Preliminary Airplane Safety Assessment
PCM	Power Conditioning Module
PLD	Programmable Logic Devices
PR	Problem Report
PRA	Particular Risk Assessment
PSSA	Preliminary System Safety Assessment
RAA	Resource Assurance Analysis
RBT	Requirements-Based Test
RDC	Remote Data Concentrator

Acronym	**Definition**
RFLP+B	Requirements, Functional, Logical, Physical + Behavior
ROI	Return on Investment
S&MF	Single and Multiple Failure
SAE	Society of Automotive Engineers
SE	Systems Engineering
SFHA	System Functional Hazard Assessment
SIL	Systems Integration Laboratory
SLOC	Software Lines of Code
SME	Subject Matter Expert
SoS	System of Systems
SSA	System Safety Assessment
STC	Supplemental Type Certificate
SysMBD	System Model-Based Design
SysML	Systems Modeling Language
T&E	Test and Evaluation
TC	Type Certificate
UML	Unified Modelling Language
V&V	Verification and Validation
VM	Virtual Machine
ZSA	Zonal Safety Analysis

Executive summary

Aircraft manufactures and modifiers are developing highly integrated and distributed system architectures with multiple aircraft functions. System architectures and associated requirements for aerospace digital avionics systems have experienced acceleration in complexity and integration over the last two decades. Where initial generations of digital avionics automated individual functions that were often stand-alone or limited in integration with other airplane-level functions, today's complex avionics architectures can be highly integrated across complex systems.

The purpose of this report is to develop recommendations that assist in the certification and assurance of highly integrated and distributed digital aircraft systems, including systems development processes and architectures, requirements validation and integration, new and novel electronic hardware and software implementation techniques, tools, methods, and processes, continued operational safety and streamlining approaches for safety assurance and aircraft certification.

The research objectives included the following:

- Assess current guidance and standards for Integrated Modular Avionics (IMA) systems, Distributed Integrated Modular Avionics (DIMA) systems, and other distributed systems and architectures.
- Identify the issues, challenges, and gaps in the current standards/guidance.
- Assess the latest solutions, tools, and technologies.
- Develop criteria, mitigation approaches, recommendations, training material, and position papers.

The research identified the potential benefits of using Model-Based Systems Engineering (MBSE) and Model-Based Safety Analysis (MBSA) for highly integrated systems architectures. It determined that the existing guidance/standards, while they could potentially be improved, are acceptable.

The major impediment to successful large-scale implementation of MBSE and MBSA is not related to improving existing industry guidance/standards. The biggest impediments are the lack of common data standards and tool capabilities. Data standards are the rules by which data is described and recorded. In order to share, exchange, and understand data, the format, as well as the meaning, must be standardized. Data standards are the foundation for data interchange/exchange/language consistency.

The research included a recommended path to achieve data standards, which in turn would help with tool development and compatibility. The research also included recommended training curriculum to implement MBSE and MBSA.

1 Introduction

Initial generations of digital avionics systems architectures were often designed with stand-alone or limited functional integration with other systems. Over time, systems architectures have become more integrated, and are progressing to even higher degrees of integration. This is particularly true for digital systems architectures involving Integrated Modular Avionics (IMA) and Distributed Integrated Modular Avionics (DIMA).

The benefits of IMA/DIMA architectures include the following: increased performance and functionality, weight reduction (due to reduced wiring and Line Replaceable Units (LRUs), system stability, more efficient modular upgrades, and other benefits.

The benefits of highly-integrated systems architectures also introduce challenges managing increased orders of magnitude in digital data. A single interface in a legacy federated architecture can now be translated into several intermediate interfaces in an IMA/DIMA architecture. In addition, the data required to detail all the interfaces in an IMA/DIMA architecture is significantly greater than that needed for a single-system federated interface. As a result, there is a greater number of cross-functional interfaces between different systems; there are more interfaces that are highly interrelated. As a result, there is an increased engineering effort to produce, analyze, validate, and verify that the systems are going to behave as expected with no anomalous behavior.

Regardless of an aircraft's systems architecture, the flying public expects that the level of safety will be maintained or improved. Increasing system complexity and dependency could potentially suggest that industry standards, guidance, and practices may need to evolve to ensure that safety and development assurance processes maintain the excellent safety records that aviation has enjoyed to date. Industry is developing model-based systems engineering (MBSE) and model-based safety analysis (MBSA) processes and guidance to address these challenges and potential gaps in existing standards. MBSE and MBSA are being developed to help improve the efficiency of managing the data associated with increasingly integrated systems. They are not required to implement a safe systems architecture.

In particular, evolution of industry standards, guidance, and practices should consider the process of validating and verifying a system of components at the aircraft level. Finally, the evolution of industry standards should consider the future implementation of the Next Generation Air Transportation System (NextGen), which introduces each aircraft as a node within a broader system of systems (SoS). The same architecture trends and integration challenges facing the integrated NextGen environment parallel those occurring within commercial airplanes and other products. MBSE and MBSA processes and tools could potentially be extended to help with validation and verification for NextGen systems.

This document summarizes the approach that will be used to identify potential gaps and improvement recommendations as digital systems continue to evolve and grow in complexity and integration, with the goal of maintaining the excellent safety record that industry has realized to date.

1.1 Research scope

The following objectives are identified as successful outcomes of this study, directly benefiting the Federal Aviation Administration (FAA) and industry:

- Identifying potential improvements in the current guidance, particularly related to analyzing and evaluating cumulative airplane-level effects from failure conditions and validating that a sufficient level of safety will be maintained.
- Examining how MBSE/MBSA can be one means to address existing gaps in current standards/guidance (at the integrated platform level).
- Developing recommendations that assist in the certification and development of highly integrated and distributed digital aircraft systems.

In order to achieve these goals, the work will include the following (detailed in section 2):

- Assessing current guidance and standards for IMA systems, DIMA systems, and other distributed systems and architectures.
- Identifying issues, challenges, and gaps in current standards/guidance related to development assurance (which includes minimizing the safety risk for undetected errors). This includes examining Boeing's experiences from applying current guidance and standards in the certification of highly integrated, distributed systems architectures, and then making recommendations on future improvements.
- Assessing the latest solutions, tools, and technologies.
- Developing criteria, mitigation approaches, recommendations, training material, and/or position papers.

1.2 Research approach, activities, and principal results

The research approach, as shown in Table 1, for this study is divided into six different White Papers:

Table 1. Research approach

Research Objectives	**Detailed Work Description**
Assess current guidance and standards for IMA systems, DIMA systems, and other distributed systems and architectures.	**White Paper 1 (Section 1, 2, 3 and 4)** Analyze existing industry processes, survey industry committee members responsible for guidelines around development, validation, and verification of highly integrated, complex digital systems: Identify existing industry guidelines for requirements definition, verification and validation (V&V) processes, systems integration, and change impact analysis. Identify requirements for regression analysis and test. Identify potential shortcomings in current processes, particularly related to section 4.6.4 (Aircraft/System Integration) and section 6 (Modifications to Aircraft or Systems) of Society of Automotive Engineers (SAE) Aerospace Recommended Practice (ARP) 4754A [1]. Identify integral systems integration process gaps related to safety that are not currently part of ARP 4754A and ARP4761 [2]. Identify common errors which occur in the interrelationships between process steps in ARP 4754A, RTCA Document (DO)-178 [3], DO-254 [4], and DO-297 [4], particularly those that could result in incomplete and incorrect requirements and/or systems integration issues for IMA architectures. Identify potential shortcomings in DO-331 [6] (Model-Based Development and Verification Supplement to DO-178C and DO-278A). Identify modeling standards. **White Paper 2 (Section 5)** Conduct analysis on process execution problems: Survey Boeing's Subject Matter Experts (SMEs), including those who have work experiences as Authorized Representatives (ARs). The SMEs will have experiences across multiple programs, multiple design disciplines, and multiple suppliers. Identify potential gaps in the development/design assurance processes for development programs (which are addressed by ARP 4754A [1]) Assess frameworks for highly integrated, distributed platform systems.

Research Objectives	**Detailed Work Description**
	Identify gaps in existing V&V processes for highly integrated, distributed platform systems. Analyze integration related problem reports and determine root causes. Investigate how a potential use of virtualization, design of experiments, and distributed test could mitigate integration and safety issues for process execution problems.
Identify the issues, challenges, and gaps in the current standards/guidance	**White Paper 3 (Section 6)** Identify potential safety issues related to verification and validation during development and change impact analysis of highly integrated complex digital systems. Identify gaps in the design/development assurance processes, used to verify and validate integrated complex digital systems, and identify any potential failures and failures effects resulting from such complex systems integration for highly integrated, distributed platform systems. Ensure common understanding of system design model definition. Model used to express architectural, behavioral, performance, and interface requirements of the system being modeled. System Design Model can be analyzed to predict and understand the modeled system's capabilities and operational quality attributes (e.g., performance, reliability, safety, security). System Design Model is typically technology independent, where the requirements expressed by the model may be allocated to either hardware and/or software. Identify Model-Based Design (MBD) processes, where executable models (System Design Model) are developed at the system level for behavioral evaluation and defining requirements for software process. System development process. Requirements development. Requirements validation. System-level verification. Identify MBD processes to integrate between system and software level models. Identify MBD processes to integrate between system and airborne electronic hardware models. Identify how to conduct model coverage analysis. Identify integration touchpoints of MBD with V&V processes.

Research Objectives	Detailed Work Description
Assess the latest solutions, tools, and technologies	**White Paper 4 (Section 7)** Identify state of the art for MBSE and MBSA. Identify how MBSE/MBSA implementation experience has shown that errors and omissions, with potential of adverse effects on safety, can be found and resolved earlier in the design cycle and with change impact analysis, resulting in less effort and disruption to industry and regulatory agencies than with traditional methods. Identify how MBSE/MBSA techniques can be one means of addressing potential gaps in existing guidance related to redundancy and capability in IMA architectures to enable more efficient means of achieving product safety standards. Identify required functionality needed for MBSE/MBSA tools to be effective (tool agnostic). Investigate how model-based design approaches, using formalisms, could mitigate integration and safety issues for process execution problems. Identify MBSE and MBSA approaches to design that emphasize the injection of additional formalism into preexisting design processes. These formalisms can be introduced at a variety of levels of abstraction of design. At each level, the formalisms enable a number of new analytic capabilities: Analyses of individual systems (e.g., demonstrating their conformance to requirements). Analyses of interactions between systems (e.g., demonstrating their conformance to communication protocol requirements). Analyses of the emergent behavior created by the combination of individual systems into a greater SoS (e.g., demonstrating properties of the SoS given properties of the systems themselves). Identify integration touchpoints of MBSE/MBSA with V&V processes.
Develop criteria, mitigation approaches, recommendations, training material, or position papers	**White Paper 5 (Section 8)** Identify the prerequisites/criteria before MBSE and MBSA can be applied. Identify the appropriate levels of abstraction of the design where additional formalism can be applied. Appropriate formalisms must be chosen for each of these target abstraction levels. Once the appropriate formalisms have been selected, it must be possible to obtain the required information, in order to construct the enhanced models at each level. The more complete the formalism, the greater detail that is required.

Research Objectives	Detailed Work Description
	Identify the limitations/challenges that these prerequisites introduce: Consistency among the models at different levels of abstraction can be both necessary and difficult to obtain. The degree to which the representations of the various models are substantially equivalent can more easily satisfy this need. Assessing the completeness of any particular model can be difficult. This limitation can be eased somewhat to the degree that the models themselves become the requirements for the system. Verifying the correctness of the individual models themselves can be challenging. More formal representations carry more opportunities for guaranteeing at least the consistency of the model.
Develop criteria, mitigation approaches, recommendations, training material, or position papers	**White Paper 6 (Section 9)** Identify potential additional training material. Identify proposed modifications to existing guidance/standards.

White Paper 1 activities included a review of potential process issues and shortcomings via review of the following industry process documents:

1. SAE ARP 4754A/EUROCAE ED-79A, "Guidelines for Development of Civil Aircraft and Systems," December 21, 2010, covering development assurance processes.
2. SAE ARP 4761, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems," 1996, describing safety assessment processes.
3. DO-178B/C, "Software Considerations in Airborne Systems and Equipment Certification," RTCA Inc., Washington, DC, 2001, covering software design assurance processes.
4. DO-254, "Design Assurance Guidance for Airborne Electronic Hardware," RTCA Inc., Washington, DC, April 19, 2000, covering airborne electronic hardware design assurance processes.
5. DO-297, "Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations," RTCA Inc., Washington, DC, November 8, 2005, covering integrated modular avionics.
6. DO-331, "Model-Based Development and Verification Supplement to DO-178C and DO-278A", RTCA Inc., Washington, DC, December 13, 2011, covering use of model-based

development and verification in the software lifecycle for software that is produced in accordance with DO-178C.

7. SAE ARP 4754/EUROCAE ED-79, “Certification Considerations for Highly Integrated or Complex Aircraft Systems,” 1996, likewise covering development assurance processes.
8. SAE AIR6110, “Contiguous Aircraft/System Development Process Example,” February 5, 2020, covering a contiguous example of the aircraft and systems development for a fictitious aircraft design.
9. SAE AIR6218, “Constructing Development Assurance Plan for Integrated Systems,” October 22, 2018, covering the crucial elements to be considered when constructing the development assurance plans as described in Chapter 2 of ARP 4754A.
10. Advisory Circular 20-174, “Development of Civil Aircraft and Systems,” Sept 30, 2011, recognizes ARP 4754A as an acceptable method for establishing a development assurance process.
11. Advisory Circular 20-170, “Integrated Modular Avionics Development, Verification, Integration and Approval Using RTC/DO-297,” October 28, 2010, covering guidance on how to obtain FAA airworthiness approval for the development, verification and integration of an integrated modular avionics systems for installation into an aircraft or engine.
12. Advisory Circular 20-115D, “Airborne Software Development Assurance Using EUROCAE ED-12 and RTCA DO-178,” July 21, 2017, covering acceptable means of compliance for the software aspects for airborne systems and equipment.
13. Advisory Circular 20-152, “Design Assurance Guidance for Airborne Electronic Hardware,” July 5, 2005, covering acceptable means to gain FAA approval for showing the equipment design is appropriate for its intended function.

2 Continual evolution of integrated architectures

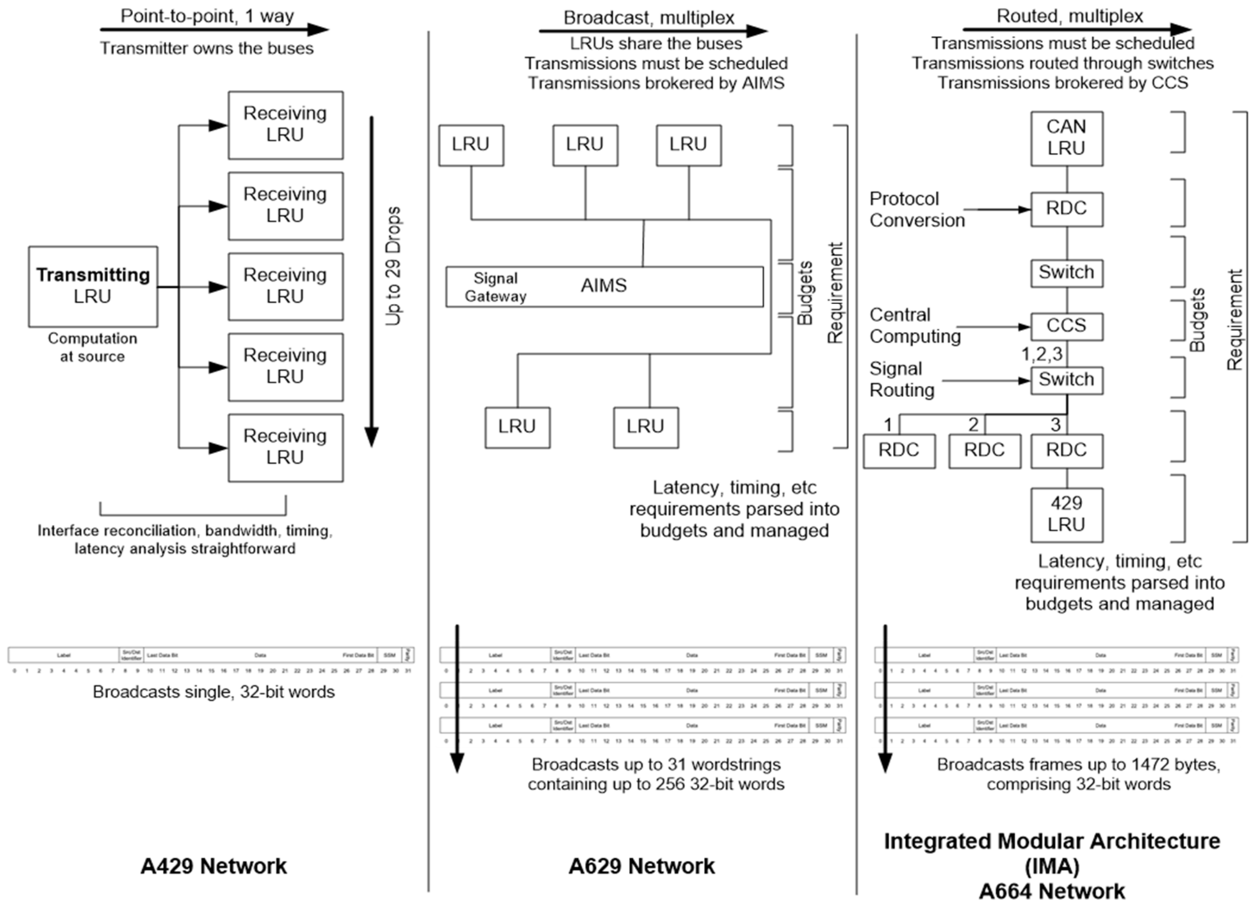
Because of improved performance, functionality, and automation, there has been a trend toward integrated architectures throughout the aerospace industry. Using Boeing as an example, the 777 airplane introduced the first centralized computing system to Boeing Commercial Airplanes (BCA), based on the Aeronautical Radio, Incorporated (ARINC) 629 protocol. The 787 airplane incorporated IMA, based on the ARINC 664 protocol (Ethernet).

Figure 1 shows the evolution of avionics architectures with different ARINC protocols.

ARINC 429 protocol uses point-to-point unidirectional buses. If bi-directional communication is required, there will be two buses. Most LRUs execute computing functions locally. Signals are transmitted from the broadcaster to all receivers.

ARINC 629 protocol uses multiplex buses, two-way communications, and multiple transmitters per an individual bus. There is some computing centralized within an Aircraft Information Management Systems (AIMS) (which also routes signals). Signals are broadcast to every LRU connected to the bus. Bus schedules are required to avoid signal clashes.

ARINC 664 protocol uses multiplex buses, two-way communications and multiple transmitters per individual bus. The computing is centralized. Most LRUs gather raw data that are processed by centralized computing functions. Message traffic is directed to subscribers through routers (switches); signals (messages) are not broadcast over the entire network.



Point-to-point, 1 way
Transmitter owns the buses
Transmitting LRU
Computation at source
Receiving LRU
Up to 29 Drops
Interface reconciliation, bandwidth, timing, latency analysis straightforward
Broadcasts single, 32-bit words
A429 Network
Broadcast, multiplex
LRUs share the buses
Transmissions must be scheduled
Transmissions brokered by AIMS
LRU
Signal Gateway
AIMS
Budgets
Requirement
Latency, timing, etc requirements parsed into budgets and managed
Broadcasts up to 31 wordstrings containing up to 256 32-bit words
A629 Network
Routed, multiplex
Transmissions must be scheduled
Transmissions routed through switches
Transmissions brokered by CCS
CAN LRU
Protocol Conversion
RDC
Switch
Central Computing
CCS
1,2,3
Signal Routing
1
2
3
429 LRU
Latency, timing, etc requirements parsed into budgets and managed
Broadcasts frames up to 1472 bytes, comprising 32-bit words
Integrated Modular Architecture (IMA) A664 Network

Figure 1. Commercial airplane digital networks evolution

Initial BCA digital networks incorporated the ARINC A429 protocol. The 777 airplane introduced the first centralized computing system to BCA, based on the A629 protocol. The 787 airplane incorporated IMAs, based on the ARINC A664 protocol (Ethernet).

Each part of the digital network's evolution has increased performance, allowing more information to be transmitted and received more efficiently. However, this evolution has also increased the functional integration efforts to ensure that the system is functionally integrated.

Software interfaces are not as "transparent" as mechanical interfaces. In addition, most complex system interfaces have inherent programmatic and contractual complexity, as they could potentially cross multiple hardware and software suppliers. Using the number of signals and software lines of code (SLOC) as rudimentary metrics for increased integration, Figure 2 shows the trends across several aircraft models over the past decades.

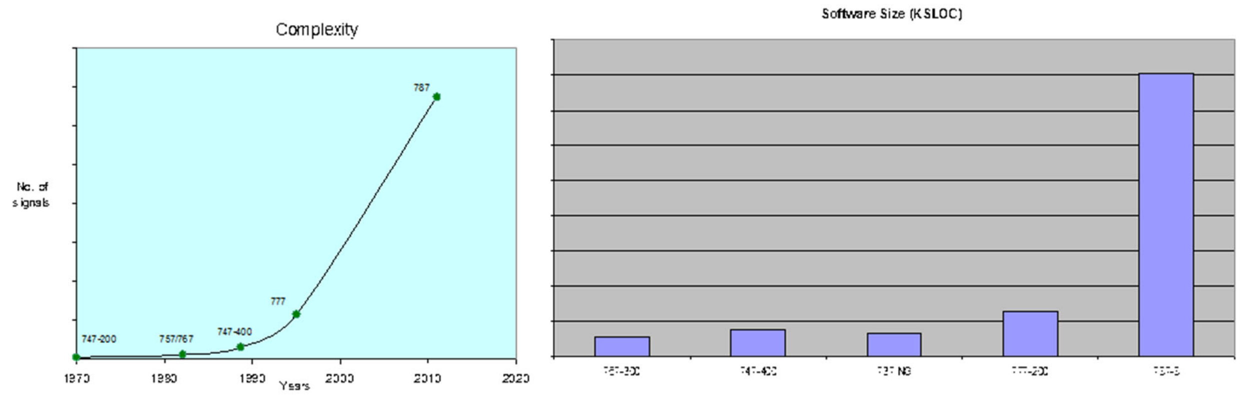


Figure 2. Integrated, distributed systems architectures

Figure 3 provides a practical example of how increased integration manifests itself. Both architectures are implementing the same function: a hydraulic pump was started with a button press on the flight deck overhead panel, and the transition was indicated with a message and with a light on the panel switch.

In the federated system (labeled "Legacy Architecture" in the left-hand portion of Figure 3), a control switch in the flight deck is activated and sends a discrete signal directly to the Hydraulic Pump Control Unit. Another discrete signal is sent directly back to an indicator light in the flight deck. Managing the interfaces (latency, timing, for example) is relatively straightforward.

In the IMA/DIMA architecture (labeled "IMA (A664) Architecture" in the right-hand portion of Figure 3), the same control switch can exist in the flight deck. However, instead of sending a discrete signal directly to the Hydraulic Pump Control Unit, the activation signal requires several protocol conversions (as it passes through several switches and remote data concentrators). During each signal protocol conversion, the signals must be validated. In addition, end-to-end timing, latency, and jitter, for example, must be integrated across all the intermediate components.

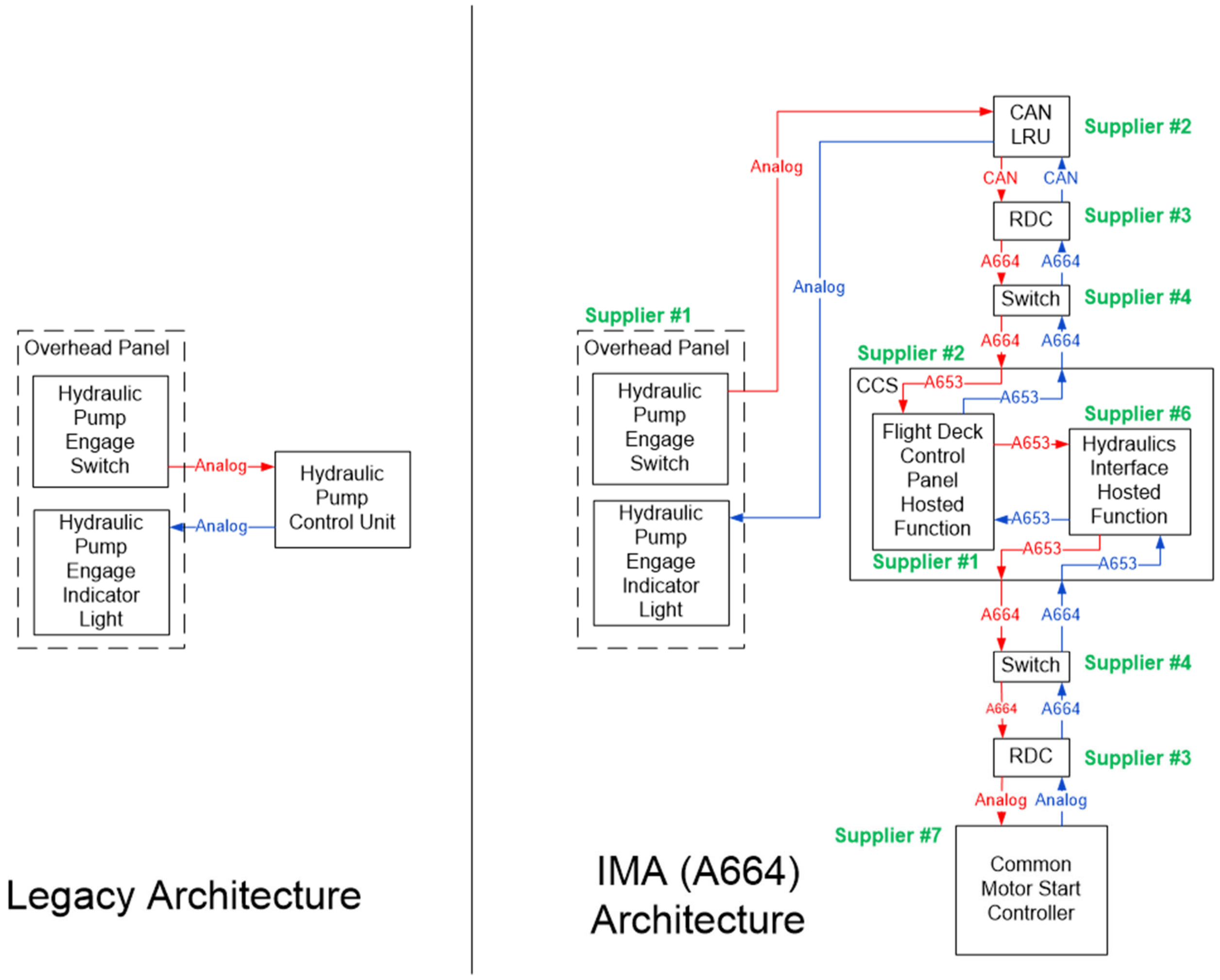


Figure 3. Same function, different networks

2.1 Discussion of NextGen integration at the system of systems level

Similar to the increased integration occurring in aircraft, the environment in which the aircraft operate is also changing. Figure 4 shows the "5A" aviation eco-system construct to describe this more integrated environment, in which the aircraft is a node in the broader aviation ecosystem. This "5A" environment consists of the airspace, airline, airplane, airport, and air traffic control elements.

An aircraft that employs modern IMA architecture can be considered a SoS. Similarly, the 5A aviation ecosystem is also very much a SoS (admittedly much larger than a single aircraft). That said, the same architecture complexity trends and integration challenges that occurred within IMA/DIMA aircraft would be facing the evolving 5A environment.

- New functionality.
- Reallocation of existing functionality (including autonomous systems).
- New interdependency.
- Increased air vehicle.

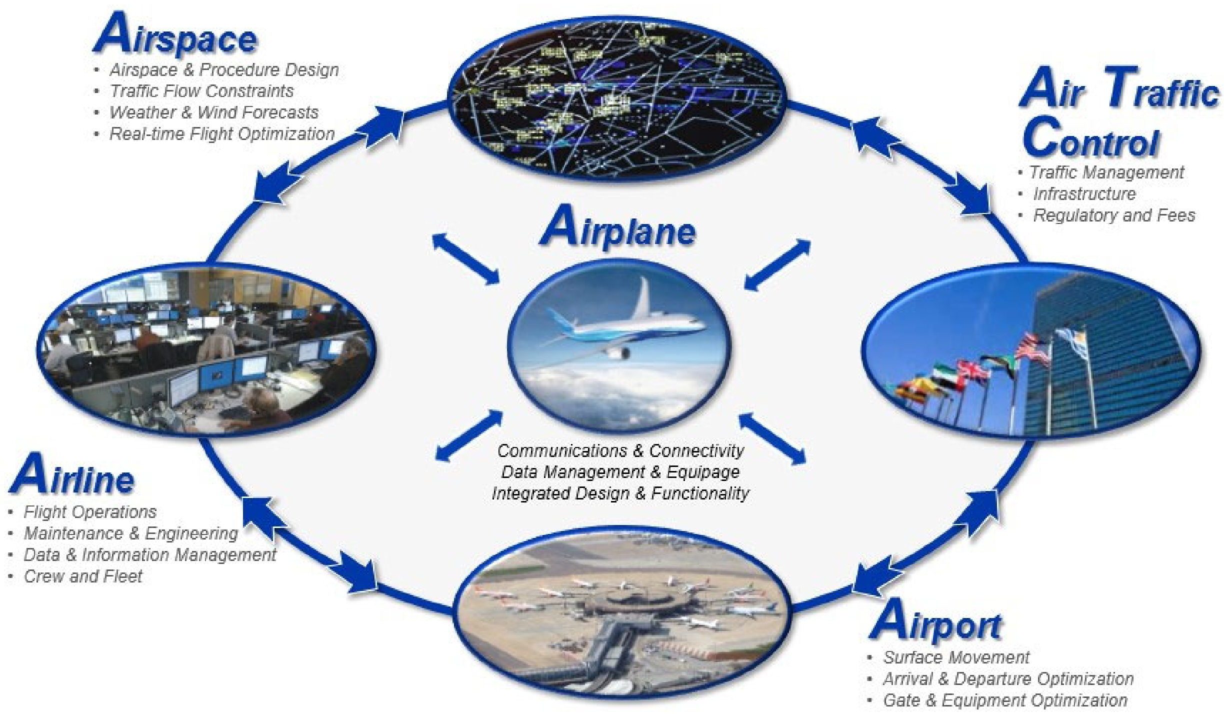

Airspace
• Airspace & Procedure Design
• Traffic Flow Constraints
• Weather & Wind Forecasts
• Real-time Flight Optimization
Air Traffic Control
• Traffic Management
• Infrastructure
• Regulatory and Fees
Airplane
Communications & Connectivity
Data Management & Equipage
Integrated Design & Functionality
Airline
• Flight Operations
• Maintenance & Engineering
• Data & Information Management
• Crew and Fleet
Airport
• Surface Movement
• Arrival & Departure Optimization
• Gate & Equipment Optimization

Figure 4. “5A” environment (source: Boeing)

While the focus of this research is on the aircraft, the potential exists that some of the recommendations could be extended to the larger 5A SoS aviation ecosystem.

3 Model-based activities

As discussed in the previous section, changes in systems architectures are driving increased interest in modeling activities to improve performance throughout the entire lifecycle.

In the context of Figure 5, Model-Based Engineering (MBE) can be considered as part of the fourth industrial revolution.

Note: The term "Digital Engineering" is typically used by the Government, particularly the Department of Defense. For the purposes of this paper, the terms "Digital Engineering" and "Model-Based Engineering" are interchangeable.

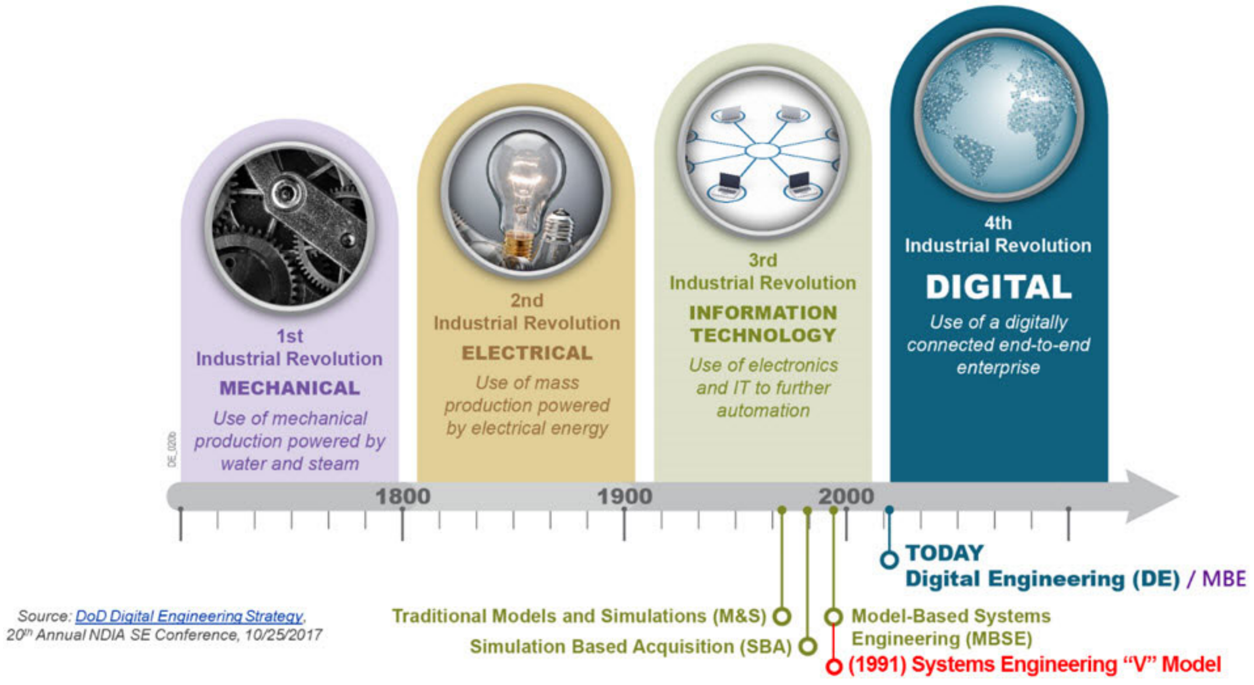

1st
Industrial Revolution
MECHANICAL
Use of mechanical production powered by water and steam
2nd
Industrial Revolution
ELECTRICAL
Use of mass production powered by electrical energy
3rd
Industrial Revolution
INFORMATION TECHNOLOGY
Use of electronics and IT to further automation
4th
Industrial Revolution
DIGITAL
Use of a digitally connected end-to-end enterprise
1800
1900
2000
TODAY
Digital Engineering (DE) / MBE
Traditional Models and Simulations (M&S)
Simulation Based Acquisition (SBA)
Model-Based Systems Engineering (MBSE)
(1991) Systems Engineering "V" Model
Source: DoD Digital Engineering Strategy, 20th Annual NDIA SE Conference, 10/25/2017

Figure 5. MBE/digital engineering (source: Boeing)

A quick summary is as follows:

1. The first industrial revolution introduced machines into the production process. This included the use of mechanical production powered by water and steam.
2. By the second Industrial Revolution, mass production was introduced, and electrification largely contributed to the success of this era.
3. The third Industrial Revolution resulted from the introduction of the computer and information technology.
4. In the current era, digitalization reshapes the way that we innovate and operate.

When one considers the anticipated changes in the entire aviation ecosystem (as shown in Figure 6), it becomes more apparent that the ability to digitally integrate and validate multiple systems interactions and behaviors will become increasingly important. Modeling has the potential to improve the efficiency of doing this integration in the broader aviation ecosystem.

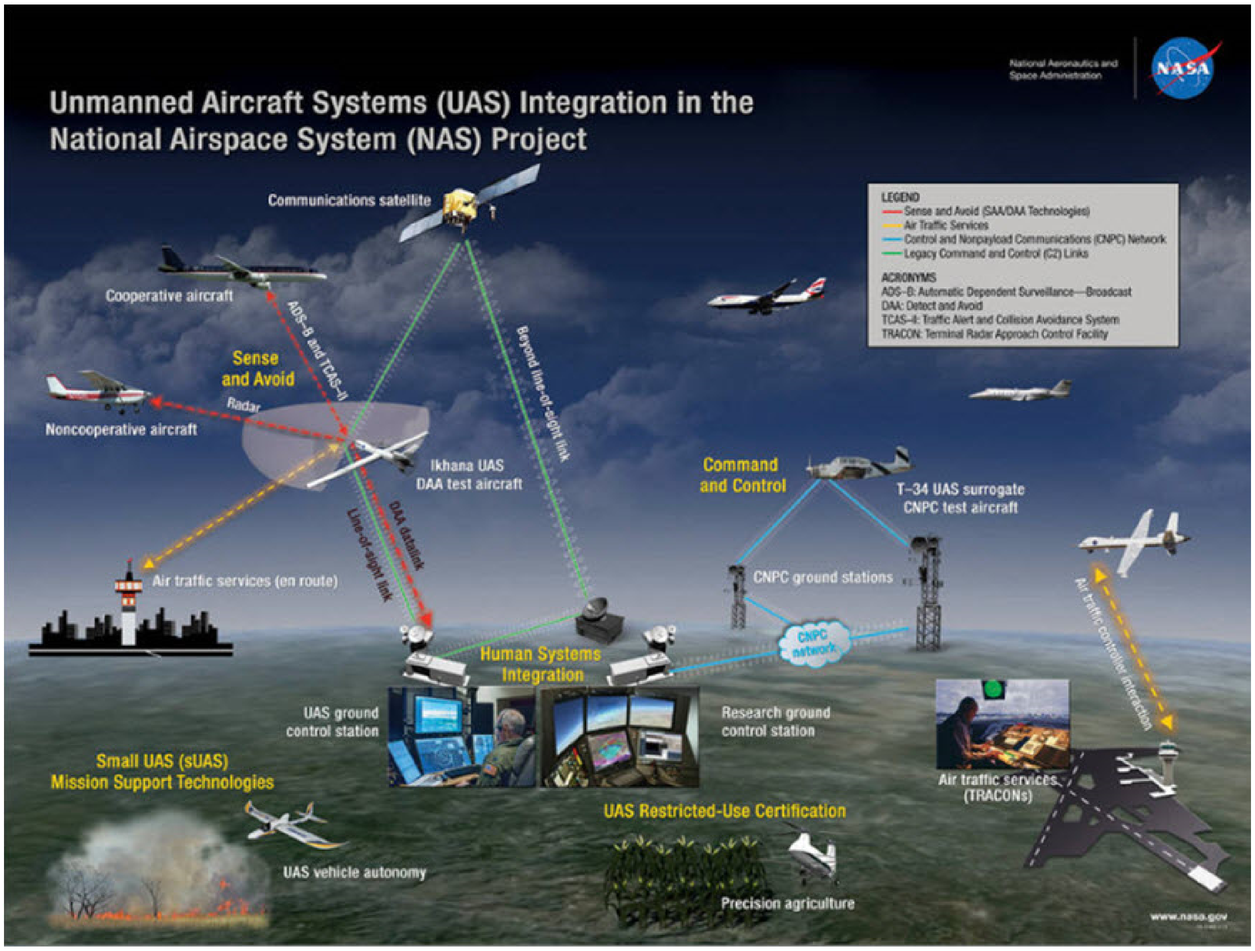

Unmanned Aircraft Systems (UAS) Integration in the National Airspace System (NAS) Project
National Aeronautics and Space Administration
NASA
LEGEND
🟥 Sense and Avoid (SAA/DAA Technologies)
🟨 Air Traffic Services
🟦 Control and Nonpayload Communications (CNPC) Network
🟩 Legacy Command and Control (C2) Links
ACRONYMS
ADS-B: Automatic Dependent Surveillance—Broadcast
DAA: Detect and Avoid
TCAS-II: Traffic Alert and Collision Avoidance System
TRACON: Terminal Radar Approach Control Facility
Communications satellite
Cooperative aircraft
ADS-B and TCAS-II
Beyond line-of-sight link
Sense and Avoid
Radar
Noncooperative aircraft
Ikhana UAS DAA test aircraft
Command and Control
T-34 UAS surrogate CNPC test aircraft
DAA datalink
Line-of-sight link
Air traffic services (en route)
CNPC ground stations
CNPC network
Air traffic controller interaction
Human Systems Integration
UAS ground control station
Research ground control station
Air traffic services (TRACONs)
Small UAS (sUAS) Mission Support Technologies
UAS Restricted-Use Certification
UAS vehicle autonomy
Precision agriculture
www.nasa.gov

Figure 6. National airspace system evolution (source: National Aeronautics and Space Administration [NASA])

To ensure there is a common framework and taxonomy, this report will use industry standard definitions for different modeling activities (as defined in Table 2). Any deviations from industry standard definitions will be annotated.

Table 2. Industry modeling activities and taxonomy

Modeling Activity	**Taxonomy**
Model-Based Enterprise	An **organization** that applies modeling and simulation technologies to integrate and manage its technical and business processes (National Institute of Standards and Technology or NIST).
Model-Based Engineering	An **approach** to product development, manufacturing, and lifecycle support that uses a digital model and simulation to drive first time quality and reliability (NIST).

Modeling Activity	**Taxonomy**
Model-Based Systems Engineering (SE)	**Execution of discipline** (SE) using digital model principles for system-level modeling and simulation of physical and operational behavior throughout the system lifecycle (International Council on Systems Engineering or INCOSE).
Model-Based Definition	A **part's definition** using a three-dimensional (3D) model. May define feature and part characteristics within the 3D model. Industry standard (The American Society of Mechanical Engineers [ASME] Y14.41).
Model-Based Instructions	**Graphical display of information** necessary for build/assembly, including MBD engineering intent (e.g., process specs, geometric definition and tolerances) (NIST).

3.1 Model-based engineering

The NIST definition of MBE is as follows:

An approach to product development, manufacturing and lifecycle support that uses a digital model and simulation to drive first time quality and reliability.—NIST

(The Department of Defense [DoD] has referred to this approach as "Digital Engineering.")

It is important to emphasize that MBE is an approach; it is not a specific process or tool.

In an MBE approach, the models are the cohesive element across the system's lifecycle. Data connectivity is critical in an MBE approach, which digitally connects value streams and workflows across the product lifecycle. A significant amount of data can be required to help validate and verify that an aircraft is going to perform as expected with no anomalous behavior. Section 4.1 provides examples of the data flows between the safety processes, the development assurance processes, and the software and hardware design assurance processes. The goal of MBE is to help these types of data connectivity more efficiently.

Data connectivity can help eliminate non-value added manual data translation – which is effort that adds cost and increases chances of introducing quality defects. The MBE approach can help with product development by ensuring that the information being used to make decisions is shared, complete, and accurate. This helps ensure that everyone is (digitally) "on the same page." Doing so will enable rapid decisions for design and airworthiness. It should also reduce scrap and rework by reducing translation and reentry errors, thus achieving first time quality.

Aspects of MBE have been implemented in various forms for decades. However, this was mostly accomplished by a document-centric, "siloed" perspective (e.g., mostly from an aircraft design perspective, from a production system perspective). MBE, with improved tools integration and processes, should support the full realization of the “digital thread” throughout the product lifecycle. Establishing a digital thread allows the program to connect information and have an authoritative source of truth, from beginning to end, and at the level of detail deemed appropriate by the program – from requirements, through detailed design, integration, V&V, customer delivery, and support services thereafter. A digital thread connects traditionally siloed elements in design, manufacturing, and service processes, and provides an integrated view of the product lifecycle from conceptualization to retirement.

MBE centralizes all information about the system in a model, often called the “authoritative source of truth.” Stakeholders, such as development teams and suppliers, can access the model at different views and levels of detail. The key is that different stakeholders, while having different interests in certain data types and levels of detail, are accessing the same data from the authoritative source of truth. This promises to help facilitate making data-driven decisions in a more efficient manner from a common source of product-definition truth. In order to accomplish this, a sophisticated common modeling environment, including tools, good practices, and adherence to industry standards, is required.

Note: Industry standards do not yet fully exist to support MBSE/MBSA completely. This will be discussed in subsequent sections.

3.2 Model-based systems engineering

The International Council on Systems Engineering (INCOSE) defines MBSE as the execution of SE discipline, using digital model principles for system-level modeling and simulation of physical and operational behavior throughout the system lifecycle. MBSE is a subset of MBE and is the SE aspect of MBE. The distinctions between these two are shown in Figure 7. For the intent of this report, the focus of MBSE will be on the aircraft.

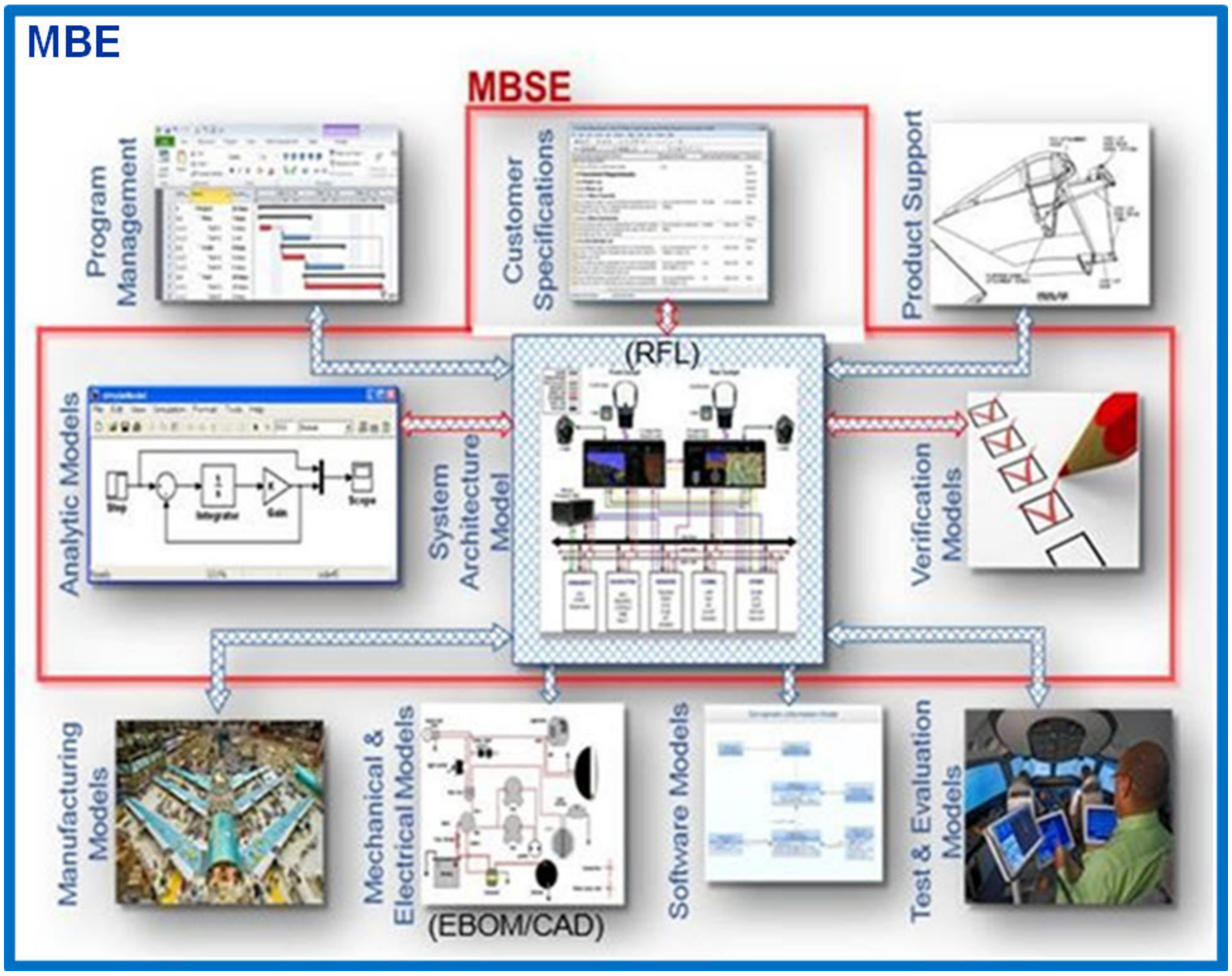


Figure 7. Scope of MBE and MBSE

MBSE is used to integrate SoS, system, segment, subsystem, component, and software requirements with the functional, logical, and physical architecture, plus behavior of the system (RFLP+B):

- **Requirements:** The traditional centerpiece of the systems development process, where the needs of the platform are expressed in terms of customer business and operational needs, regulatory compliance, and other system of system needs. These top-level requirements are decomposed to specify platform capabilities, functions (and associated performance), logical elements, and physical installations in a hierarchical construct. Verification of the design implementations is performed against requirements and traced from the bottom to the top of the hierarchy.

- **Functional Architecture:** What the platform and its systems do – a set of functions and their sub-functions that define the transformations of input flows into output flows performed by the system to achieve its mission.
- **Logical Architecture:** The organization and relationships of logical elements of a system/platform. System block diagrams and interfaces are traditional representations of the logical architecture.
- **Physical Architecture:** An implementation of the logical architecture into a buildable form for the production process that helps define the specific configuration – the natural transition where the logical elements take physical form (e.g., LRUs, software, wire bundles, motors) on the platform and define what the factory builds.
- **Behavioral Architecture:** An architecture defining the behavior of functions statically in diagrams (e.g., logic) or dynamically in models for simulation.

As depicted in Figure 8, a key aspect of MBSE is the integration of the different architectures: integrated requirements, functions and logical system elements that drive, and are ultimately consumed by, the physical design of the system.

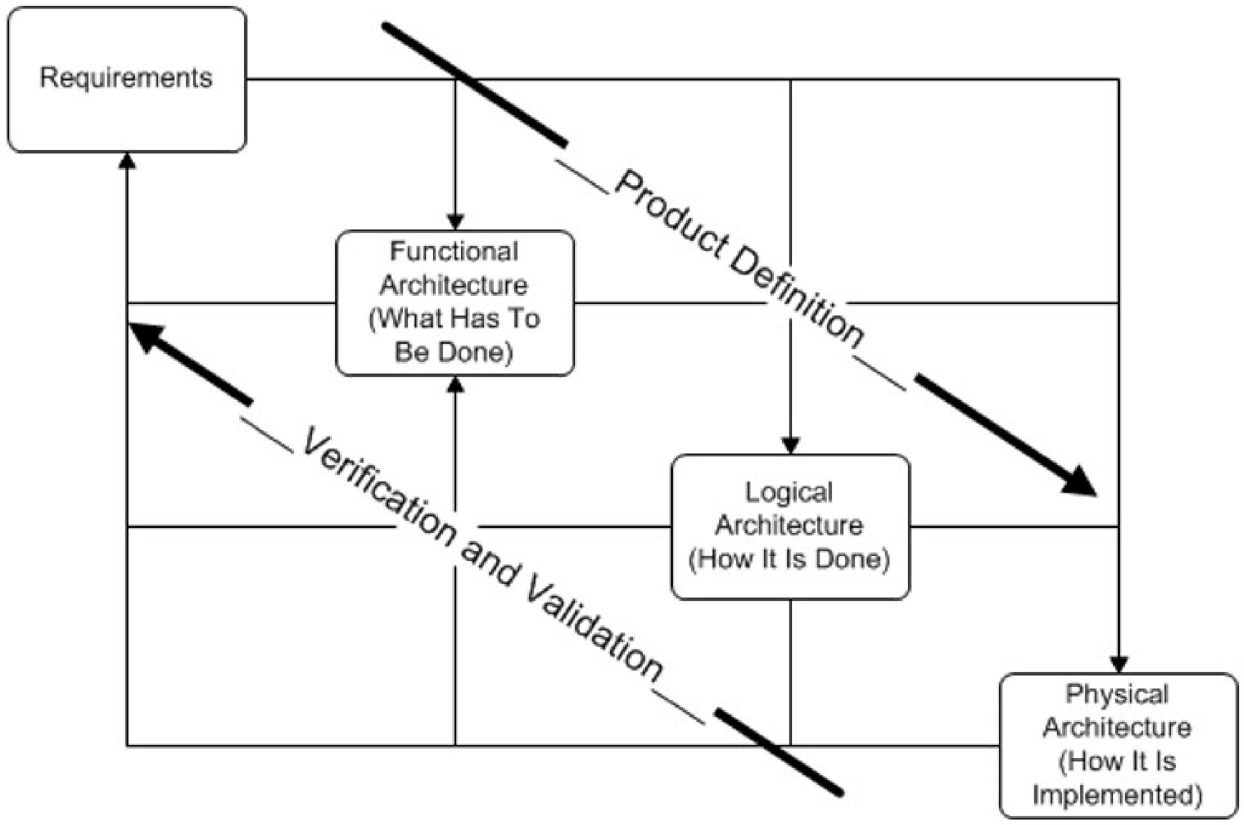

Requirements
Product Definition
Functional Architecture (What Has To Be Done)
Verification and Validation
Logical Architecture (How It Is Done)
Physical Architecture (How It Is Implemented)

Figure 8. Integrated systems engineering architectures

The purpose of the integrated architectures in MBSE is to ensure that the product will perform its intended functions with no anomalous behavior, as defined by the requirements. In addition, as systems become more integrated, it is important to ensure that there is no unacceptable emergent behavior. Some of the key processes to achieve the integrated SE architecture include:

- Requirements allocation (to functional, logical, and physical architectures).
- Functional allocation (to logical architecture).
- Logical allocation (to physical architecture).

3.2.1 Requirements allocation

As shown in Figure 9, requirements allocation, which is an important part of MBSE, is conducted to help ensure a complete and correct set of requirements. The purpose of requirements allocation is to:

- Provide a framework for identifying and validating functional requirements.
- Identify missing requirements.
- Develop derived requirements.
- Identify unrealistic and conflicting requirements.
- Eliminate poorly written requirements.
- Improve product integration by combining functions within the system and eliminating duplicate functions.

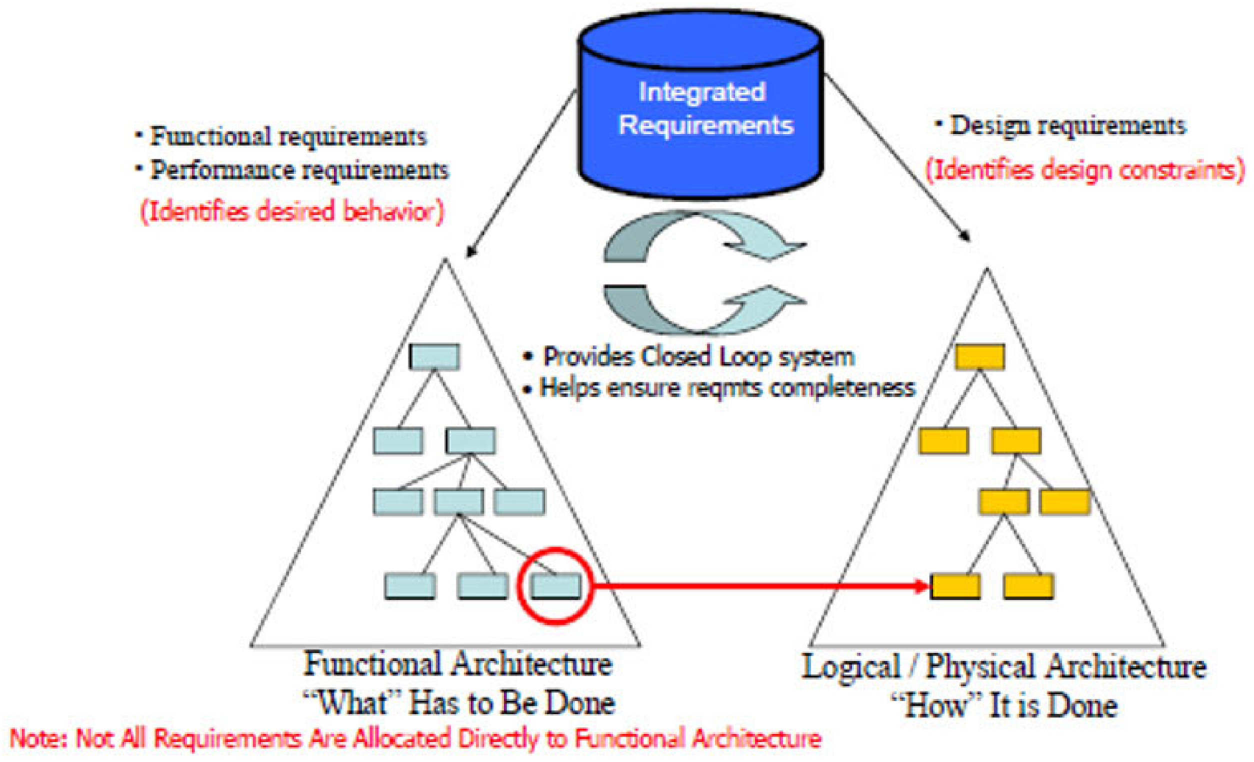

Integrated Requirements
• Functional requirements
• Performance requirements
(Identifies desired behavior)
• Design requirements
(Identifies design constraints)
• Provides Closed Loop system
• Helps ensure reqmts completeness
Functional Architecture
"What" Has to Be Done
Logical / Physical Architecture
"How" It is Done
Note: Not All Requirements Are Allocated Directly to Functional Architecture

Figure 9. Requirements completeness and correctness

Requirements allocation is complete when:

- All design and physical/installation requirements are allocated to the systems, subsystems, or equipment (or discrepancies resolved).
- All functional and performance requirements are allocated to functions or functional interfaces (or discrepancies resolved).

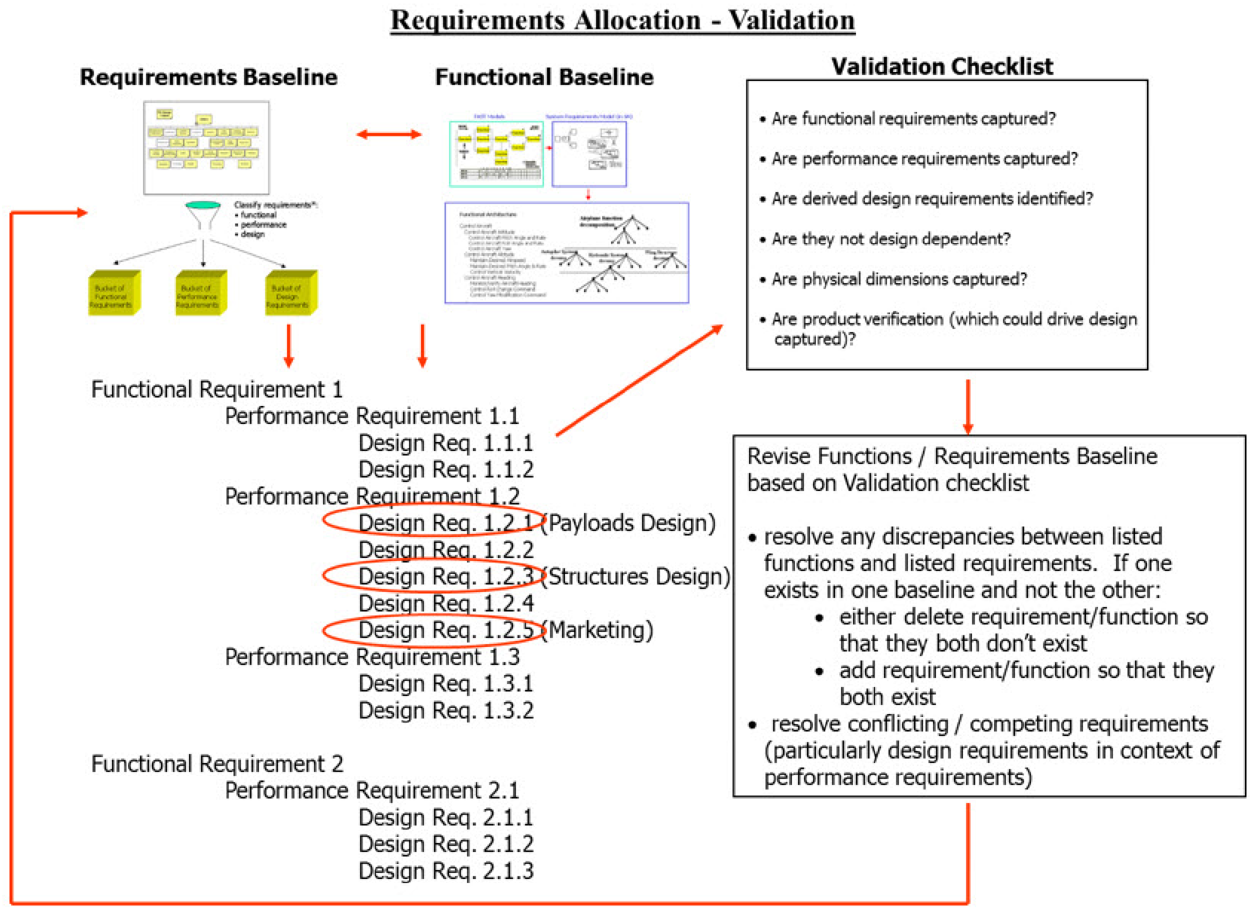
Figure 10 provides an example of some of the considerations that should be made during the requirements allocation to help ensure a complete and correct set of requirements.

Note: This example is mostly related to the requirements allocation to the functional architecture. Similar requirements validation activities should be conducted for the requirements allocation to the logical and physical architectures.

One of the benefits of MBSE is to allow the different stakeholders to have an integrated view of the requirements driving the design, for example:

- What does it have to do?
- How well does the system need to do it (i.e., performance)?

- What constraints will limit the design trade space (e.g., material, environmental conditions)?
- What are the cross-functional interfaces?



Validation Checklist

- Are functional requirements captured?
- Are performance requirements captured?
- Are derived design requirements identified?
- Are they not design dependent?
- Are physical dimensions captured?
- Are product verification (which could drive design captured)?

Revise Functions / Requirements Baseline based on Validation checklist

- resolve any discrepancies between listed functions and listed requirements. If one exists in one baseline and not the other:
  - either delete requirement/function so that they both don't exist
  - add requirement/function so that they both exist
- resolve conflicting / competing requirements (particularly design requirements in context of performance requirements)

Figure 10. Requirements allocation – validation

Requirements allocation can help identify the following conditions:

- Unallocated requirements.
- Functions without allocated requirements.
- Logical/physical definitions without allocated requirements.

Metrics, in and of themselves, will not guarantee that all missing requirements will be identified. For example, if a logical definition has nine requirements allocated to it, and a tenth requirement is missing, the metrics will not identify this. Metrics will provide first order validation (e.g., functions identified that no systems are implementing, logical definitions with no functions or requirements). By conducting the allocation, the designer will have the context to better understand tradeoffs and cross-functional impacts (i.e., improve the chances that missing and conflicting requirements will be identified and resolved).

3.2.2 Functional decomposition and allocation

The purpose of this SE process is to transform the functional, performance, interface, and other requirements that were identified through requirements analysis into a coherent description of system functions that could be used to guide the logical design activity. Decomposing higher-level functions into lower-level functions, identifying functional interfaces, identifying timing relationships between functions, allocating performance requirements to functions and design requirements to logical designs, and allocating functions to the logical designs are integral parts of functional allocation. Typical deliverables include the following: functional architecture, data flow diagrams (DFDs), functional flow block diagrams (FFBDs), and allocation matrices.

Functions are discrete actions (typically using action verbs followed by nouns) necessary to achieve the system's objectives.

The functions identify "what" the system had to do (e.g., calculate air temperature). These functions may be stated explicitly, or they may be derived from stated requirements. The functions will ultimately be performed or accomplished through use of equipment, personnel, software, or a combination.

Functions are hierarchical in nature and have two key relationships: parent-child functions and sibling functions. For example, airplane-level functions are decomposed into systems-level functions. Systems-level functions are decomposed into sub-functions with the intent of defining alternative sub-function arrangements and sequences and functional interfaces. Validation checks should be conducted to ensure that the task completion of the children function would result in the task completion of the parent function.

The result of the functional decomposition process is a functional architecture, consisting of, for example, the airplane, systems, and component level functions.

In its most basic form, the functional architecture is a simple hierarchical decomposition of the functions with associated performance requirements. As the architecture definition is refined and made more specific, the functional architecture becomes more detailed and comprehensive.

Functions are identified as part of the functional decomposition process. These functions are then allocated to the systems/subsystems that would implement these functions. Table 3 represents a notional view of a function to systems allocation matrix.

Table 3. Systems required to implement airplane-level function

Decomposed functions	**Implementing system**
Function	System
Function	System A
Function	System A
Function	System B
Function	System C
Function	System D
Function	System D
Function	System E

The functional allocation helps assess the coupling, connectivity, and cohesion of the design.

3.2.3 Coupling

Coupling is a measure of dependence between system elements within a system architecture. The desire is to have low coupling; systems function as much as possible as standalone entities, and neither require many inputs from other systems, nor produce many outputs for other systems. To ensure low coupling, sibling functions with many data flows between their descendants are allocated, as much as possible, to the same system. Sibling functions with few data flows between their descendants may be allocated to different systems. Figure 11 and Figure 12 show the difference between high coupling and low coupling.

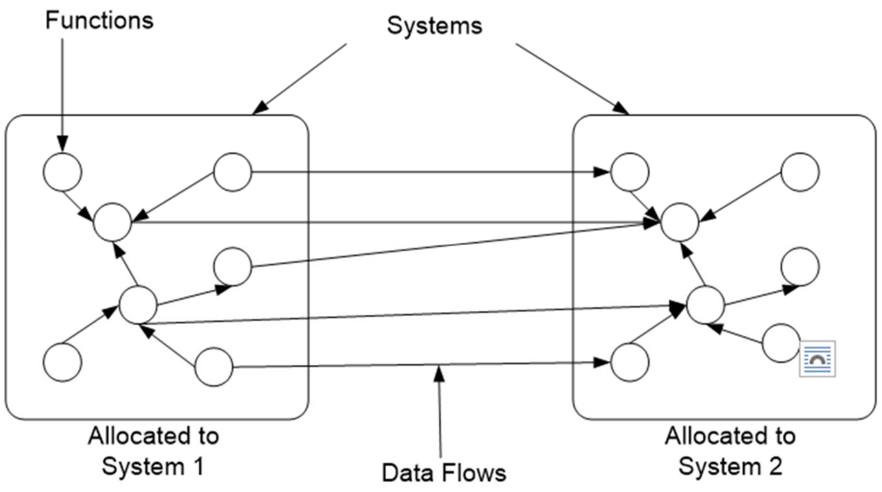


Figure 11. High coupling

Low coupling is desired.

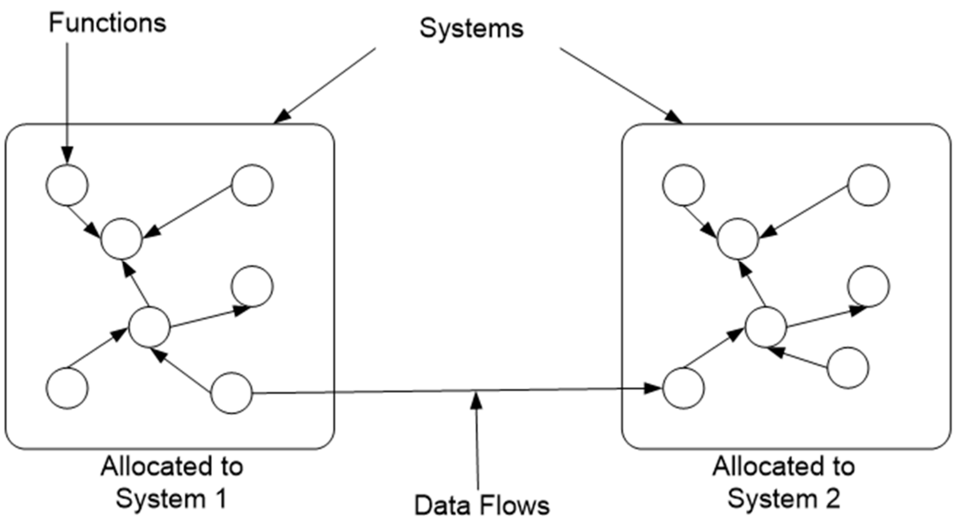


Figure 12. Low coupling

3.2.4 Cohesion

Cohesion is the similarity of tasks performed within modules (e.g. functional, sequential, communication, procedural, temporal, and logical).

A cohesive module (system) performs a single task, requiring little interaction with other modules/procedures. A function is cohesive to the degree that the function and all its descendants are allocated to a single system. High cohesion is achieved by allocating functions to systems as high up the function tree as possible. High cohesion contributes to modular design and fault isolation. Figure 13 and Figure 14 show low cohesion and high cohesion.

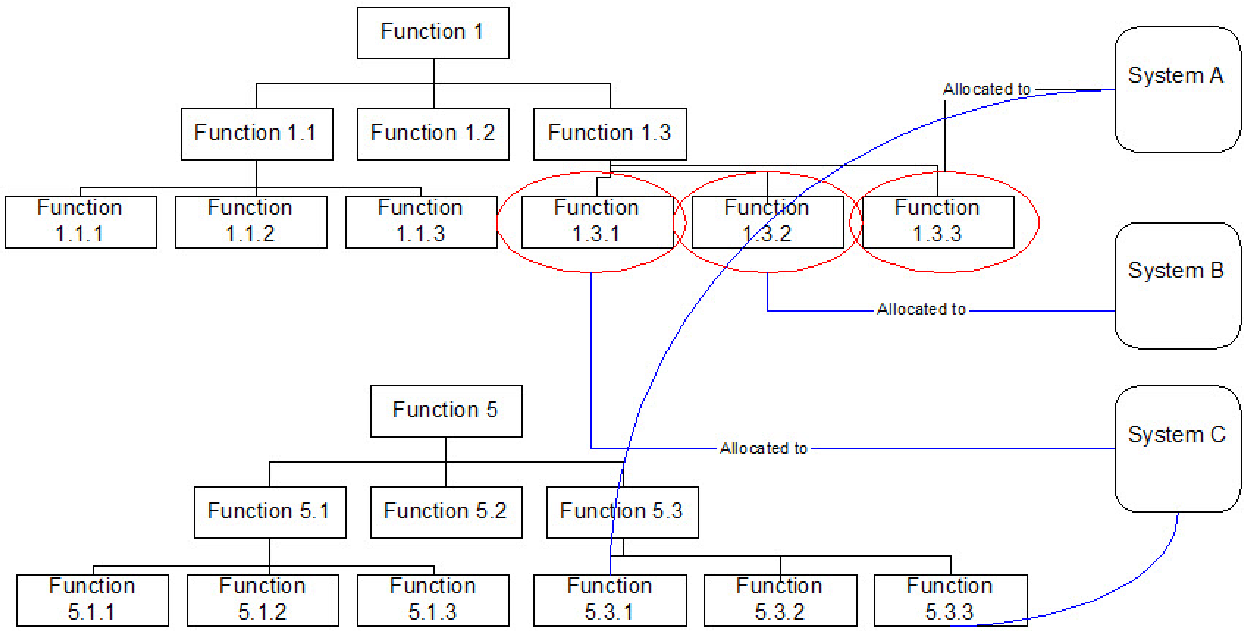


Figure 13. Low cohesion

High functional cohesion is desired.

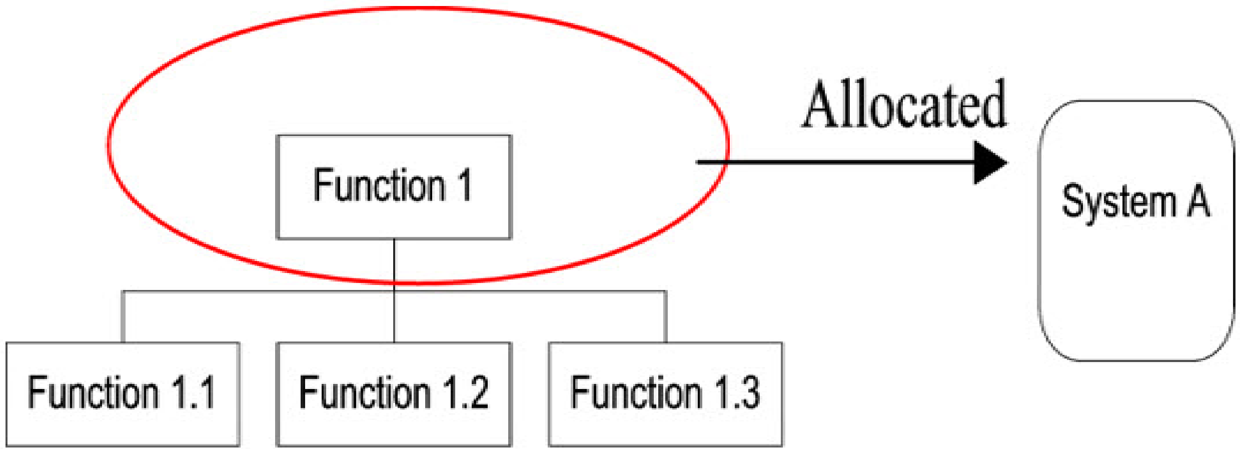


Figure 14. High cohesion

3.2.5 Connectivity

Connectivity is a measure of the degree to which data flows occur between non-sibling functions. High and low connectivity are depicted in Figure 15 and Figure 16.

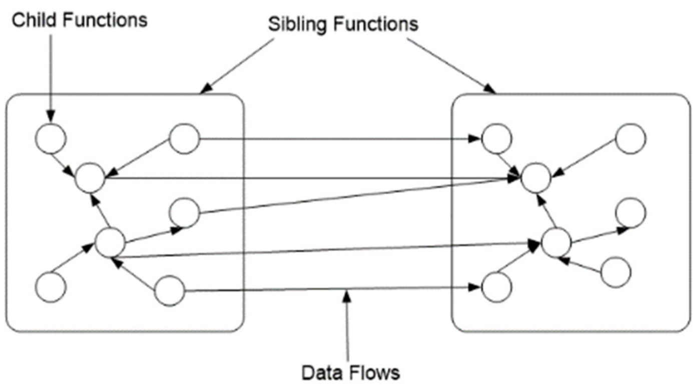


Figure 15. High connectivity

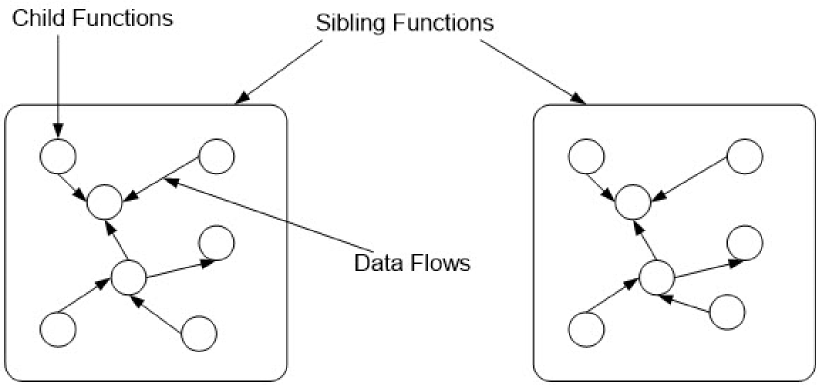


Figure 16. Low connectivity

3.2.6 Iterative architecture

Functional decomposition and allocation are repeated to define successively lower level functional and performance requirements, thus defining architectures at ever-increasing levels of detail. The functional allocation and analysis process was, per the design process, conducted in iterative – often simultaneously performed – steps.

These steps are highly interrelated, and require multiple iterations at each level. As the lower-level work products are finalized, detailed review of the higher-level tiers is necessary to correct and realign the levels.

The functional allocation and analysis activities result in the creation of the functional architecture and the allocation between the functions and the logical architecture.

3.2.7 Logical allocation

Logical allocation to the physical architecture is repeated to allocate successively lower levels of the logical architecture at increasing levels of detail. The logical architecture is allocated in sufficient detail to support the integrated system design. This includes enabling physical integration. One of the key goals of logical allocation is to validate that the logical elements have been “consumed” by the physical architecture.

3.3 Model-based safety analysis

Traditional airplane safety analyses (e.g., Preliminary Airplane Safety Assessment [PASA], Preliminary System Safety Assessment, Fault Tree Assessments, Failure Modes, and Effects [Criticality] Analysis) are often performed manually (and completely and correctly). However, they can be labor intensive, especially for modern, highly integrated airplanes. MBSA is a potential way to improve efficiency, while achieving the same desired results.

MBSA is an opportunity to more closely couple model-based design activities and the safety analyses. A key goal is to ensure that a common model is used for both SE and safety engineering. Working from integrated, common models should reduce the likelihood of late discoveries during the safety analyses, which will help reduce non-recurring cost during the design process.

Industry consensus guidance on MBSA will be forthcoming. An MBSA appendix will be contained in the upcoming release of ARP 4761A. The MBSA appendix describes the concepts and processes associated with performing a safety analysis using failure propagation models.

Performing the safety assessment of a design consists of understanding the system content and behavior to provide safety analysis results that are compared to objectives/requirements.

In order to understand system behavior, the safety analysis process interprets design documents and involves clarifying discussions with the engineers who independently develop the design. In the classic safety assessment approach, the safety analyst uses the acquired understanding to construct fault trees or other safety analysis artifacts manually. The MBSA methodology achieves the same results as the classical method, but adds benefits of integrating safety analyses with a shared model truth-source.

As shown in Figure 17, the overall goals of MBSA are to:

- Support safety analysis of its function and/or architecture by creating representative failure models of the system.
- Help address the complexity of functions and systems.
- Establish a common communication mechanism between designer and safety analyst.

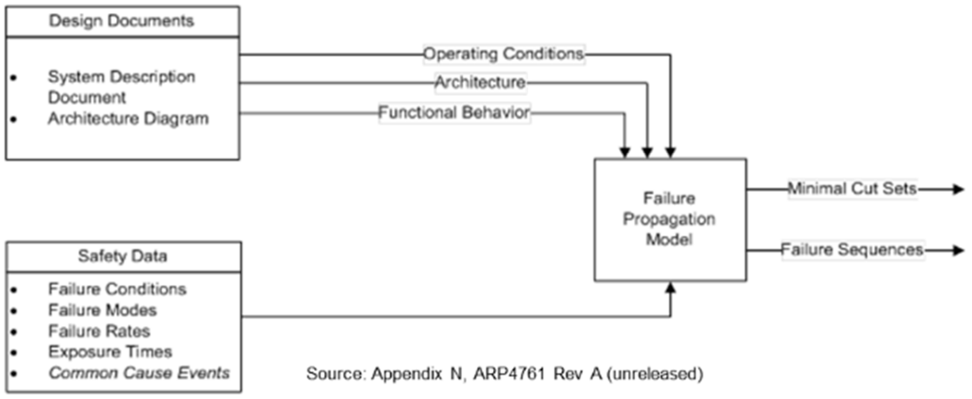


Figure 17. Model-based safety analysis approach

MBSA is a generic term for a family of techniques and methods, based on a Failure Propagation Model (FPM). The Failure Propagation Model is comprised of all the components in the system architecture necessary to represent each subsystem and their fault behaviors to the required level of abstraction, in order to understand how component failures in the system architecture affect their supported functions. Each component in the model has attributes (e.g., resource dependency logic, failure rate), which are used by the various analyses engines.

The FPM describes the relationship between the inputs and outputs of the elements in a nominal situation the failure events with their occurrence conditions (based on input data and failure mode) and their effects on outputs, and the links between elements according to the system architecture.

Figure 18 identifies the MBSA process inputs and outputs. MBSA is considered a specialized subset of MBSE (creation of the FPMs). Ideally, MBSA will be using the same models developed as part of MBSE (e.g., functions, systems architecture).

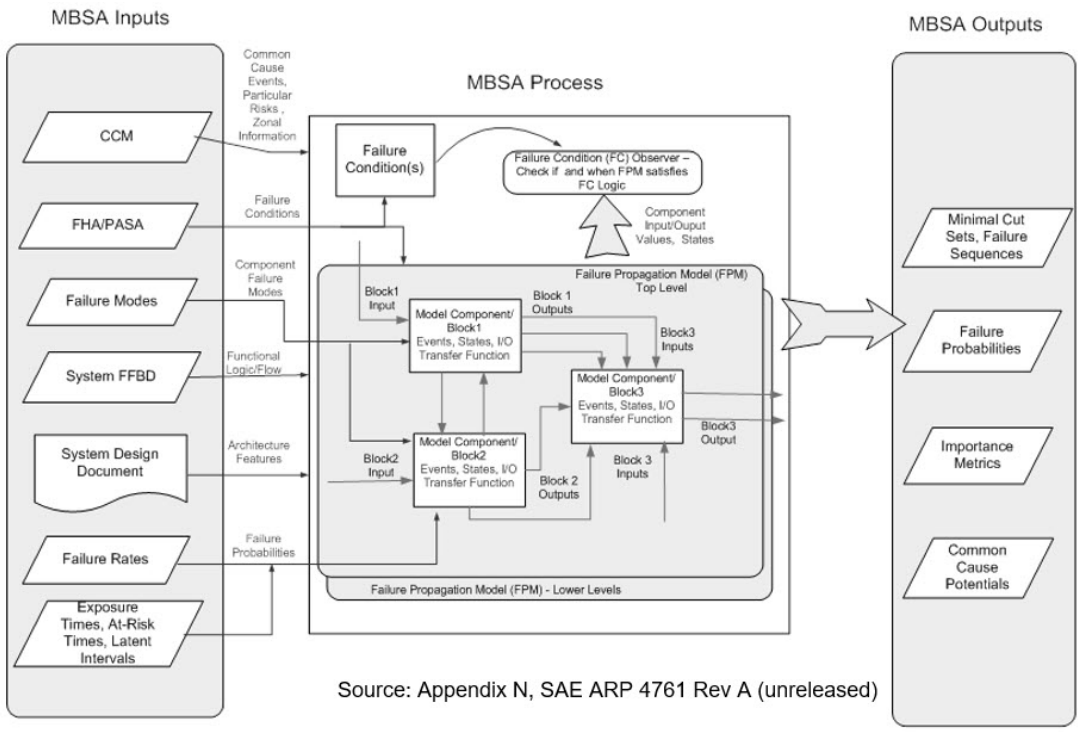

MBSA Inputs
MBSA Process
MBSA Outputs
CCM
FHA/PASA
Failure Modes
System FFBD
System Design Document
Failure Rates
Exposure Times, At-Risk Times, Latent Intervals
Common Cause Events, Particular Risks , Zonal Information
Failure Conditions
Component Failure Modes
Functional Logic/Flow
Architecture Features
Failure Probabilities
Failure Condition(s)
Failure Condition (FC) Observer – Check if and when FPM satisfies FC Logic
Component Input/Ouput Values, States
Failure Propagation Model (FPM) Top Level
Block1 Input
Model Component/ Block1 Events, States, I/O Transfer Function
Block 1 Outputs
Block3 Inputs
Model Component/ Block3 Events, States, I/O Transfer Function
Block3 Output
Block2 Input
Model Component/ Block2 Events, States, I/O Transfer Function
Block 2 Outputs
Block 3 Inputs
Failure Propagation Model (FPM) - Lower Levels
Minimal Cut Sets, Failure Sequences
Failure Probabilities
Importance Metrics
Common Cause Potentials
Source: Appendix N, SAE ARP 4761 Rev A (unreleased)

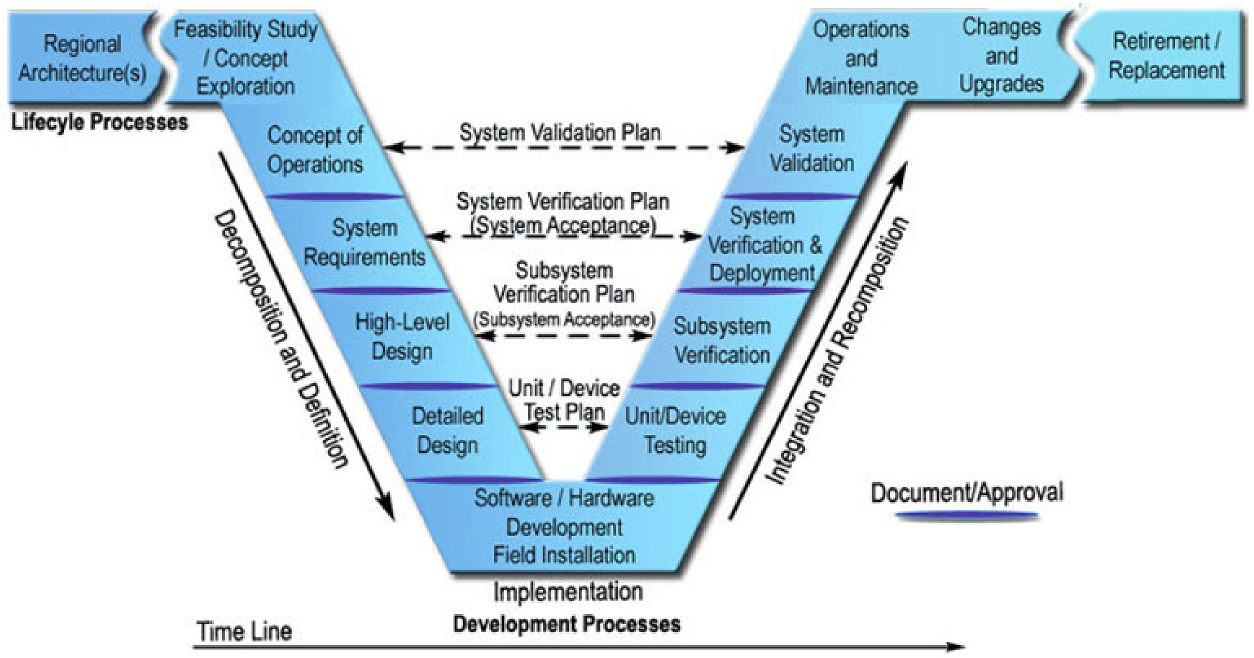
Figure 18. MBSA process (inputs and outputs)

3.4 Systems engineering "V" model in a model-based environment

The traditional SE "V" portrays the decomposition of requirements and design definition from the system down to components (left side of the "V") and the resulting systems integration and validation from components up to the end-user system (right side of the "V"). It has been used by government, industry, and academia for over 25 years as a model of the product development process for complex engineered systems. The V-model originated at a time when engineering processes had become narrowly stove-piped, and handoffs and interfaces between disciplines were document-centric and were manual exchanges. The traditional SE "V" was developed in the 1990s, some 30-40 years after the recognition of the need for SE to develop complex ballistic missile systems.

The icon portrays the engineering lifecycle as a linear series from left to right and the decomposition of requirements and system elements from the top to bottom of the left side of the V. The right side of the V represents the realization of the system concept as it passes through integration, test, and evaluation and finally fielding of the system.

As shown in Figure 19, the "V" symbol depicts the end-to-end product development process across the entire life cycle. The "V" starts with customer needs and ends with delivered solutions to fill those needs. The "V" has historically focused on the development of the physical systems (both hardware and software) to meet those needs. In other words, the traditional SE "V" focuses on the design and delivery of the physical product to meet customer needs. A series of milestones or artifacts mark the linear progression (from left to right) through the SE process.



SOURCE: US Department of Transportation Federal Highway Administration
https://ops.fhwa.dot.gov/publications/seitsguide/section3.htm

Figure 19. Traditional systems engineering "V"

The iconic SE "V" (in its various representations) is an intuitive and instructive framework for depicting product development. However, there are some drawbacks to the traditional SE "V." Unfortunately, the "V" is sometimes interpreted only being applicable to development of a product, when in reality, it should also be applied simultaneously to the production system, as well as the services and support systems associated with the product.

- The layout of the "V" implies a sequential development process. Attempts to map the "V" symbol to a project schedule, timeline, or standard gated process, only serves to exacerbate this misconception.
- The origin of the "V" was during the "document-centric" age, when information was literally handed off between individuals, groups, and lifecycle phases. Unfortunately, the "V" symbol does not adequately reflect the digital era of an MBE ecosystem, including Digital Thread and Digital Twins.

- The layout of the "V" also fails to depict the integrated and iterative nature of product development clearly. Attempts to address this shortcoming by adding horizontal "feedback" links across the "V" only serve to further complicate the symbol and are often overlooked or misunderstood by the audience.
- Historical attempts to update the "V" symbol have only served to increase its complexity and made it that much more difficult to understand.

However, this linear representation of the "V" model is not as helpful in depicting the real-time interchange of data and information between the complex interactions of an MBE ecosystem.

MBSE and MBSA involve the transition from a document-focused mindset to a digital engineering mindset that leverages information flow across the lifecycle. One of the more unique challenges is the development of physical worlds and virtual representations of the products and services. This in itself leads to a virtual product development cycle that can be treated as a "mirror image" of the physical product development cycle.

MBE requires a multi-dimensional view into the activities involved in SE. Additionally, in MBE, data is more expressive than what is generally allowed by passing documents in the "V." Therefore, as more model-based approaches are implemented, the "V" loses some of its effectiveness for SE.

Boeing developed Figure 20 to acknowledge the importance of treating the digital twins of "as-designed," "as-built," "as-delivered," configurations as products in their own right. It is intended to:

- Represent MBE as a multi-dimensional, iterative process encompassing both physical and virtual implementations.
- Reflect the integrated nature of MBE, linked with feedback to related lifecycle elements.
- Show relationships spanning business domains (e.g., product, production, service and support).
- Communicate how the SE process is different by using MBE.
- Be easy to understand, yet remain flexible and tailorable to design iterations.

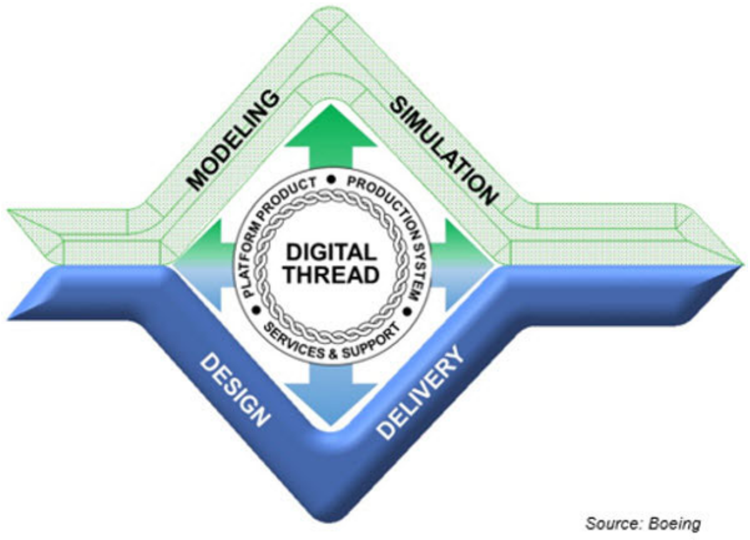


Figure 20. SE diamond

Note: The purpose of this is to highlight how MBE changes the traditional SE processes. It is not meant to advocate for a new icon.

Regardless of how this is represented graphically, it is important to recognize the development of virtual systems or digital twins that inform the development of the physical systems. This is one of the major changes, as we transition from SE to MBE.

Figure 20 updates the traditional "V" to add a virtual representation of the product, production system, and services and support systems. The "MBE Diamond" separates the physical and virtual paths into two Vs that face each other (top and bottom).

The bottom half of the MBE Diamond, as shown Figure 21, aligns with the traditional SE "V" and is focused on the instantiation of the physical systems (including the platform product, production system, services and support, test and evaluation [T&E]) across the lifecycle. The bottom half of the diamond represents the physical systems (retaining the traditional SE "V" flow). The "V" starts with customer needs and ends with delivered solutions to fill those needs. The "V" has historically focused on the development of the physical systems (both hardware and software) to meet those needs.

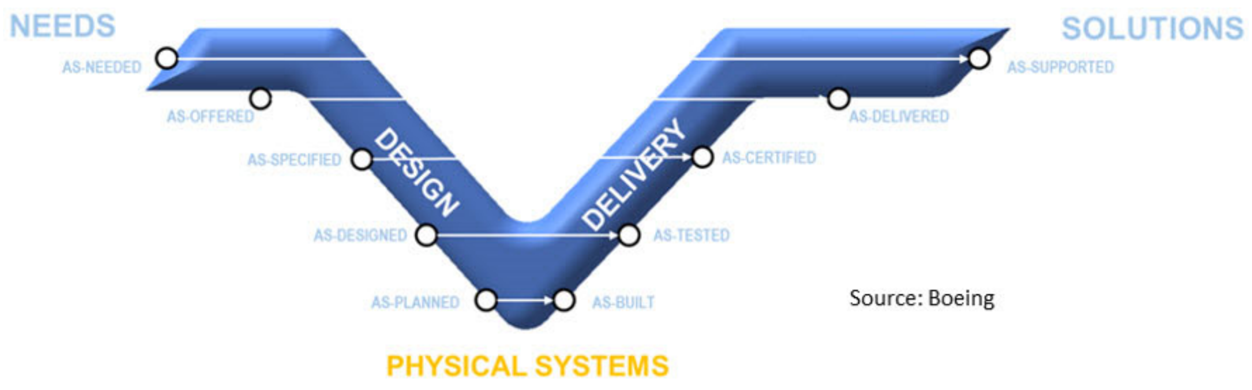


Figure 21. Bottom half of MBE diamond

The top half of the MBE diamond represents the "Digital Twins" (i.e., the virtual systems representation of the physical systems). Figure 22, with a virtual "V" flipped on top of the physical "V," recognizes the development of the virtual systems or digital twins that inform the physical systems development. It demonstrates the transition from systems engineering to MBSE, which uses modeling and simulation to create digital twins of the physical systems. Similar to the lower "V," symmetry exists between the artifacts on the left and right side of the upper "V." Models developed on the left side of the MBE diamond are used for increasingly higher fidelity simulations on the right side.

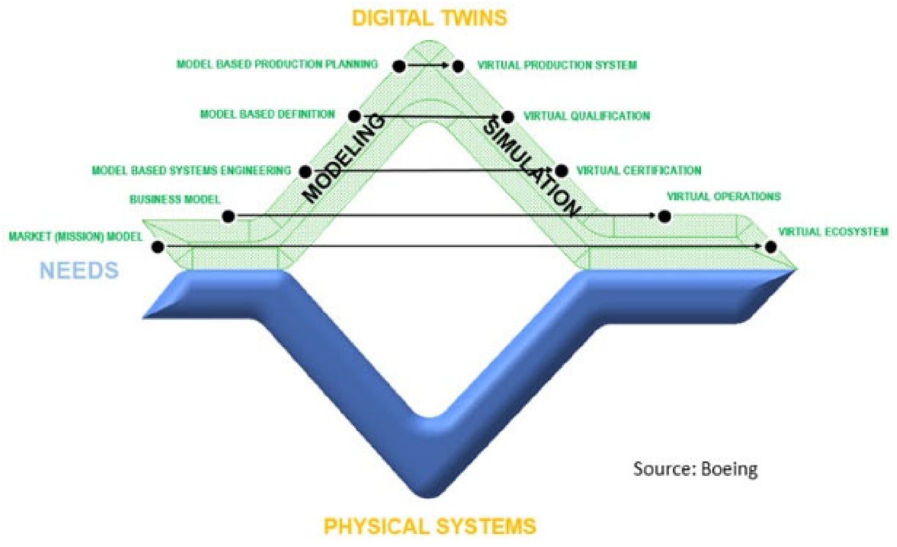


Figure 22. Top half of MBE diamond

The integrated physical and virtual development cycle is represented from left to right through the diamond, from needs through sustainment/operations and maintenance (O&M). Figure 23 shows the symmetry between the virtual and physical systems. This is intentional, as the digital twins are used to inform the development of the physical systems.

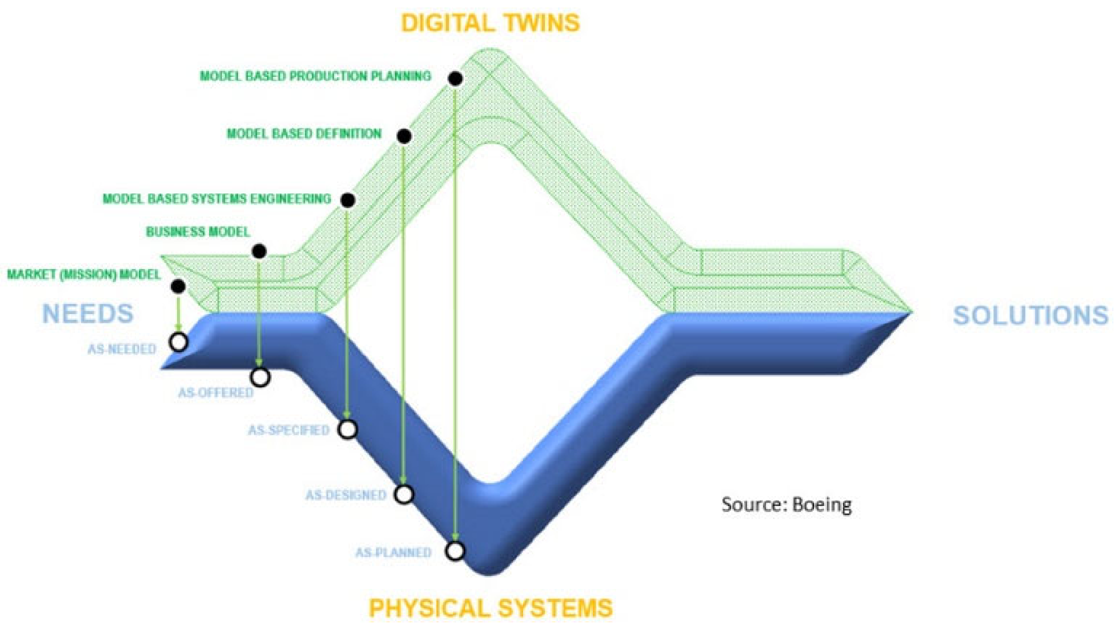


Figure 23. Digital twins and physical systems (left side)

Figure 24 shows the symmetry between the artifacts for virtual and physical systems. The physical systems can be used to inform the virtual systems; the virtual systems can be used to inform the physical systems. For example, production data analytics from the shop floor can be used to update production simulation models to conduct real-time “what-if” analyses related to production rate changes. As another example, data from high-fidelity virtual simulations of the product operating in its physical environment can potentially be used to reduce the need for physical testing.

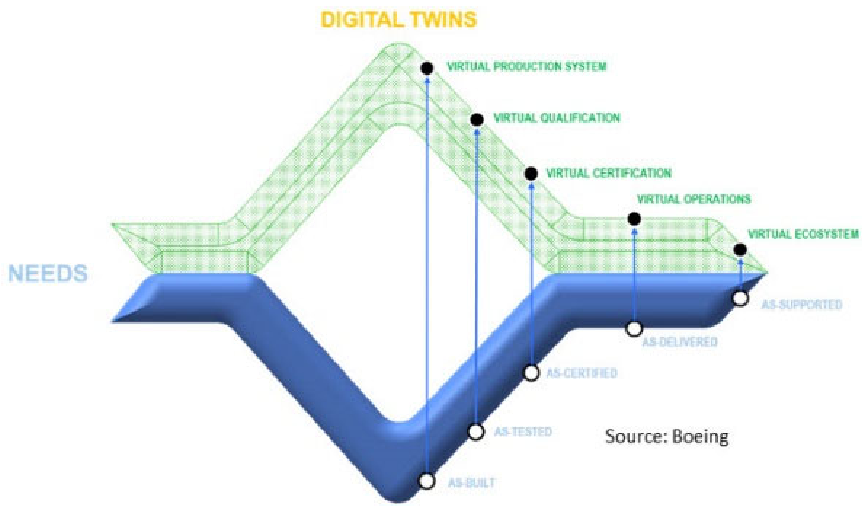


Figure 24. Virtual and physical systems (right side)

As shown in Figure 25, the central, circular arrows symbolize the integrated development of the platform product, production system, services and support, and T&E. The Digital Thread is represented in the interior of the diamond, linking models/simulation (digital twins) to the physical design of the systems (platform product, production system, services and support, and T&E). The Digital Thread provides continuous feedback into the process leading to better-informed designs that balance system performance, lifecycle cost, and operations and maintenance targets.

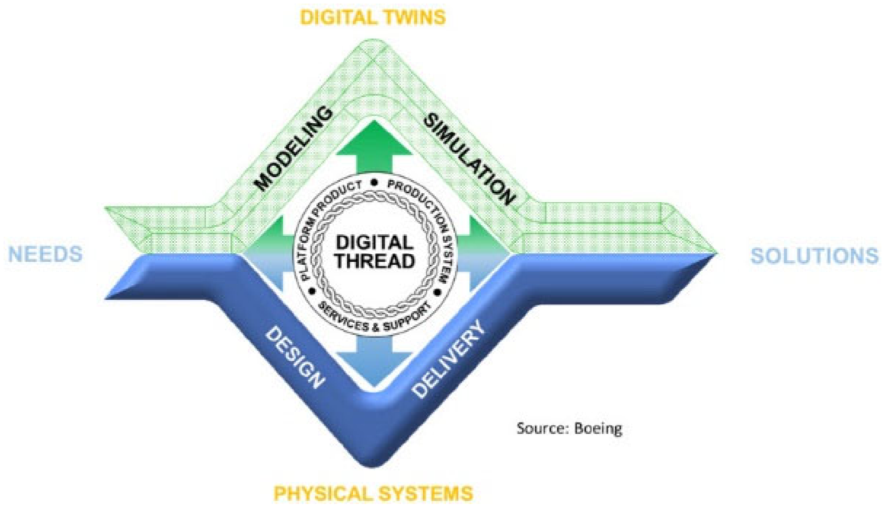


Figure 25. Digital thread

The Digital Thread is represented in the interior of the diamond, linking models/simulation (digital twins) to the physical design of the systems (Platform Product, Production System, services and support, and T&E). The Digital Thread provides continuous feedback into the process, leading to better-informed designs that balance systems. The MBE Diamond treats the development of the digital twin and physical twin as concurrent paths that both inform each other across the product lifecycle. The process is integrated via the digital thread across the platform product, production system, and services and support system.

4 Current guidance and standards for IMA and DIMA systems

The first part of this research is to assess the current guidance and standards for IMA, DIMA, and other distributed system architectures. Table 4 describes the approach to address these research objectives.

Table 4. Existing guidance for IMA and DIMA systems

Research Objectives	**Detailed Work Description**
Assess current guidance and standards for Integrated Modular Avionics (IMA) systems, Distributed Integrated Modular Avionics (DIMA) systems, and other distributed systems and architectures.	Analyze existing industry processes, survey industry committee members responsible for guidelines around development, validation and verification of highly integrated, complex digital systems: Identify existing industry guidelines for requirements definition, verification and validation (V&V) processes, systems integration and change impact analysis. Identify requirements for regression analysis and test. Identify potential shortcomings in current processes, particularly related to Section 4.6.4 (Aircraft/System Integration) and Section 6 (Modifications to Aircraft or Systems) of SAE Aerospace Recommended Practice (ARP) 4754A [1]. Identify integral systems integration process gaps related to safety that are not currently part of ARP 4754A and ARP 4761. Identify common errors which occur in the interrelationships between process steps in ARP 4754A, RTCA Document (DO)-178, DO-254 [4], and DO-297 [5], particularly those which could result in incomplete and incorrect requirements and/or systems integration issues for Integrated Modular Avionics architectures Identify potential shortcomings in DO-331 (Model-Based Development and Verification Supplement to DO-178C and DO-278A). Identify modeling standards.

4.1 Existing guidelines on system development and assurance processes

The following guidelines are used for requirements definition, V&V processes, systems integration, and change impact analysis for IMA and DIMA architectures. Figure 26 shows the standards often used for system, software, and airborne electronic hardware development; it illustrates the flow between safety assessment processes covered by ARP 4761 [2], development assurance processes covered by ARP 4754 [7], and design assurance processes covered by DO-178 [3] and DO-254 [4]. For the purpose of this document, DO-178 and DO-254 will be referred to as "design assurance activities."

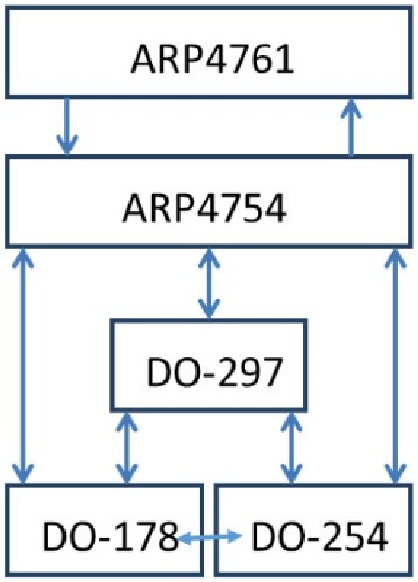


Figure 26. Standards used for system, software, and AEH development

Table 5 provides additional details on the purpose and primary level at which these standards are applied (e.g., airplane, system).

Table 5. Existing industry processes

Industry Guideline	**Purpose**	**Primary Applicable Level**
ARP 4761, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment [2]	Provides guidelines and methods for performing the safety assessment for civil aircraft, including (but not limited to) safety analyses, such as Functional Hazard Assessment (FHA), Preliminary System Safety Assessment (PSSA), and System Safety Assessment (SSA).	Airplane system/subsystem
ARP 4754A, Guidelines for Development of Civil Aircraft and Systems [1]	Provides guidelines on the development assurance process. This includes validation of requirements and verification of the design implementation for certification and product assurance. The development planning elements consist of: ▪ Development. ▪ Safety program. ▪ Requirements management. ▪ Validation.	Airplane system/subsystem

Industry Guideline	**Purpose**	**Primary Applicable Level**
	▪ Implementation verification. ▪ Configuration management. ▪ Process assurance. ▪ Certification. ▪ Software integration process. ▪ Software configuration management. ▪ Software quality assurance process. ▪ Certification liaison.	
DO-254, Design Assurance Guidance for Airborne Electronic Hardware [4]	Provides design assurance guidance for the development of airborne electronic hardware. Key processes include: ▪ Hardware safety assessment. ▪ Requirements capture process. ▪ Validation. ▪ Verification. ▪ Configuration management. ▪ Process assurance. ▪ Certification liaison.	Airborne Electronic Hardware (AEH)
DO-297, Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations [5]	Provides guidance for IMA modules, applications, and systems. The integral processes consist of: ▪ Safety assessment. ▪ System development assurance. ▪ Validation. ▪ Verification. ▪ Configuration management. ▪ Quality assurance. ▪ Certification Liaison.	Software AEH

As shown in Table 6, the FAA has issued four Advisory Circulars (ACs) that define industry guidelines as acceptable means of compliance to the policies. Compliance is determined by the FAA or their delegates.

Table 6. Advisory circulars and industry guidelines

Advisory Circular	**Industry Guideline**
AC20-174 [10]	ARP 4754A [1]
AC20-170 [11]	DO-297 [5]
AC20-115C [12]	DO-178C [3]
AC20-152 [13]	DO-254 [4]

It is important that to clearly define the interfaces and dataflow between these guidelines.

4.1.1 Interrelationships for safety assessment and development assurance processes

Per ARP 4754A [1], as presented in Figure 27, the following data may be passed **to** the safety assessment processes **from** the airplane/system development assurance processes:

- Aircraft functions.
- System functions.
- Systems architecture.
- Systems implementation.
- Functional Development Assurance Level (FDAL).
- Item Development Assurance Level (IDAL).

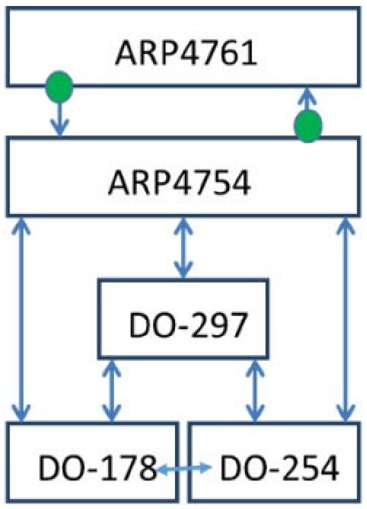


Figure 27. Interrelationships between ARP 4754 and ARP 4761

Per ARP 4754A [1], the following data may be passed **from** the safety assessment processes **to** the airplane/system development assurance processes:

- Failure condition, effects, classification.
- Safety requirements.
- Safety objectives.
- Architectural safety requirements (derived from safety analyses).

- Item requirements (derived from safety analyses).
- Separation requirements.
- Verification.

Function, failure, and safety information (particularly, derived safety requirements) flow from the ARP 4761 processes to the ARP 4754A processes. System design information flows from the ARP 4754A processes to the ARP 4761 processes.

4.1.2 Interrelationships between development and design assurance activities

The transition from development assurance processes to software and hardware design assurance processes occurs when the requirements are allocated to hardware and software items. This is when the transition occurs from ARP 4754 [7]/ARP 4754A [1] to DO-178 [3] and DO-254 [4].

4.1.3 Interrelationships between DO-178 and ARP 4754

Per DO-178C, the following data may be passed **from** the system development processes **to** the software lifecycle processes:

- Systems requirements allocated to software.
- System safety objectives.
- Software level of software components and a description of associated failure condition(s), if applicable.
- System description and hardware definition.
- Design constraints, including external interfaces, partitioning requirements, for example.
- Details of any system activities proposed to be performed as part of the software lifecycle. Note that system requirement validation is not usually part of the software lifecycle processes. The system lifecycle processes are responsible for assuring any system activities proposed to be performed as part of the software lifecycle.
- Evidence of the acceptability, or otherwise, of any data provided by the software processes to the system processes on which any activity has been conducted by the system processes. Examples of such activity are the system processes' evaluations of:
    - Derived requirements provided by the software processes to determine if there is any impact on the system safety assessment (SSA) and system requirements.

    - Issues raised by the software processes with respect to the clarification or correction of system requirements allocated to software.

- Evidence of software verification activities performed by the system lifecycle processes, if any.

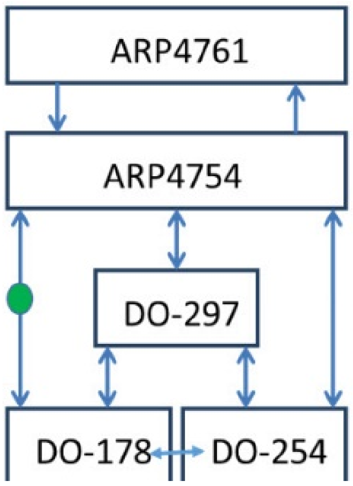


Figure 28. Interrelationships between DO-178 and ARP 4754

Per DO-178C, as noted with a green dot in Figure 28, the following data may be passed **to** the system development processes **from** the software lifecycle processes:

- Details of derived requirements created during the software lifecycle processes.
- A description of the software architecture, including software partitioning.
- Evidence of system activities performed by the software lifecycle processes, if any.
- Problem or change documentation, including problems identified in the system requirements allocated to software and identified incompatibilities between the hardware and software.
- Any limitations of use.
- Configuration identification and any configuration status constraints.
- Performance, timing, and accuracy characteristics.
- Data to facilitate integration of the software into the system.

4.1.4 Interrelationships between DO-254 and ARP 4754

Per DO-254 [4], as depicted with a green dot in Figure 29the following data may be passed **from** the system development processes **to** the airborne electronic hardware lifecycle processes:

- Design and safety requirements allocated to hardware.
- Design assurance level (DAL) for each function, along with associated requirements and failure conditions, if applicable.
- Allocated probabilities and at-risk exposure time for hardware functional failures.
- Hardware/software interface description.
- Requirements for safety strategies and design constraints, such as testability, design methods, and hardware architectures.
- Requirements for system verification activities to be performed by hardware-level verification.
- Installation, ergonomic, and environmental requirements allocated to hardware.
- Integration problem reports that may have impact on requirements. These may arise as a result of activities, such as system verification, generation of system requirements or SSA.

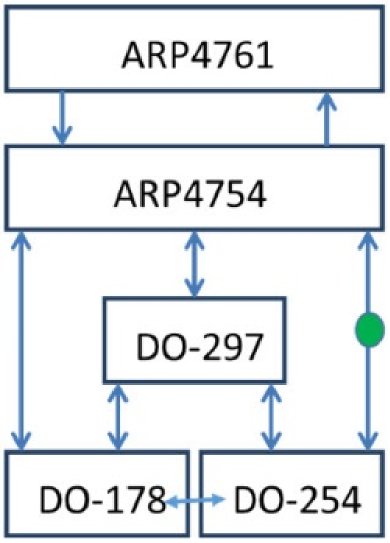


Figure 29. Interrelationships between DO-254 and ARP 4754

Per DO-254 [4], the following data may be passed **to** the system development processes **from** the airborne electronic hardware lifecycle processes:

- Implementation of the requirements, such as mechanical drawings, schematics, and parts lists.
- Hardware-derived requirements that may have an impact on any allocated requirement.
- Implementation architecture, including fault containment boundaries.
- Evidence of any required system verification and validation activities performed during the hardware design lifecycle.
- Product safety analysis data, such as:
    - Probabilities and failure rates for designated hardware functional failures of concern to the SSA process.
    - Common mode fault analysis.
    - Isolation boundaries and generic fault mitigation strategies.
    - Latency analysis data relevant to system requirements. Examples are hardware provisions for fault monitoring, fault detection intervals, and undetectable faults.
- Requirements for hardware verification activities to be performed by system-level verification.
- Assumptions and analysis methods regarding installation requirements and environmental conditions necessary for the analyses to be valid.
- Problem or change reports that have an impact on system, software, or allocated hardware requirements.

4.1.5 Interrelationships between DO-178 and DO-254

Per DO-178C, as noted with a green dot in Figure 30, the following data may be passed between the software and hardware processes:

- All requirements, including derived requirements, needed for hardware/software integration, such as definition of protocols, timing constraints, and addressing schemes for the interface between hardware and software.
- Instances where hardware and software verification activities requiring coordination.
- Identified incompatibilities between the hardware and the software.

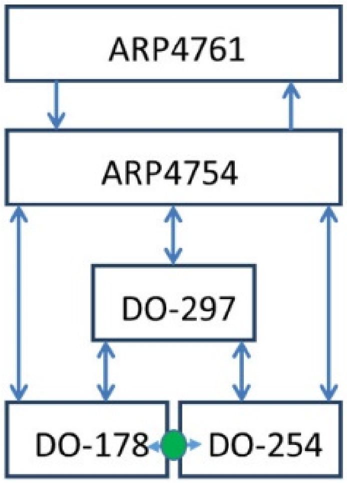


Figure 30. Interrelationship between DO-178 and DO-254 [4]

Table 7 identifies the data connectivity between safety, development and design assurance processes. Any time that there is an interface and/or information flow, the possibility exists for an error or omission to be introduced. This is one of the reasons why MBSE/MBSA methods are being considered to improve the integration of the required data for the safety, development, and design assurance processes. The goal is to use a more integrated data set to help manage the data connectivity.

Table 7. Data connectivity between safety, development and design assurance processes

Sends Data	**ARP 4761**	**ARP 4754A**	**DO-178**	**DO-254**
ARP 4761		Failure condition, effects, classification. Safety requirements. Safety objectives. Architectural safety requirements (derived from safety analyses). Item requirements (derived from safety analyses). Separation requirements. Verification.		

Sends Data	ARP 4761	ARP 4754A	DO-178	DO-254
ARP 4754A [7]	Aircraft functions. System functions. Systems architecture. Systems implementation. Functional Development Assurance Level (FDAL). IDAL.		Systems requirements allocated to software. System safety objectives. Software level of software components and a description of associated failure condition(s). System description/hardware definition. Design constraints. Details of any system activities proposed to be performed as part of the software lifecycle. Evidence of the acceptability. Evidence of software verification activities performed by the system lifecycle processes, if any.	Design and safety requirements allocated to hardware. DAL for each function, along with associated requirements and failure conditions. Allocated probabilities and at-risk exposure time for hardware functional failures. Hardware/software interface description. Requirements for safety strategies and design constraints. Requirements for system verification activities. Installation, ergonomic and environmental requirements allocated to hardware. Integration problem reports.
DO-178		Details of derived requirements created during the software lifecycle processes. A description of the software architecture, including software partitioning.		All requirements, including derived requirements, needed for hardware/software integration, such as definition of protocols, timing constraints, and addressing schemes for the

Sends Data	ARP 4761	ARP 4754A	DO-178	DO-254
		Evidence of system activities performed by the software lifecycle processes, if any. Problem or change documentation, including problems identified in the system requirements allocated to software and identified incompatibilities between the hardware and software. Any limitations of use. Configuration identification and any configuration status constraints. Performance, timing, and accuracy characteristics. Data to facilitate integration of the software into the system.		interface between hardware and software. Instances where hardware and software verification activities requiring coordination. Identified incompatibilities between the hardware and the software.
DO-254 [4]		Implementation of the requirements. Hardware derived requirements. Implementation architecture. Evidence of any required system verification and	All requirements, including derived requirements, needed for hardware/software integration, such as definition of protocols, timing constraints, and addressing schemes for the interface	

Sends Data	ARP 4761	ARP 4754A	DO-178	DO-254
		validation activities. Product safety analysis data. Requirements for hardware verification activities. Assumptions and analysis methods regarding installation requirements and environmental conditions. Problem or change reports.	between hardware and software. Instances where hardware and software verification activities requiring coordination. Identified incompatibilities between the hardware and the software.	

4.1.6 DO-297

DO-297 [5] provides guidance on how to do design assurance on an IMA system (as opposed to traditional federated systems architectures). An IMA architecture provides a set of shared computing, networking, and input/output resources supporting the computing and system interface needs for multiple airplane systems. An IMA architecture integrates airplane systems, such as avionics, display/flight management systems, electrical systems, environmental control systems, mechanical/hydraulics systems, and many others. The IMA platform is defined and developed independently from the specific aircraft functions and the hosted applications, making it possible for the developed IMA platform to be reusable with different sets of hosted applications.

Per DO-297, the following are key platform characteristics of an IMA architecture:

- Platform resources are shared by multiple applications.
- An IMA platform provides robust partitioning of shared resources.
- An IMA platform only allows hosted applications to interact with the platform and other hosted applications through well-defined interfaces.
- Shared IMA platform resources are configurable.

Per DO-297, the following are key application characteristics of an IMA architecture:

- An application may be designed independent of other applications and obtain incremental acceptance on the IMA platform independently of other applications.
- Applications can be integrated onto a platform without unintended interactions with other hosted applications.
- Applications may be reusable.
- Applications are independently modifiable.

The IMA platform consists of the modules and the connecting infrastructure. The application-ready IMA platform refers to the means of structuring, connecting and combining modules to support the requirements of the hosted applications and aircraft functions. The integrated IMA platform consists of the IMA platform and the hosted applications. The IMA architecture consists of the IMA platform, connections, and components that satisfy the requirements of all of the hosted applications and aircraft functions.

The certification process for IMA systems has six incremental tasks, as follows:

- Module acceptance.
- Application software or hardware acceptance.
- IMA system acceptance.
- Aircraft integration of IMA system, including V&V.
- Change of modules or applications.
- Reuse of modules or application.

4.2 Potential process shortcomings (particularly related to aircraft/system integration and modifications to aircraft or systems)

Systems integration becomes increasingly important as systems architectures evolve from federated architectures to integrated, distributed systems architectures. Figure 31 shows the same function implemented by both a federated systems architecture and an IMA architecture. Due to the significantly lower number of cross-functional interfaces, it is generally easier for a single designer or team to define, validate, and verify the interfaces for a federated systems architecture. In addition, it is generally easier for a single designer or team to understand the failure behavior.

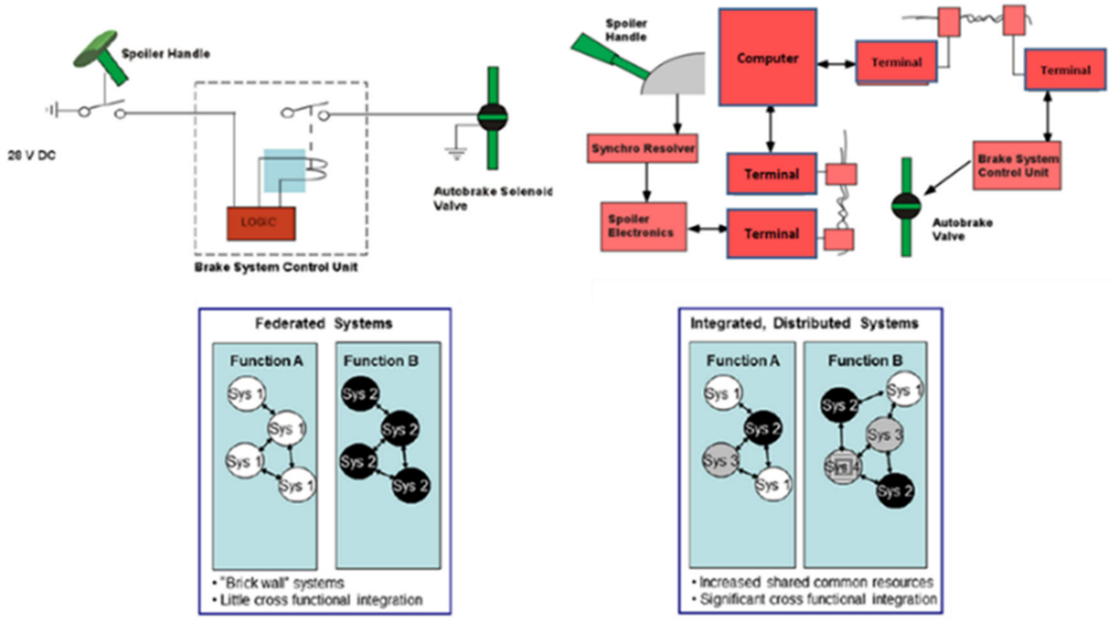


Figure 31. Federated versus integrated, distributed systems

This is different from integrated, distributed systems architectures in which there are a great number of cross-functional interfaces and in which there is increased sharing of common computing and data transmission resources. This increases the importance of conducting systems integration in early design development phases.

However, existing industry guidance is still generally geared more toward federated systems architectures than integrated, distributed systems architectures. For example, ARP 4754A [1] section 4.6.4 Aircraft/System Integration states: "Normally, systems integration begins with item by item integration and progresses to complete system integration." This implies that systems integration occurs on the right hand side of the SE "V." However, with integrated, distributed systems architectures, it important to consider systems integration during the early development stages (the left side of the SE "V").

As shown in Figure 32, there is a need to have an integrated architecture at the very beginning. It is important to understand the bigger picture of the puzzle before decomposing into the lower level (i.e., item level) pieces. Systems integration needs to be "baked in" from the beginning; it should not start after the items have been developed.

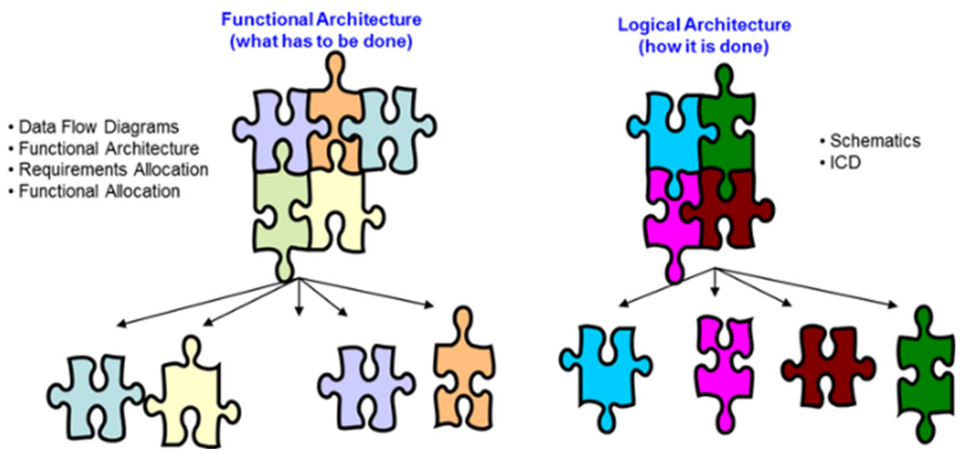


Figure 32. Required integrated picture

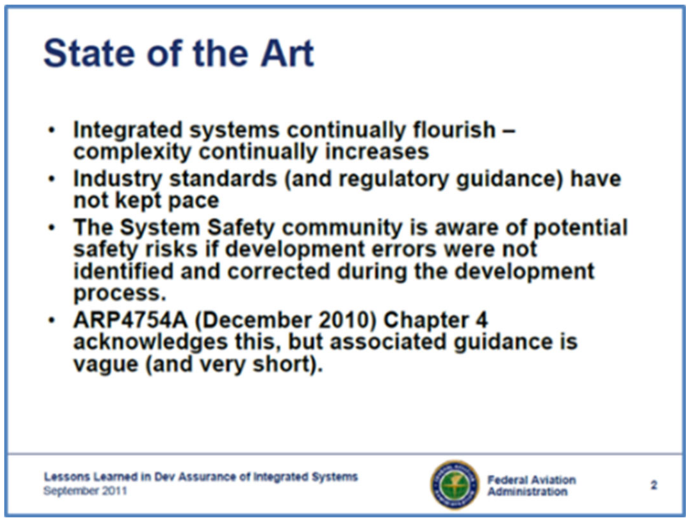
Creating an integrated functional architecture is an approach that can help improve earlier integration. The functional architecture will be more stable than the logical architecture. Creating a stable functional architecture will minimize logical architecture rework.

To minimize rework, the aircraft should be functionally and logically integrated prior to detailed design work (particularly at the item level). They both need to be accomplished and integrated; the logical architecture alone will not identify what functionality is lost when a component fails or a signal is interrupted.

As described in Table 7, key outputs of the ARP 4754A [1] processes include the aircraft functions, the systems-level functions and the systems architecture. These are integral inputs to the safety processes and the software and hardware design assurance processes.

Establishing an integrated systems architecture (particularly for increasingly integrated, distributed architectures) is a key factor in helping make sure that there is a complete and correct set of requirements. This is important for requirements related to the interoperability/integration between systems (interaction between two or more systems working to implement a higher-level function). Interoperability/integration is the manner in which these systems are interconnected and the way each system contributes to the top-level function.

With the increasing level of integration between aircraft functions and the systems that implement them, one system may impose requirements on other systems (e.g., performance, design constraints). Other examples include identifying the requirements associated with maintaining airplane functionality with power transitions (e.g., power up and power down, in-flight power down and power up, power interruptions, transitions and power quality variations). If this is not done correctly, it can increase the possibility of a development error related to systems integration.


State of the Art

- Integrated systems continually flourish – complexity continually increases
- Industry standards (and regulatory guidance) have not kept pace
- The System Safety community is aware of potential safety risks if development errors were not identified and corrected during the development process.
- ARP4754A (December 2010) Chapter 4 acknowledges this, but associated guidance is vague (and very short).

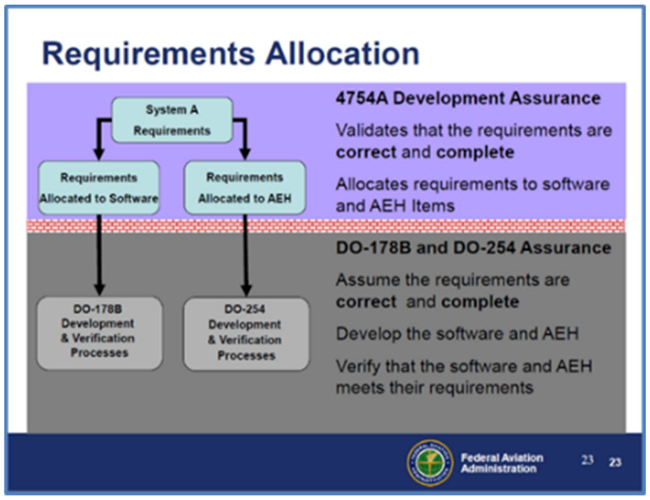
Lessons Learned in Dev Assurance of Integrated Systems
September 2011

Federal Aviation Administration 2

Figure 33. FAA comment on ARP 4754A [1] systems integration

However, as the FAA has acknowledged (see Figure 33), the “associated guidance is vague (and very short).”

This is important because DO-178 and DO-254 [4] are predicated on the assumption that the allocated requirements (see Figure 34) are complete and correct.


Requirements Allocation

System A Requirements
Requirements Allocated to Software
Requirements Allocated to AEH
DO-178B Development & Verification Processes
DO-254 Development & Verification Processes

4754A Development Assurance

Validates that the requirements are **correct** and **complete**

Allocates requirements to software and AEH Items

DO-178B and DO-254 Assurance

Assume the requirements are **correct** and **complete**

Develop the software and AEH

Verify that the software and AEH meets their requirements

Federal Aviation Administration 23 23

Figure 34. FAA training on ARP 4754A relationship to DO-178/254

Note: There is potential room for improvement in industry guidance related to systems integration. However, this should not be construed to mean that companies have not taken the required steps to address systems integration.

It would help to clarify the three different types of integration, including the required objectives for each:

- Vertical.
- Horizontal.
- Diagonal.

4.2.1 Vertical integration

Vertical traceability, as shown in Figure 35, exists within each hierarchy (requirement, functional, logical, and physical). Going from higher level to lower levels within a hierarchy is usually referred to as decomposition. The purpose of the vertical integration analysis is to ensure that:

- The lower level supports the higher level.
- There is traceability.
- Totality of the different levels are complete, consistent, and correct within the architecture (e.g., within requirements architecture).

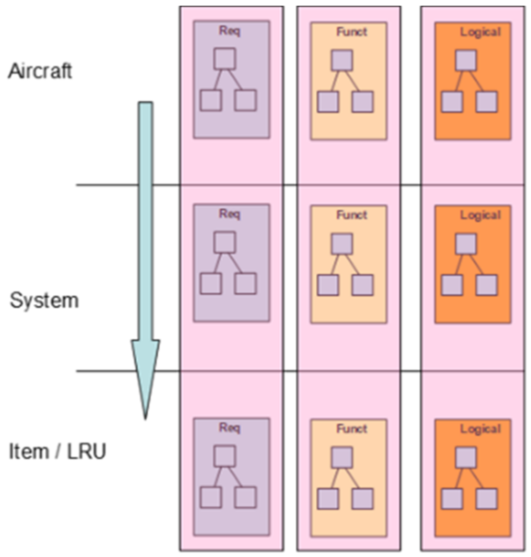


Figure 35. Vertical integration

4.2.2 Horizontal integration

Horizontal allocation/traceability, as shown in Figure 36, exists at each tier level (e.g., aircraft, system, or item). Going from architecture to architecture at the same tier level is usually referred to as allocation (e.g., requirements allocation, functional allocation). The purpose of the horizontal integration analysis is to ensure the following:

- There is an integrated product architecture at each tier level:
  - Requirements completeness (e.g., missing requirements at the aircraft level, clarity of cumulative airplane-level requirements).
  - Functional completeness.
  - Logical completeness.
- Totality of the requirements, functional and logical architectures are complete, consistent and correct at a given tier level (e.g., at the aircraft level).
- In an ideal world, this work would be completed prior to moving to the lower level tiers (for example, you would have an integrated airplane product architecture before moving to the system level).

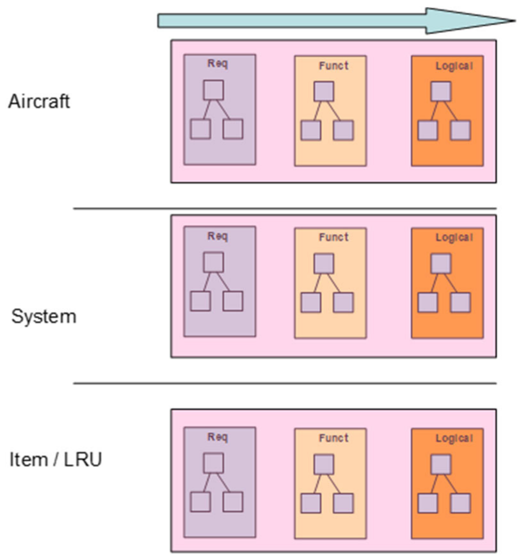


Figure 36. Horizontal integration

4.2.3 Diagonal integration

Diagonal allocation/traceability, as shown in Figure 37, goes from one tier level to a lower tier level (e.g., from aircraft to systems). Going from the logical architecture at one tier level to the requirements architecture at a lower tier level is usually referred to as derived.

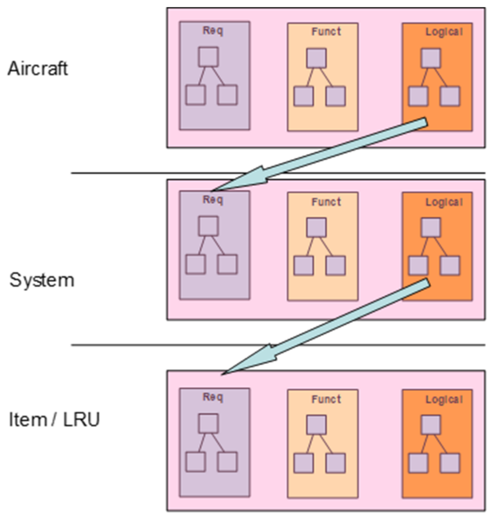


Figure 37. Diagonal integration

Derived requirements are additional requirements resulting from design or implementation decisions during the development process, which are not directly traceable to higher-level requirements.

Derived requirements stemming from technology, architecture, system and item interfaces, or implementation choices become more clearly visible, as work on the system architecture progresses. At each phase of the development activity, decisions are made as to how particular requirements or groups of requirements are to be met. The consequences of these design choices become requirements for the next phase of the development. Since these requirements result from the design process itself, they may not be uniquely related to a higher-level requirement and are referred to as derived requirements.

Note: These architectural decisions were made as part of the horizontal integration at the higher tier level.

Derived requirements should be examined to determine which aircraft-level function (or functions) they support so that the appropriate Failure Condition classification can be assigned and the requirement validated. While derived requirements will not impact the higher-level requirements, some may have implications at higher levels. Derived requirements should be reviewed from a safety perspective at progressively higher system levels, until it is determined that no further impact is propagated.

Examples of derived requirements include:

- Derived requirements may result from the decision to select a separate power supply for equipment performing a specific function. The requirements for the power supply, including the safety requirements, are derived requirements. The failure condition resulting from the fault or failure of the function supported by the power supply determines the necessary development assurance level.

- Derived requirements may also result from architecture choices. For example, selecting a triplex architecture for achieving a high integrity functional objective would have different consequences and different derived requirements from selection of a dual monitored architecture for achievement of the same objective.

Derived requirements may result from a design decision to isolate function implementations having more severe Failure Condition classifications from the failure effects of systems having less severe Failure Condition classifications.

Derived requirements also include those defining the hardware-software interface. Some of these requirements may be significant at the system level. The remainder, dealing with detailed aspects of the hardware-software interface, may be handled under the guidance of DO-178/ED-12 and DO-254 [4]/ED-80.

4.2.4 Modifications to aircraft/system

Section 6 of ARP 4754A [1] addresses modifications to the aircraft or systems. Aircraft projects typically involve a mixture of new, modified, and reused systems. It is rare for an aircraft project to be a completely "clean sheet." Section 6 of ARP 4754A includes the following:

- Introducing a new aircraft-level function.
- Replacing an item or system with another on an existing aircraft.
- Adapting an existing item or system to a different aircraft type.
- Modification to an item or system without adding a function.
- Supplemental Type Certificate (STC) production introduction.

However, ARP 4754A does not clearly communicate which of the development assurance processes need to be conducted and which development assurance artifacts need to be created/updated, depending on the type and scope of the project. It would be beneficial to have a consistent determination process, based on the scope of the change, to identify which of the following processes (and their associated artifacts) need to be accomplished:

- Requirements management.
- Safety assessment process.

- Requirements validation.
- Implementation verification.
- Configuration management.
- Process assurance.

This determination process should involve consideration of certification basis, system similarity, and system complexity to identify the appropriate techniques to apply to the system(s) for the aircraft's development.

4.3 Integral systems integration processes (potential process gaps related to safety)

ARP 4754A [1] identifies that the following should be considered during a modification impact analysis (which would typically be captured as part of the requirements, functional or logical architectures):

- Safety related information is changed; for example:
    - Failure condition classification(s) changed or added.
    - Development assurance level.
    - Safety margins are reduced.
    - Architectural assumptions.
    - Validity of the environmental qualification test results is affected.
    - V&V methods or procedures are modified.
- Operational or procedural characteristics are changed; for example:
    - Aircraft operational or airworthiness characteristics.
    - Flight crew procedures.
    - Increased pilot workload.
    - Situational awareness, warnings, cautions, or advisories.
    - Displayed information to make flight decisions.

However, improvements to industry guidance could be made by also including considerations of change impact to the physical architecture. For example, it would it be beneficial to also consider:

- Integration of functional/logical architectures and the physical architectures are maintained.
- Physical location of equipment in threat zones does not violate required functional independence, redundancy, for example.
- Appropriate physical separation is maintained (e.g., three redundant systems are located in the same threat zone or unacceptable loss of downstream systems functionality because upstream systems were not properly separated).

Figure 38 shows the integration of the logical and physical to assess safety impact.

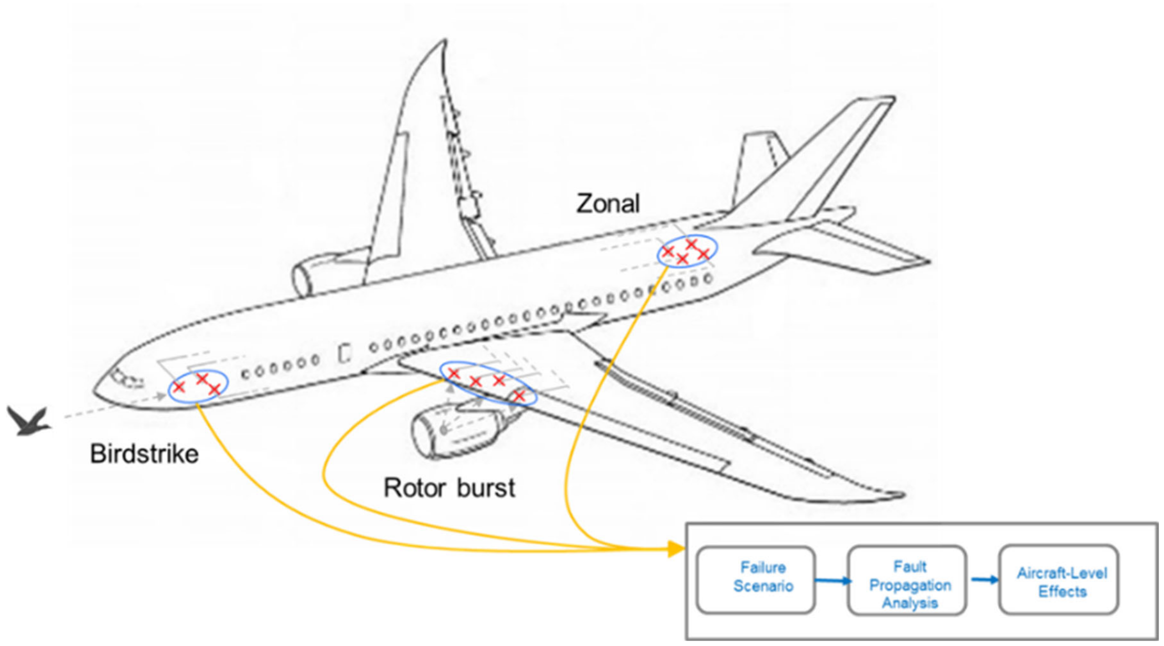


Figure 38. Integrating logical and physical to assess safety impact

It is admittedly extremely unlikely that a designer would put the three redundant systems in the same threat zone. However, as airplanes become more integrated, there can be unacceptable cascading failure effects that might not be immediately apparent by just looking at the initial source of failure (e.g., equipment in the threat zone). Integrating the physical architecture with the requirements/functional/logical architectures can help adequate safety be maintained during modification impact analysis.

It would be beneficial to emphasize that modifications should consider potential impacts to Particular Risk Assessments (PRA) or Zonal Safety Analysis (ZSA).

The unreleased ARP 4761A will have a new appendix related to "Cascading Effects Analysis," which is a qualitative, bottom-up analysis method that evaluates an initiating condition (e.g., a failure condition, failure mode, or combination of failure modes) and captures the total effect on the aircraft for that initiating condition.

Figure 39 is a simplified diagram that shows the results of the cascading failure effects of electrical component failures. The safety analysis needs to look at the cumulative effects of all systems-level effects. For a highly integrated, distributed systems architecture, it is possible for the cumulative effect of acceptable systems-level effects (acceptable systems-level effect due to loss of functionality or loss of redundancy) to be catastrophic at the airplane level.

Note: This example below is for illustrative purposes only; aircraft systems would not be designed and certified this way.

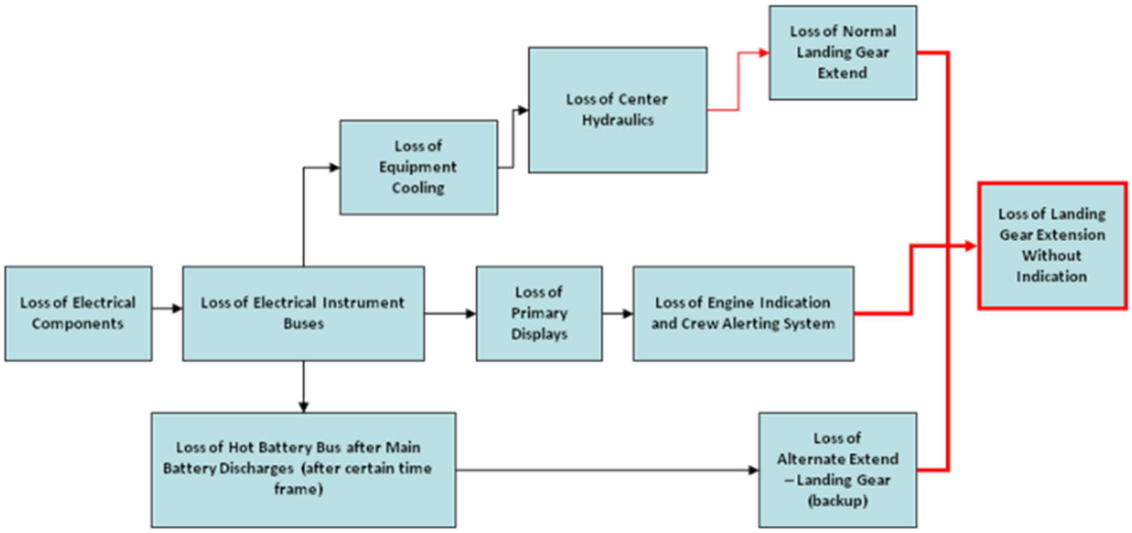

Loss of Electrical Components
Loss of Electrical Instrument Buses
Loss of Equipment Cooling
Loss of Center Hydraulics
Loss of Normal Landing Gear Extend
Loss of Primary Displays
Loss of Engine Indication and Crew Alerting System
Loss of Hot Battery Bus after Main Battery Discharges (after certain time frame)
Loss of Alternate Extend – Landing Gear (backup)
Loss of Landing Gear Extension Without Indication

Figure 39. Unacceptable, cumulative cascading failure effects

Similarly, it is important to evaluate the stack-up of degradation in each of the airplane-level functions to determine the overall airplane-level hazard categorization (as shown in Figure 40). Similar to the example above (in which the cumulative effect of acceptable systems-level effects can be catastrophic at the airplane level), it is possible that the cumulative effect of each acceptable airplane-level function may not be acceptable at the airplane level.

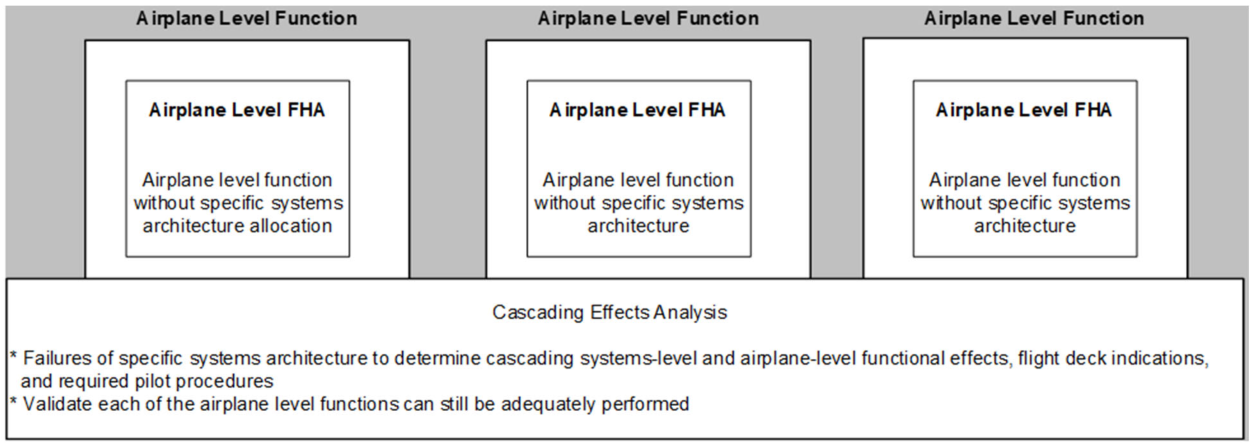

Airplane Level Function
Airplane Level FHA
Airplane level function without specific systems architecture allocation
Airplane Level Function
Airplane Level FHA
Airplane level function without specific systems architecture
Airplane Level Function
Airplane Level FHA
Airplane level function without specific systems architecture
Cascading Effects Analysis
* Failures of specific systems architecture to determine cascading systems-level and airplane-level functional effects, flight deck indications, and required pilot procedures
* Validate each of the airplane level functions can still be adequately performed

Figure 40. Cascading effects analysis – multi-airplane-level function assessment

The following example shows how this can occur, using the airplane-level functions in Figure 41. Consider a failure that results in the following:

- Floating flight controls surfaces (impacting "Control Lift and Drag").
- Loss of cabin depressurization (impacting "Pressurization").
- Loss of wing ice protection (impacting "Emergency Prevention & Management").

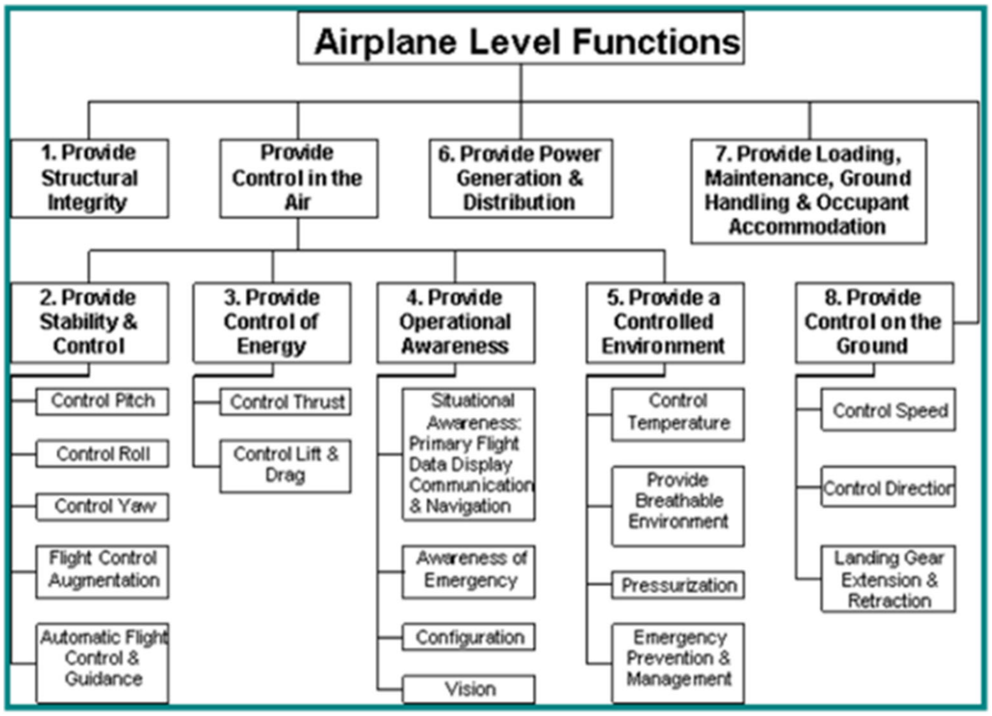
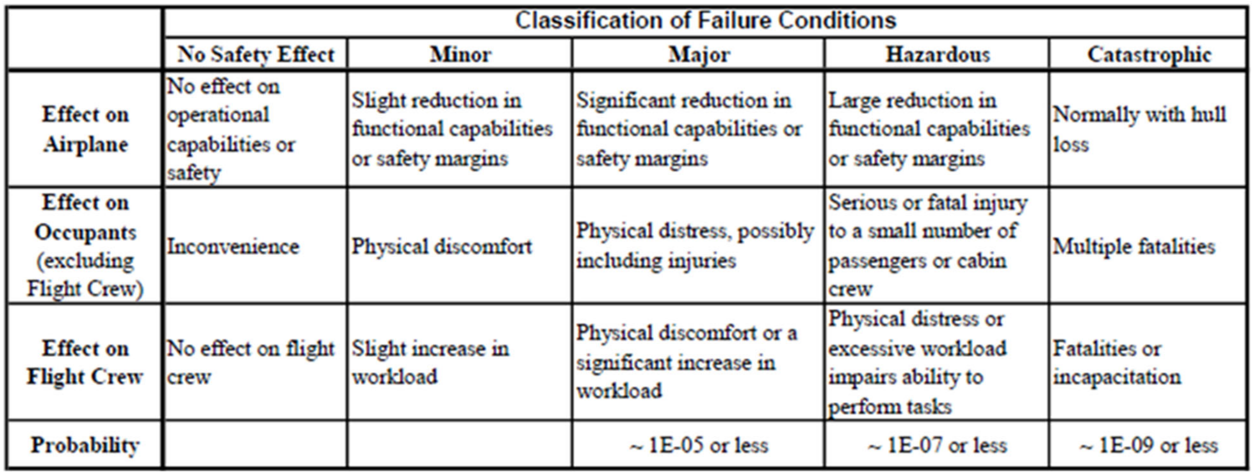

Airplane Level Functions
1. Provide Structural Integrity
Provide Control in the Air
6. Provide Power Generation & Distribution
7. Provide Loading, Maintenance, Ground Handling & Occupant Accommodation
2. Provide Stability & Control
Control Pitch
Control Roll
Control Yaw
Flight Control Augmentation
Automatic Flight Control & Guidance
3. Provide Control of Energy
Control Thrust
Control Lift & Drag
4. Provide Operational Awareness
Situational Awareness: Primary Flight Data Display Communication & Navigation
Awareness of Emergency
Configuration
Vision
5. Provide a Controlled Environment
Control Temperature
Provide Breathable Environment
Pressurization
Emergency Prevention & Management
8. Provide Control on the Ground
Control Speed
Control Direction
Landing Gear Extension & Retraction

Figure 41. Airplane-level functions

In this hypothetical scenario, it is possible for there to be an unacceptable drag count due to increased drag after descending to 10,000 feet (as a result of the decompression), floating spoiler pairs, and potential ice accumulation on the wing (if unable to avoid icing conditions). The increased drag count could preclude the airplane from reaching a suitable airport. Although this example is partially due to the structure of the airplane functional decomposition, the point is still warranted. Other airplane-level functional decompositions with different structures could similarly result in the cumulative effect of each acceptable individual airplane-level function being unacceptable.

In addition, ARP 4761 could further expand on the human-crew interface to help evaluate that the crew workload is acceptable. Figure 42 is based on material in the Arsenal version of AC/AMJ 25.1309. Three items need to be considered in assessing the overall effect:

- Effect on the airplane.
- Effect on occupants.
- Effect on flight crew.



	Classification of Failure Conditions				
	No Safety Effect	Minor	Major	Hazardous	Catastrophic
Effect on Airplane	No effect on operational capabilities or safety	Slight reduction in functional capabilities or safety margins	Significant reduction in functional capabilities or safety margins	Large reduction in functional capabilities or safety margins	Normally with hull loss
Effect on Occupants (excluding Flight Crew)	Inconvenience	Physical discomfort	Physical distress, possibly including injuries	Serious or fatal injury to a small number of passengers or cabin crew	Multiple fatalities
Effect on Flight Crew	No effect on flight crew	Slight increase in workload	Physical discomfort or a significant increase in workload	Physical distress or excessive workload impairs ability to perform tasks	Fatalities or incapacitation
Probability			~ 1E-05 or less	~ 1E-07 or less	~ 1E-09 or less

Figure 42. Hazard category classifications

It would be helpful if ARP 4761 provided additional guidance on assessing the interactions between the flight crew and systems in the presence of failures. While this is less important for federated systems architectures, it becomes more important for integrated, distributed architectures, in which many systems functions that were typically separated with limited interdependence are now interrelated and highly integrated. It is important to validate that the flight crew will have sufficient remaining functionality to cope with failures. Considerations could include:

- Validate the flight crew has the required system functionality to address the failure (e.g., cursor control device is available to take required actions).
- Validate the flight crew has the ability to view all messages (e.g., page control).
- Validate that adequate messages are displayed for airplane system and operational awareness.
- Validate that the crew procedures associated with messages adequately address safety concerns.
- Validate that the crew procedures do not provide conflicting direction (e.g., one procedure states to maintain altitude, another procedure states to descend immediately).

These types of flight crew-systems interaction considerations could be made during the Cascading Effects Analyses.

4.4 Potential common errors in interrelationships between industry standards

Figure 43 identifies the interrelationships between the safety, development assurance, and software/hardware design assurance processes. Similar to managing the interfaces during the design of an actual product, it is also important to manage the process interfaces. One of the key interrelationships are between the safety and the development assurance processes.

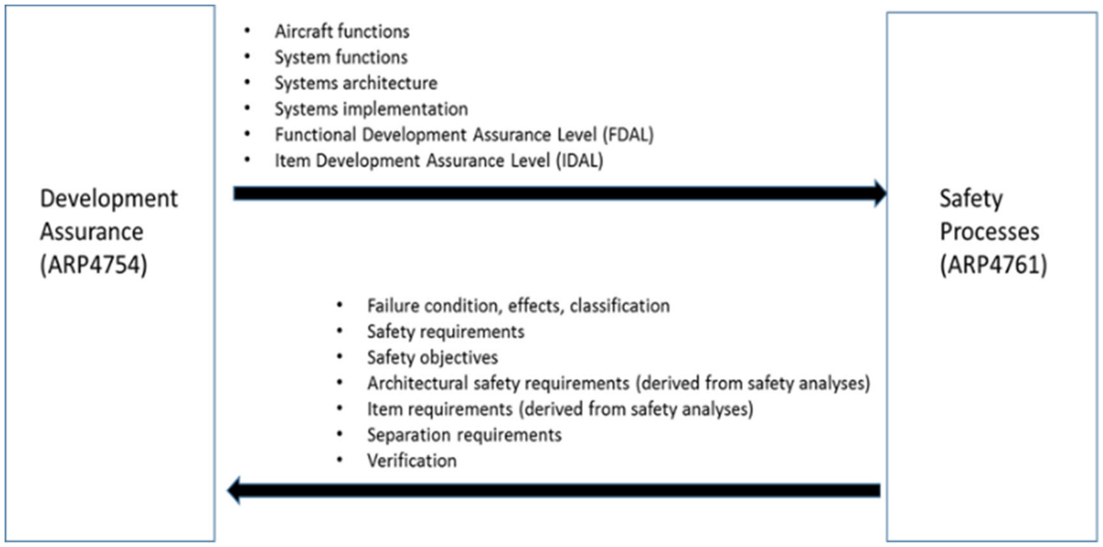
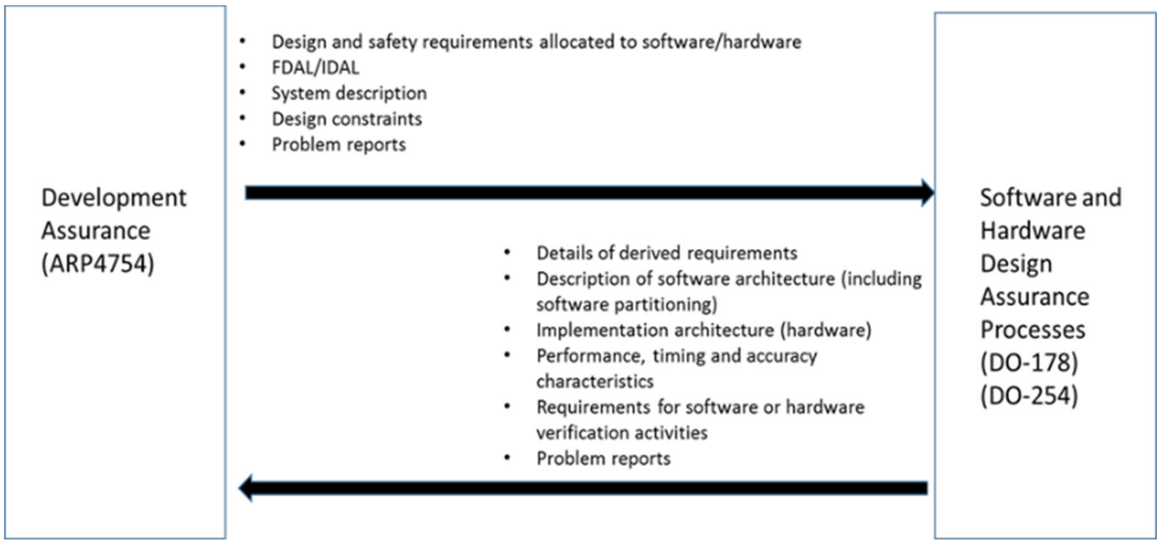

Development Assurance (ARP4754)
Safety Processes (ARP4761)
• Aircraft functions
• System functions
• Systems architecture
• Systems implementation
• Functional Development Assurance Level (FDAL)
• Item Development Assurance Level (IDAL)
• Failure condition, effects, classification
• Safety requirements
• Safety objectives
• Architectural safety requirements (derived from safety analyses)
• Item requirements (derived from safety analyses)
• Separation requirements
• Verification

Figure 43. Safety and development assurance interfaces

If the functions are not captured during the development assurance process, potential failure conditions and hazard categories will not be assigned during the safety processes. As a result, this increases the risk that the DALs may not be correct.

Figure 44 shows the critical interrelationships between the airplane and systems development assurance processes and the software/hardware design assurance processes, which include:

- Systems architecture.
- Allocated requirements (to software and hardware).



Development Assurance (ARP4754)
Design and safety requirements allocated to software/hardware
FDAL/IDAL
System description
Design constraints
Problem reports
Software and Hardware Design Assurance Processes (DO-178) (DO-254)
Details of derived requirements
Description of software architecture (including software partitioning)
Implementation architecture (hardware)
Performance, timing and accuracy characteristics
Requirements for software or hardware verification activities
Problem reports

Figure 44. Development and design assurance interfaces

The software and hardware design assurance processes are assumed to start with a complete and correct set of allocated requirements. No one intentionally has incomplete or incorrect requirements. SAE ARP 4754 [7] acknowledged it is virtually impossible to validate requirements are complete and correct for complex systems. Subsequent sections will evaluate how modeling can potentially improve the quality of the requirements. Models can potentially address some of the limitations of text-based requirements.

4.5 DO-331 (development and verification supplement for DO-178C and DO-278A)

DO-331 [6] was published in 2011 as a supplement to DO-178C and must be met for model-based software development processes. Per DO-331, regardless of where the model development started (i.e., system or software domain), guidance of DO-331 will apply. This implies that systems must meet both the ARP 4754A [1] and several DO-331 objectives. It would be helpful to clarify these boundaries so that design teams can coordinate and plan to meet the two standards efficiently, without duplicating systems and software team efforts.

4.6 Modeling standards

Unified Modeling Language (UML) and Systems Modeling Language (SysML) are language specifications (which is different from a modeling specification).

UML is used in software engineering as a way to provide visualization of a system's design. UML diagrams (as pictured in Figure 45) have two different views of a system model:

- Behavior (used to describe the functionality).
- Structure (used to document the software architecture).

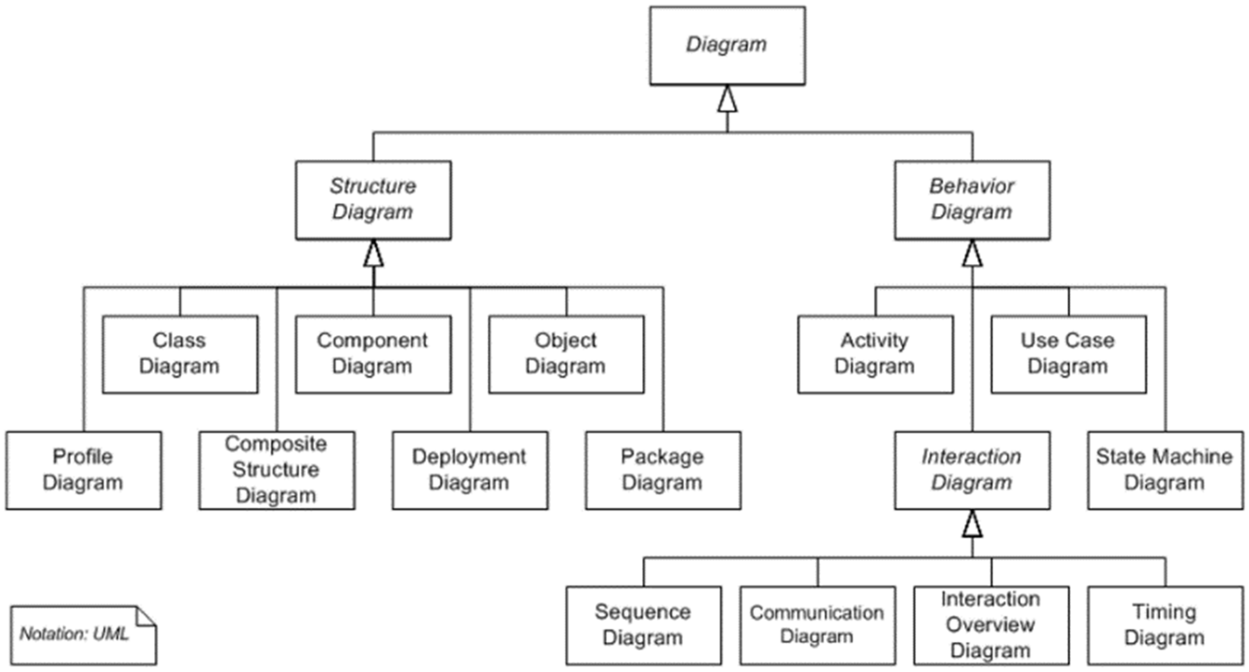


Figure 45. UML diagrams

SysML is a graphical modeling language for SE applications. Per the Object Management Group (OMG), SysML is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametrics, which are used to integrate with other engineering analysis models.

5 Process execution

This section of the report focuses on the assessment of the current guidance and standards for IMA, DIMA, and other distributed system architectures. Table 8 describes the approach to address these research objectives.

Table 8. Current guidance and research approach

Research Objectives	**Detailed Work Description**
Assess current guidance and standards when Integrated Modular Avionics (IMA) systems, Distributed Integrated Modular Avionics (DIMA) systems, and other distributed systems and architectures.	Conduct analysis on process execution problems: Survey Boeing's Subject Matter Experts, including those who have work experiences as Authorized Representative (AR) Advisors and ARs. The SMEs will have experiences across multiple programs, multiple design disciplines and multiple suppliers. Identify potential gaps in the development/design assurance processes for development programs (which is addressed by ARP 4754A [1]). Assess frameworks for highly integrated, distributed platform systems. Identify gaps in existing V&V processes for highly integrated, distributed platform systems. Analyze integration related problem reports and determine root causes. Investigate how a potential use of virtualization, design of experiments, and distributed test could mitigate integration and safety issues for process execution problems.

5.1 Process execution feedback

For the past decade, Boeing has worked on:

- Development assurance on multiple development programs.
- Planning documents that identify the development assurance activities that will take place when implementing production-type design changes after the initial Type Certificate (TC) or Amended Type Certificate (ATC) has been issued.
- The SAE S-18 committee responsible for ARP 4754A [1].
- Development of Aerospace Information Report (AIR) 6110, Contiguous Aircraft/System Development Process Example, which provides a contiguous example of the design development process and shows the relationships between ARP 4754A and ARP 4761.

These activities have resulted in gaining experiences related to the theory of development assurance, its application throughout industry, and any potential differences.

Interviews were conducted with SMEs with the following development and design assurance experiences:

- Software (those with experience as ARs).
- AEH (those with experience as ARs).
- Boeing enterprise designated experts in requirements management.
- Flight test.

It is of significant value that highly experienced engineers are able to provide feedback on real-world occurrences of development/design assurance. This includes understanding the operational aircraft/system impacts that may be based on requirements issues; it includes understanding the processes, tools, and techniques used. The SMEs collectively have significant experiences applying the following:

- ARP 4754A [7]].
- ARP 4761 [2].
- DO-178 [3].
- DO-254 [4].
- DO-297 [5].
- DO-331 [6].

5.2 Potential gaps in industry development/design assurance processes

Based on experiences conducting development assurance on multiple programs and discussions with development assurance SMEs, a consistent theme was the importance of a complete and correct set of requirements. There is always risk for late changes when any of the following occur:

- Incomplete requirements.
- Incorrect requirements.
- Missing requirements.

For DIMA architectures, these requirement errors or omissions can occur if the development assurance plans and activities do not consider the integrated nature of the systems architecture.

It should be noted that ARP 4754A [1] does not explicitly mention Interface Control Document (ICD), Aircraft Systems Integration Plan, or Systems Integration Plan. AIR 6218, Constructing Development Assurance Plan for Integrated Systems, discusses the importance of developing an integrated approach as early as the planning phase of the development assurance activity. However, it does not provide a framework for doing some of the critical integration activities. These are potential areas of improvement in industry guidance.

5.3 Frameworks for highly integrated, distributed platforms

As discussed in section 4.2, the FAA has acknowledged that the systems integration guidance in ARP 4754A [1] is "short and ambiguous" (reference Figure 33).

In order to ensure a complete and correct set of requirements, it is important to have a framework for highly integrated, distributed systems. To address this, the research first focused on the required integration activities. It is critical to understand how a system is supposed to behave and ensure that it will work with interfacing systems. The following objectives need to be addressed:

- System architecture is complete, accurate, necessary, and sufficient.
- Potential discrepancies have been identified and addressed.

It is important to recognize the inherently iterative nature of development. There are different levels of abstraction that are required at different time frames. In addition, some systems have longer lead times than others do. Some data can and should be captured as soon as possible, but other data (e.g., detailed Built-In Test Equipment [BITE] reports) cannot be fully defined and validated until lower level design is underway. There does not appear to be an existing framework to show how to achieve this incremental, recursive integration, particularly for integrated, distributed systems architectures.

Figure 46 provides a framework to help address the incremental, iterative aspects of integration.

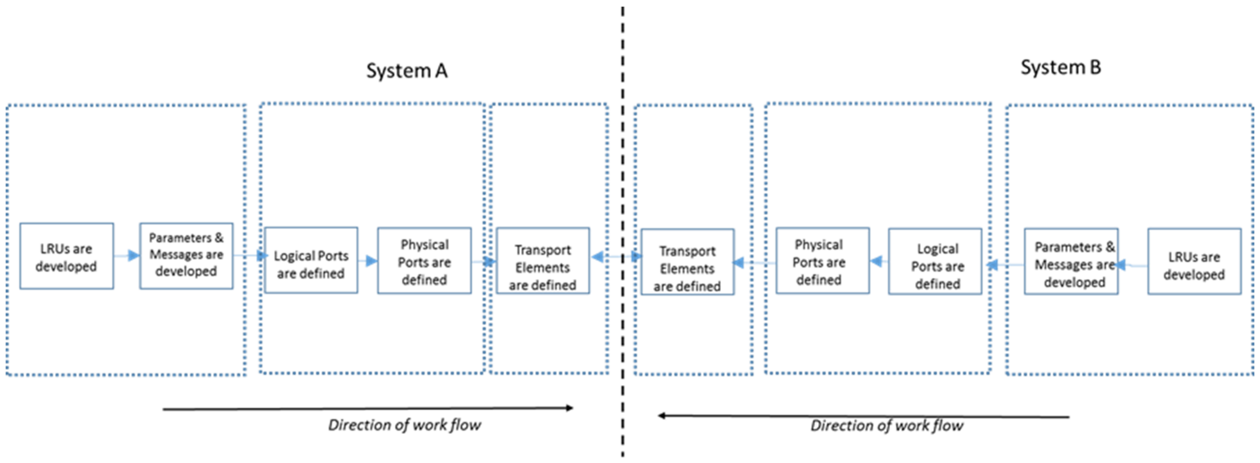

System A
System B
LRUs are developed
Parameters & Messages are developed
Logical Ports are defined
Physical Ports are defined
Transport Elements are defined
Transport Elements are defined
Physical Ports are defined
Logical Ports are defined
Parameters & Messages are developed
LRUs are developed
Direction of work flow
Direction of work flow

Figure 46. Cross-functional integration for a federated system

The first step, as shown in Figure 47, is the incremental architecture development. This includes the following steps:

- Architecture Definitions and Usages.
    - Validate that the hardware/software definitions that compromise your system are complete, accurate, and sufficient (e.g., Air Data Module exists).
    - Validate that the number of "usages" to meet redundancy and other architectural considerations are complete, accurate, sufficient, and necessary (e.g., Air Data Module-Left, Air Data Module-Center, and Air Data Module-Right are accurately captured). The number of usages needs to be consistent with the required redundancy management, reliability, availability, for example.
- System Interface Matrices.
    - Validate the subsystems that are publishing to your system (publishing systems) (i.e., you are receiving correct inputs from the correct subsystems).
    - Validate the subsystems that are subscribing to your system (subscribing systems) (i.e., you are providing the required outputs needed by other systems).
    - Validate the number of parameters in the system interface matrices:
        - Identify and resolve:
        - Under subscription.
        - Over subscription.

- Incorrect subscription.

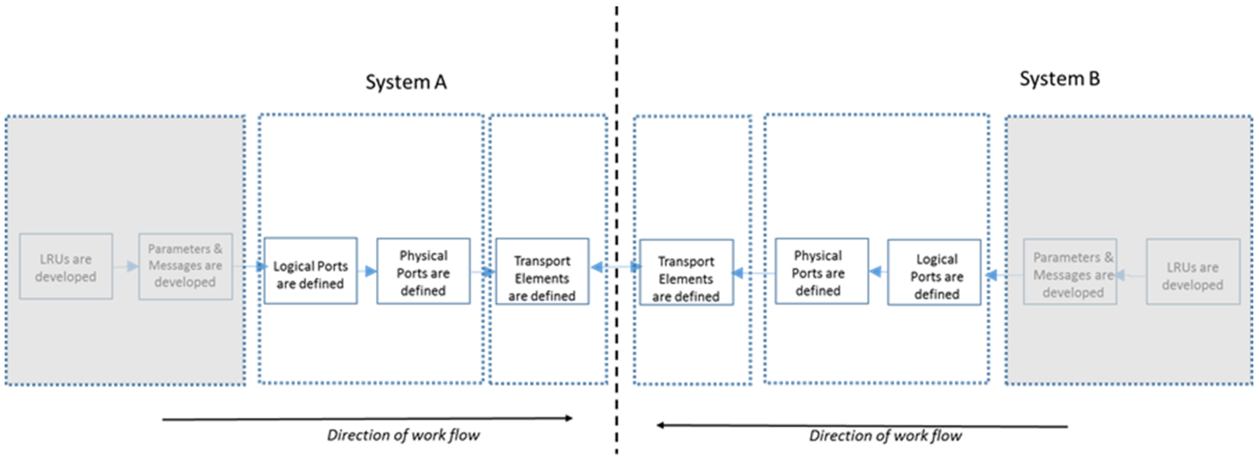

System A
System B
LRUs are developed
Parameters & Messages are developed
Logical Ports are defined
Physical Ports are defined
Transport Elements are defined
Transport Elements are defined
Physical Ports are defined
Logical Ports are defined
Parameters & Messages are developed
LRUs are developed
Direction of work flow
Direction of work flow

Figure 47. System architecture and system interfaces

The next incremental step, as shown in Figure 48, is the logical and physical port definition and development.

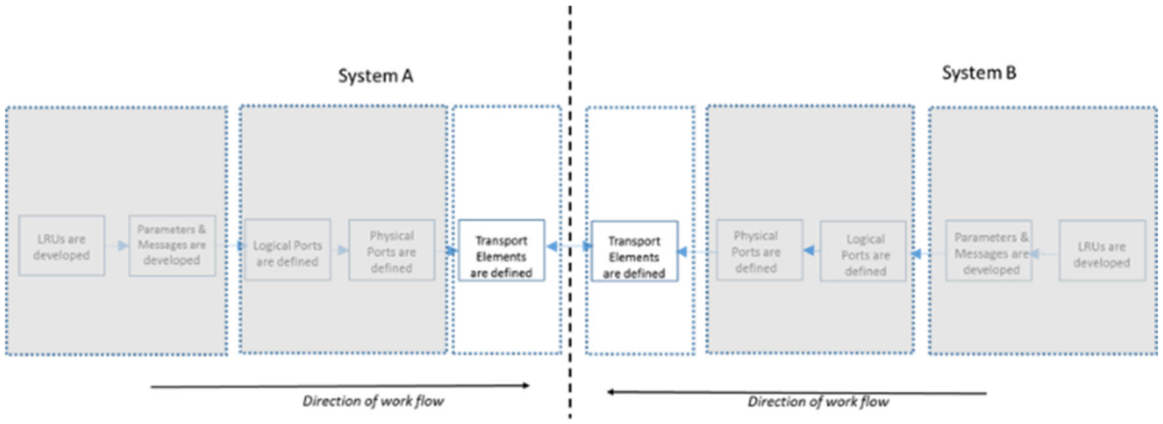

System A
System B
LRUs are developed
Parameters & Messages are developed
Logical Ports are defined
Physical Ports are defined
Transport Elements are defined
Transport Elements are defined
Physical Ports are defined
Logical Ports are defined
Parameters & Messages are developed
LRUs are developed
Direction of work flow
Direction of work flow

Figure 48. Logical and physical port definition and development

The logical ports should be defined (e.g., analog port, Controller Area Network [CAN] port). Some logical ports are only supported by certain physical ports. Physical ports are system and interface dependent. Examples include CAN Physical Port, Analog Physical Input Port, Analog Physical Bidirectional Port, and Analog Physical Output Port.

The next incremental step, as shown in Figure 49, is the definition of the transport elements (e.g., wiring).

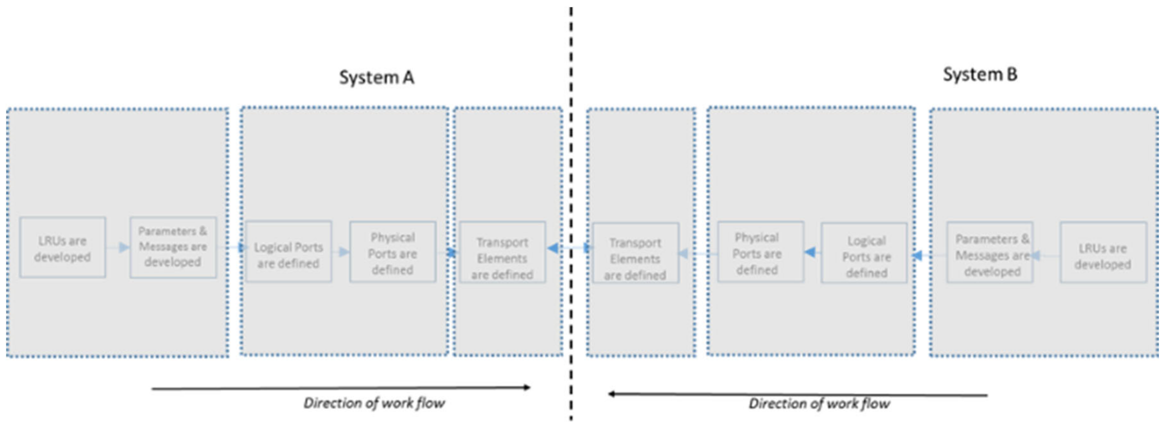

System A
System B
LRUs are developed
Parameters & Messages are developed
Logical Ports are defined
Physical Ports are defined
Transport Elements are defined
Transport Elements are defined
Physical Ports are defined
Logical Ports are defined
Parameters & Messages are developed
LRUs are developed
Direction of work flow
Direction of work flow

Figure 49. Transport elements defined

If this were a federated system, it would be sufficient to accomplish the following:

- Validate completeness, sufficiency, and accuracy of systems architecture.
- Validate that each system is providing what is required by other systems.
- Validate that each system is receiving what is needed (from the correct system).
- Validate that the parameters, logical ports, physical ports, and dataset and messages are complete and correct.

However, with DIMA architectures, additional integration activities need to occur. Section 4.2described the integration between federated systems, as shown in Figure 50.

- Integration of LRU 3 from System X with LRU 1 from System Y.
- Integration of LRU 2 from System X with LRU 2 from System Y.

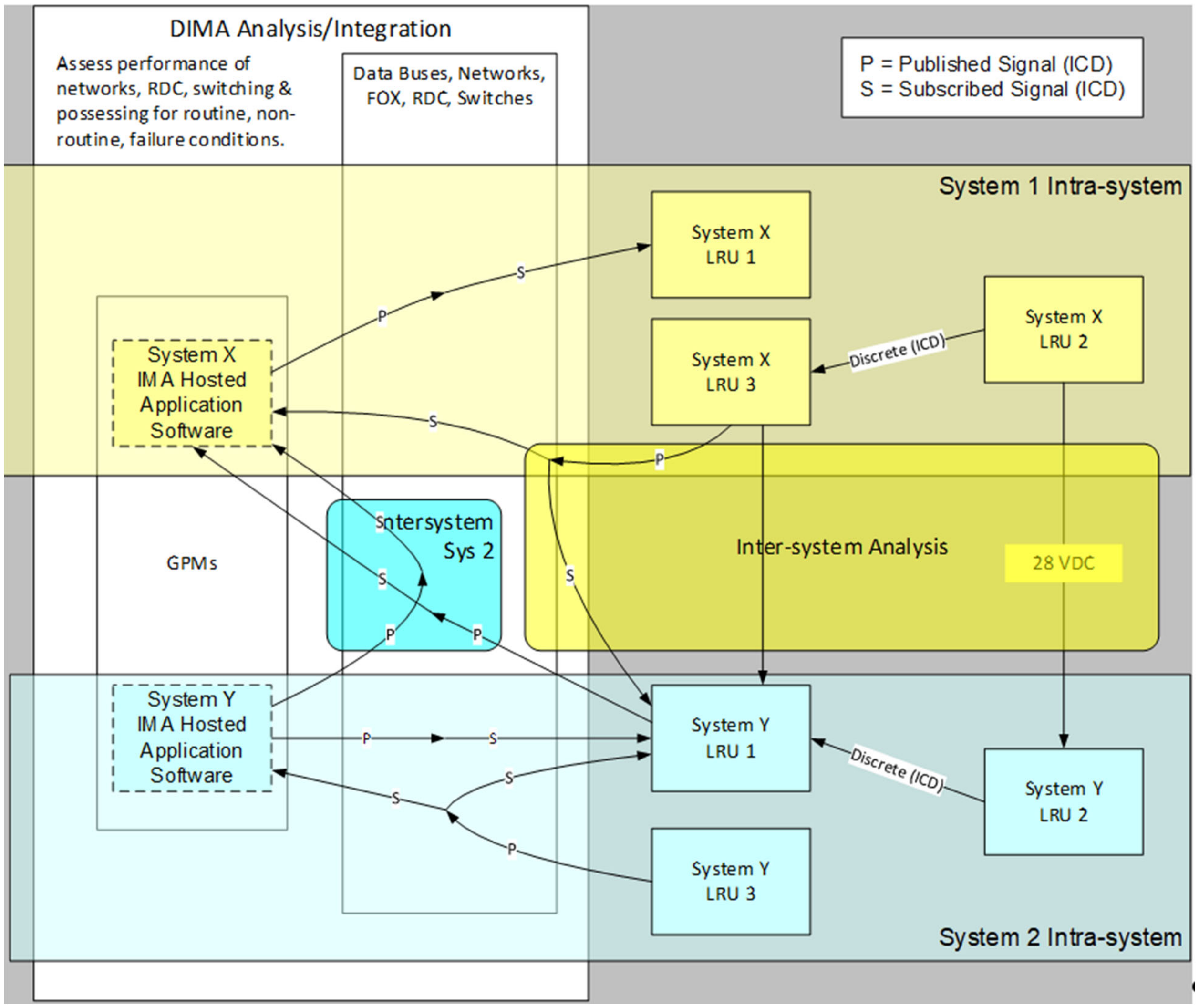

DIMA Analysis/Integration
Assess performance of networks, RDC, switching & possessing for routine, non-routine, failure conditions.
Data Buses, Networks, FOX, RDC, Switches
P = Published Signal (ICD)
S = Subscribed Signal (ICD)
System 1 Intra-system
System X LRU 1
System X LRU 2
System X LRU 3
System X IMA Hosted Application Software
Discrete (ICD)
GPMs
Intersystem Sys 2
Inter-system Analysis
28 VDC
System Y IMA Hosted Application Software
System Y LRU 1
System Y LRU 2
System Y LRU 3
Discrete (ICD)
System 2 Intra-system
P
S

Figure 50. DIMA integration framework

Because of the DIMA architecture, a system-to-platform analysis (including platform interfaces, redundancy, source preference, indicators, and separation) also needs to occur:

- System to Integrated Modular Avionics Platform Analysis.
  - Scenario 1: System hosted on a General Processing Module (GPM).
  - Scenario 2: Systems communicating directly to Common Data Network (CDN).
  - Scenario 3: System communicating across CDN via Remote Data Concentrator (RDC).
- Redundancy, Separation, and Control.
  - Failure Modes and Effects.

        - Scenario 1: System hosted on a GPM.
        - Scenario 2: System communicating directly to CDN.
        - Scenario 3: System communicating across CDN via RDC.
    - Additional Considerations
        - Minimum required resources to sustain functionality.
        - Maximum configuration analysis.
        - Minimum Equipment List (MEL) considerations.

In addition, the framework, as shown in Figure 51, can include the integration of the analysis and test activities.

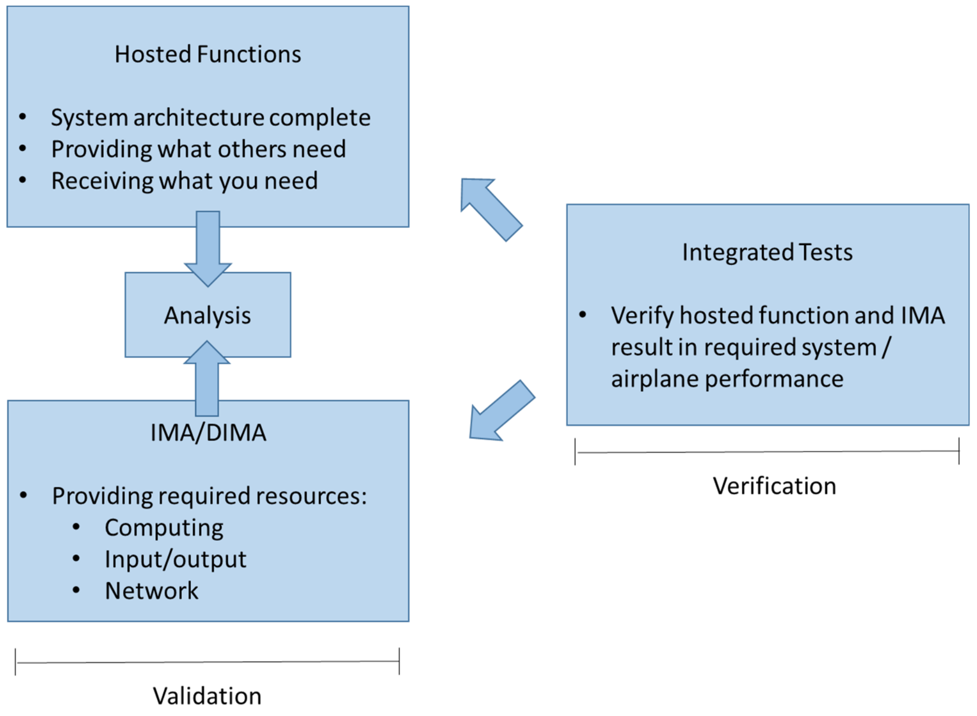

Hosted Functions
System architecture complete
Providing what others need
Receiving what you need
Analysis
IMA/DIMA
Providing required resources:
Computing
Input/output
Network
Validation
Integrated Tests
Verify hosted function and IMA result in required system / airplane performance
Verification

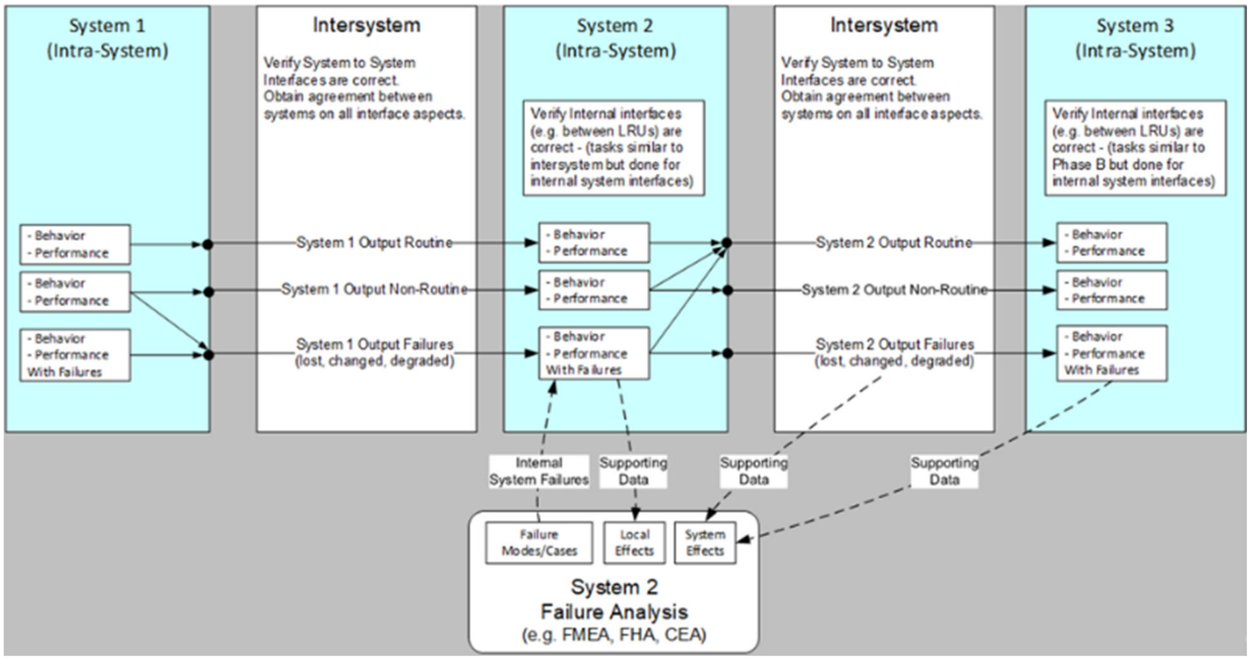
Figure 51. Hosted functions and IMA integration (V&V)

5.4 Potential gaps in existing V&V processes for highly integrated, distributed platform systems

Section 4.2 identified potential process shortcomings (particularly related to aircraft/system integration and modifications to aircraft or systems). The sections below identify other potential gaps related to development/design assurance processes.

5.4.1 Cross functional interfaces and safety

It is important to consider potential impacts of more highly integrated architectures on safety. For example, an intra-system analysis would need to contain a detailed analysis of behavior and performance with failures (internal failures or external inputs). The safety Failure Modes and Effects Analysis (FMEA) will identify the internal system failure modes and summarize the local, system and airplane effects. Figure 52 shows the interrelationships that can be considered between interface management and safety activities.



System 1 (Intra-System)
Intersystem
Verify System to System Interfaces are correct. Obtain agreement between systems on all interface aspects.
System 2 (Intra-System)
Verify Internal interfaces (e.g. between LRUs) are correct - (tasks similar to intersystem but done for internal system interfaces)
Intersystem
Verify System to System Interfaces are correct. Obtain agreement between systems on all interface aspects.
System 3 (Intra-System)
Verify Internal interfaces (e.g. between LRUs) are correct - (tasks similar to Phase B but done for internal system interfaces)
- Behavior - Performance
- Behavior - Performance
- Behavior - Performance With Failures
System 1 Output Routine
System 1 Output Non-Routine
System 1 Output Failures (lost, changed, degraded)
System 2 Output Routine
System 2 Output Non-Routine
System 2 Output Failures (lost, changed, degraded)
Internal System Failures
Supporting Data
Supporting Data
Supporting Data
Failure Modes/Cases
Local Effects
System Effects
System 2 Failure Analysis (e.g. FMEA, FHA, CEA)

Figure 52. Integration and safety activities

5.4.2 Cascading failure effects considerations

The to-be-released version of ARP 4761 Rev A contains Appendix O, Cascade Effects Analysis (CEA). It describes a qualitative, bottom-up analysis method that evaluates an initiating condition (e.g., a failure condition, failure mode, or combination of failure modes) and captures the total effect on the aircraft for that initiating condition.

Per the to-be-released version, CEA can be used to:

- Determine the effects of resource system failure conditions as part of the Airplane Functional Hazard Assessment (AFHA) or System Functional Hazard Assessment (SFHA).
- Determine the effects of resource system failure modes or combinations of resource system failure modes as part of the PASA.

- Determine the effects of shared or integrated item failure modes as part of the PSSA.
- Determine the effects of loss of function or malfunction of a shared or integrated component containing software or airborne electronic hardware as part of the IDAL assignment in the PSSA.
- Determine the effects of failure or damage scenarios identified as part of the PRA or ZSA.

The addition of Appendix O in ARP 4761A provides a method to help with airplane-level assessment of equipment failures, including:

- Primary failure effect on a system.
- Cascaded failure effects on other systems.
- Flight-deck effects.
- Crew actions.
- Validating each of the airplane-level functions can still be performed.
- Validating airplane-level hazard categorizations are acceptable, based on failure probability (25.1309 compliance).

While this is beneficial, based on experiences gained for conducting airplane-level cascading effects analyses on multiple development programs over 20 years, the area in the following section warrants emphasis for highly integrated, distributed systems architectures.

Number of Cascading Failure Effects

For highly integrated, distributed architectures, it is important to emphasize that the CEA should be evaluated until there are no further effects on system or airplane-level functions. The analyses should not stop at a preordained level of cascading effects (for example, stop after second-order effects).

As shown in Figure 53, it is also important to look for combination of effects that create additional cascading effects.

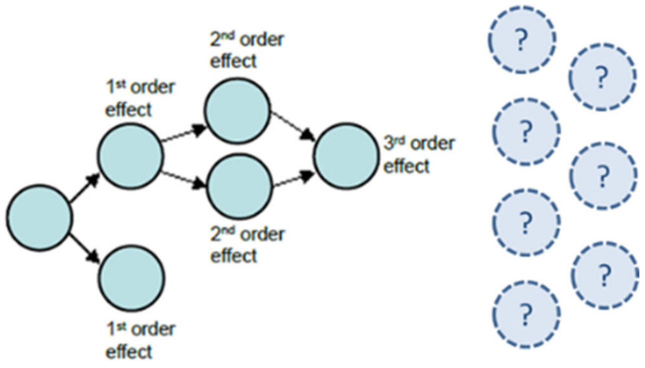


Figure 53. Cascading failure effects

5.4.3 Resource Assurance Analysis

It is important to ensure that the DALs are assigned correctly, which is more straightforward for federated systems architectures. For highly integrated, distributed systems architectures, it can be beneficial to conduct a bottom-up confirmation that the functions of one DAL are not adversely affected by functions of a lesser DAL, and where it does, that there is a documented rationale on why that approach is acceptable (as shown in Figure 54). This can help preclude late and unplanned surprises to ensure that the analysis and design considered the integrated effects of failures at the airplane level, not just at the system-by-system level.

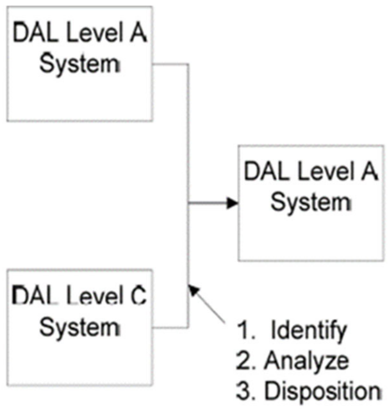


Figure 54. DAL validation for integrated architectures

For highly integrated, distributed systems, it would be beneficial to identify cases in which the DAL from a publishing system is less than the DAL of a subscribing system. Three key steps to assist with this analysis include:

- Identification and validation of the DAL levels.
- DAL analysis to validate that it supports the systems' required integrity and quality.
- Identification and disposition of discrepancies.

The disposition (justifying based on redundancy, architectural change, for example) should be captured. The Resource Assurance Analysis (RAA) should consider erroneous or degraded data or loss of function that could result from common modes to ensure that the assurance levels are commensurate with the DAL of the interface.

The RAA addresses assurance levels derived from hazard criticality, including:

- Systems DALs per SAE ARP 4754A [7].
- Software DALs per RTCA/DO-178B [3].
- Hardware (Application Specific Integrated Circuits [ASIC]/Programmable Logic Devices [PLD]) DALs per DO-254 [4].
- High Intensity Radiated Field (HIRF)/lightning protection levels.

For highly integrated, distributed architectures, this analysis helps validate the DAL levels.

5.4.4 DIMA considerations

For highly integrated, distributed systems architectures, the following should be considered during development and design assurance:

- Partial or complete failure of an IMA system, causing significant cascading failure effects on numerous aircraft functions.
- Uncommanded and inappropriate display reversions.
- Instances of simple failures (generator or engine loss), which could have a significant failure effect: disruption of power to a portion of the IMA architecture and loss of all displays on one side of the cockpit.
- Complete loss of electronic checklist (ECL) capability under certain failure scenarios.
- ECL not robust enough to deal with certain complex, multiple-system cascading failure scenarios.
- Generation of unnecessary checklists in the ECL system during cascading failure scenarios, which could add to crew workload; often, each unnecessary ECL had to be either individually worked or individually overridden.
- Failure of single elements of the electrical power distribution architecture potentially causing wholesale loss of sensor or system information and the removal of such information from the systems synoptic.

The time frame for the analyzed problem reports covered multiple years and included individual component testing, integrated laboratory integration, systems integration lab testing, airplane ground testing, and airplane flight testing. As expected for airplanes with increasing software (as shown in Figure 55), the majority of the problem reports were related to software (as opposed to hardware).

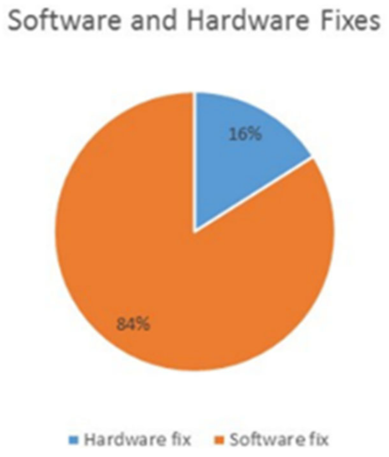


Figure 55. Breakdown of software and hardware fixes

Integration, as shown in Figure 56, occurs both horizontally and vertically and at different levels.

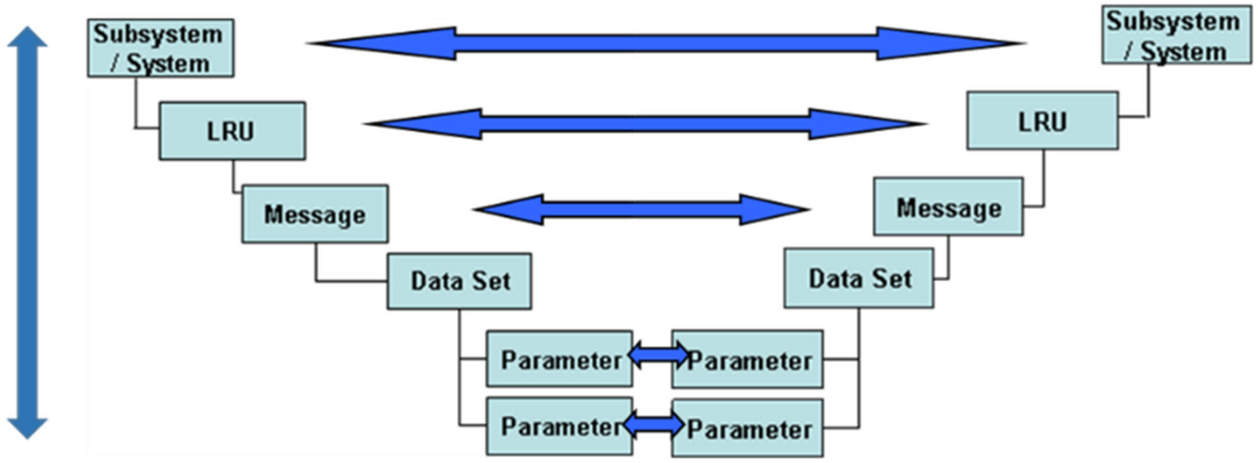


Figure 56. Multi-level horizontal and vertical integration

Figure 57 shows the results of an analysis of problem reports/design changes for the time frames when they generally occur. In the early phases of the design, there are changes to the number and types of equipment, as trade studies are conducted. As the design becomes progressively more detailed, the changes to the equipment become much less frequent. However, due the recursive, iterative nature of design, changes still occur to the lower levels design details (e.g., updating sampling port rates, adjusting transmit intervals from 50 ms to 45 ms).

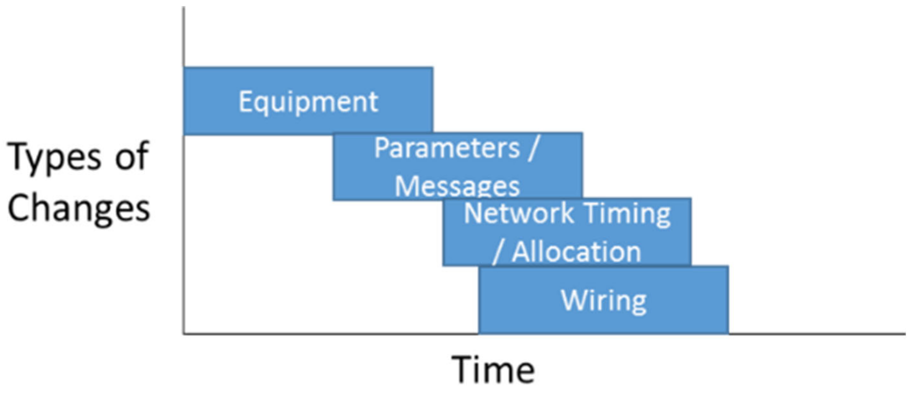


Figure 57. Types of changes over time

5.5 Virtualization, design of experiments, and distributed test

A universal goal is to minimize the number of problem reports that are discovered late in a program, which has an adverse effect on cost and schedule. This is particularly true for problem reports identified during flight testing. To fully verify and validate the development lifecycle, MBSE, MBSA, and MBD can be complemented by an early integration with T&E, and V&V activities.

This early integration requires T&E and SE staff to collaborate in developing verification and validation requirements, as well as defining test approaches that fully address complex systems and SoS in a manageable and efficient manner. This includes helping to ensure the system concepts, requirements, architectures, designs, and operations are valid, feasible, affordable, producible, and testable. Ultimately, this mitigates risks earlier in the development lifecycle, reduces costs, and shortens the time to operational readiness.

Methods, such as virtualization, provide the capability to improve requirements quality, expedite test and integration on the hardware, and enable verification through a virtual Systems Integration Laboratory (SIL).

These activities can help minimize the likelihood of errors first being discovered on the airplane during flight test. Table 9 categorizes the different types of problem reports that can be encountered during a flight test program.

Table 9. Analysis of flight test problem reports

Category	Sub-Category	Description
Possible to identify in lab	Lab/Simulation Fidelity	Lab and simulation configuration did not have sufficient fidelity to identify the problem.
	Test Plan	Test case or specific conditions were not in the lab test plan.
	Test Procedure	Test procedure was incorrect or did not provide enough information on expected results.
	Simulation error	Error in simulation requirements or implementation.
	Existing Lab PR	Problem Report (PR) was identified in the lab, but squawked on airplane (no paperwork or test crew wasn't aware).
Airplane testing specific		Specific to the airplane test configuration, problem specific to airplane test setup issue.
Per Design (i.e., not a problem)		Test personnel familiarity with design, understanding of requirements, for example.
Root cause under investigation		PR is still under investigation.

It is important to emphasize that the overwhelming majority of the PRs are identified and fixed during lab testing (prior to airplane testing). It is equally important to emphasize that this is not a safety issue; it is more a matter of identifying and fixing problems earlier in the lifecycle. Problem reports should be evaluated for their potential impact to safety and certification.

The goal is to be able, to the maximum extent possible, identify and fix problems prior to flight test. If the problem is caught in requirements instead of lab test, it is a 10:1 savings. If a problem is caught in lab test instead of flight test, it is a 10:1 savings. Having complete and correct requirements can mitigate two orders of magnitude of cost savings. It is these potential improvements that have generated interest in virtualization and other model-based activities.

5.5.1 Virtualization

Virtualization technologies allow for high-fidelity representations of aircraft systems to be created early in the product development lifecycle. Virtualization is the creation of a software emulation of hardware/equipment. This environment provides output responses that match the real signal paths and behaviors. Virtualization technologies execute the unmodified target software executable in a virtual machine (VM), and this approach can even be accomplished with processor emulation in the VM to accommodate different processor types.

As shown in Figure 58, advancements are being made to increase the fidelity of aircraft system simulations, using emulated hardware in VMs that match actual avionics. The higher fidelity VMs create a much smaller fidelity gap between the virtual world and the physical world. This high-fidelity virtual world enables problems to be caught early and extracted when the cost to extract these problems are often orders of magnitude that are less expensive to remove.

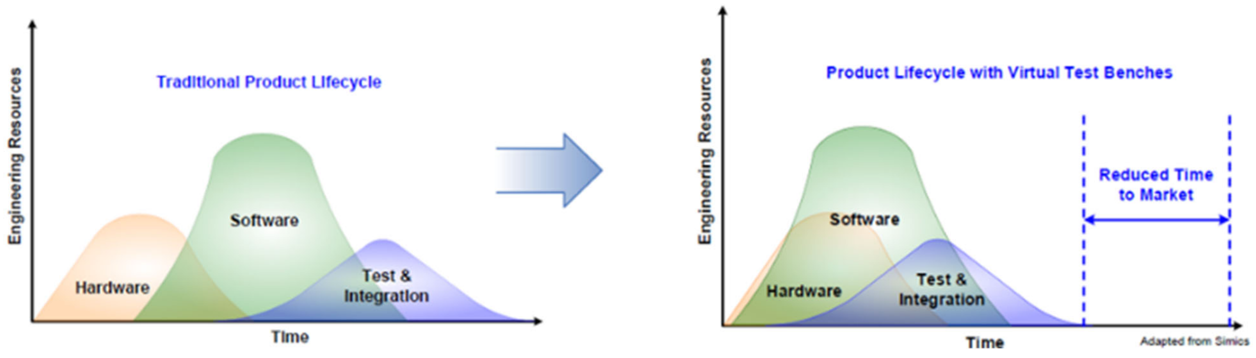


Figure 58. Early identification with virtualization

As presented in Figure 59, virtualization is an emerging technology that provides capabilities to facilitate early evaluation of requirements at both component (line-replaceable module [LRM]/line-replaceable unit [LRU]) and integration levels. Depending on the fidelity or the virtualization construct, the early evaluation could be used to facilitate validation of requirement content, to be followed by verification of actual requirements.

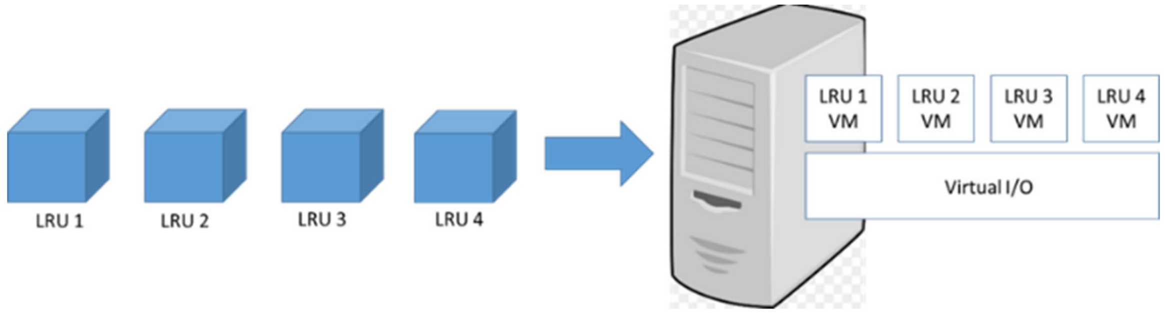


Figure 59. LRU virtualization

5.5.2 Design of experiments

Another key methodology in managing verification of complex systems is Design of Experiments (DOE). DOE can potentially be the single largest Return on Investment (ROI) factor on a program. DOE methods ensure the appropriate amount of testing is being conducted. DOE is a statistical method to determine the relationship between factors affecting a process and the output of that process. The use of DOE can yield test screening and samplings at a confidence level of 99% for the Mean and 95% for the standard deviation. DOE can mathematically confirm the interactions and independence of input factors. More importantly, DOE can help ensure repeatability and reproducibility in complex SoS designs.

5.5.3 Distributed/interoperability test

The Distributed Test capability enables the test of interoperability attributes and performance characteristics of complex, networked SoS. As the complexity of open architecture, networked systems increases, the importance of developing and validating these systems increases, as the system elements are developed and tested in disparate locations with varying requirements, program milestones, and system maturity.

Distributed Test is a tool that can manage this increased complexity in a cost-effective manner by accelerating collaboration and decision making when SMEs are not co-located. Distributed Test can be used to connect assets, thus avoiding the need to duplicate these assets at a single site. Distributed Test can be used to mitigate integration and safety issues for process execution problems.

Significant benefits can be achieved when T&E and SE jointly participate in developing verification and validation requirements, particularly in defining test approaches that are appropriate to integration, verification, and validation. This includes helping to ensure the system concepts, requirements, architectures, designs, and operations are valid, feasible, affordable, producible, and testable. Ultimately, this helps programs identify and mitigate risks earlier in the product lifecycle, decrease risks and costs, and shorten the time to operational readiness.

Interoperability testing focuses on the importance of multiple elements (components, subsystems, platforms, and/or SoS) operating an integrated fashion, hereto referred to as "interoperability." Test planning should include interoperability considerations, for example, possible development cycles/spirals, depending on program or project size and complexity.

At the top level, interoperability testing ensures the ability of systems to operate together and/or exchange information in the same environment, without disrupting any participant's ability to perform its intended, independent function.

Interoperability test is the ability of a system of interest (SoS, systems, platforms, subsystems, components, and environments [e.g., units, forces, atmospheric conditions]) to interact with all aspects of the test system (e.g., physical interfaces, environmental conditions, command, people [e.g., semantics, understanding], power, and data [e.g., processes, security, data sharing]) and accept the same from the other components of the systems of interest to enable them to operate effectively together. Interoperability test includes both the technical exchange of information and the end-to-end operational effectiveness of that exchanged information, as required for regulatory and operational requirements. Interoperability testing should consider the level of standards compliance of existing implementations, as well as new systems to be incorporated. To achieve maximum benefit, interoperability testing can be performed in the most operationally realistic environment possible, and can determine if the system of interest conforms to applicable standards and ensures that data collected are adequate for evaluating interoperability issues.

Leveraging Live, Virtual, Constructive (LVC) also enables the injection of mission-based scenarios and relevant faults into the system to evaluate the resultant system performance and emergent behaviors. The earlier the LVC test is utilized, the earlier the refinement of the concept of operations (CONOPS), concepts, requirements and design, and integration of the system can occur. As the system and the LVC environments mature, the potential risks, issues, and defects from future verification and validation test events will have been wrung out long before the real system is actually integrated and tested.

It is recommended that LVC testing be investigated as a potential mitigation of integration and safety issues for process execution problems.

6 Issues, challenges, and gaps in current standards/guidance

This research assesses the potential issues, challenges, and gaps in the current standards and guidance. Table 10 describes the approach to address these research objectives.

Table 10. Potential issues, challenges, and gaps

Research Objectives	**Detailed Work Description**
Identify the issues, challenges, and gaps in the current standards/guidance	Identify potential safety issues related to verification and validation during development and change impact analysis of highly integrated complex digital systems. Identify gaps in the design/development assurance processes, used to verify and validate integrated complex digital systems, and identify any potential failures and failures effects resulting from such complex systems integration for highly integrated, distributed platform systems. Ensure common understanding of system design model definition. Model used to express architectural, behavioral, performance, and interface requirements of the system being modeled. System design model can be analyzed to predict and understand the modeled system's capabilities and operational quality attributes (e.g., performance, reliability, safety, security).

Research Objectives	Detailed Work Description
	System design model is typically technology independent, where the requirements expressed by the model may be allocated to either hardware and/or software. Identify MBD processes, where executable models (system design model) are developed at the system level for behavioral evaluation, and requirements for software process are defined. System development process. Requirements development. Requirements validation. System-level verification. Identify MBD processes to integrate between system- and software-level models. Identify MBD processes to integrate between system and airborne electronic hardware models. Identify how to conduct model coverage analysis. Identify integration touchpoints of MBD with V&V processes.

6.1 Potential safety issues and gaps in the design/development assurance processes

In order to assess potential safety issues and gaps in industry guidance, summarizing prior sections of this report is helpful:

- Section 4.2 identified potential industry guidance shortcomings (particularly related to aircraft/system integration and modifications to aircraft or systems).
- Section 4.3 identified potential industry guidance gaps related to safety.
- Section 5.2 discussed potential industry guidance gaps related to existing V&V processes for highly integrated, distributed platform systems.

In addition to the content covered in the sections referenced above, it is important to understand what needs to be modeled, the required level of fidelity of said models, and how the different models will be integrated. This is critical to being able to use the models to help with systems architecture analyses, safety analyses, systems integration, and change impact analysis.

All models are wrong. Some are useful.—George Box

Models, by definition, are an abstraction of reality; they are a simplified version of a concept, relationship, system, or structure. A model can be graphical, mathematical, or a physical representation. Some of the objectives of modeling include:

- Ensure there is complete and correct information across an enterprise value stream.
- Increase ability and efficiency in conducting change impact analysis.
- Aid in regression analysis.
- Minimize the recreation of data in multiple "authoritative" sources.
- Increase the level of fidelity of information.
- Aid in communication and interpretation of design intent.
- Improve ability to analyze a system design earlier in the development lifecycle (e.g., prior to detailed design release, software coding, build).
- Provide verification and validation capabilities beyond what can be practically and safely achieved via testing of flight vehicles.

As stated above, there are significant potential benefits to modeling. However, it is important to clearly understand the required scope and the required level of modeling fidelity. It is a common (and correct) saying that the results of using model are only as good as the model itself. This leads to the standard cliché "garbage in, garbage out."

The models will be generated in support of multiple design disciplines (e.g., avionics, flight controls, stability and control, electrical, hydraulics, flight deck). In addition, it is common in the aerospace industry for a given aircraft to have multiple suppliers. This highlights the importance of having models that can be integrated.

- Will the models sufficiently represent the actual design in order to make a safety assessment?
- Will the models sufficiently represent the interfaces to support integration analyses?
- Will the models sufficiently represent the behavior of the as-designed/as-built flight vehicle, in order to analyze performance during nominal and failure conditions?
- Will the models enable different design disciplines to integrate with each other (as required)?
- Will the models enable the integration of different suppliers' models?

- Will the models identify when lower level models violate high-level model constraints or assumptions?
- Will the models support the recursive, iterative nature of design (as further details are developed)?

The following diagrams will provide a simplistic example of some model considerations, particularly with regards to the required level of fidelity. The concept of data models to support integration and safety assessments is discussed in further depth in section 6.2.

Figure 60 shows a simplified model, which can occur early in the architecture development. At a given point, this might be sufficient. The model shows that System A provides an output to System B. This may be the result of an early trade study. For example, the brake system, which may have been traditionally used hydraulics actuation, may now be electrically actuated. This provides a very high level of integration. However, it is not possible to conduct a safety assessment with regards to this model. For example, the level of redundancy is not modeled.

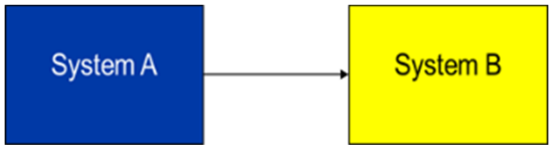


Figure 60. Early integration

In this example, architecture decisions are made to have:

- Left, center, and right components for System A.
- Left and right components of System B.

However, this level of modeling, as shown in Figure 61, would not support the ability to do cross-functional integration or safety analyses.

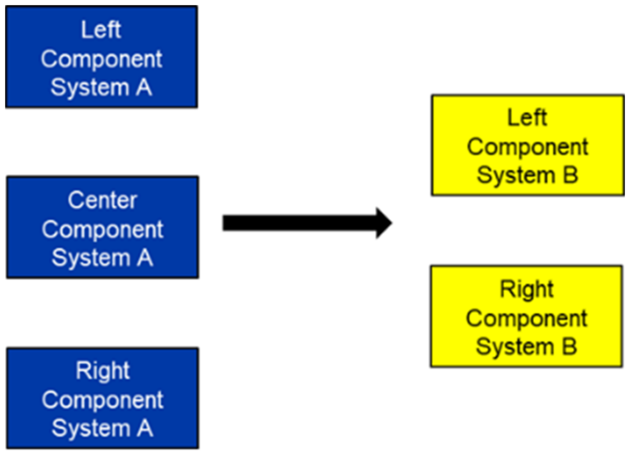


Figure 61. Architecture redundancy

Figure 62 shows one potential way this architecture could behave. The left yellow component receives its required data from both the left and center blue components. The right yellow component receives its required data from center and right blue components.

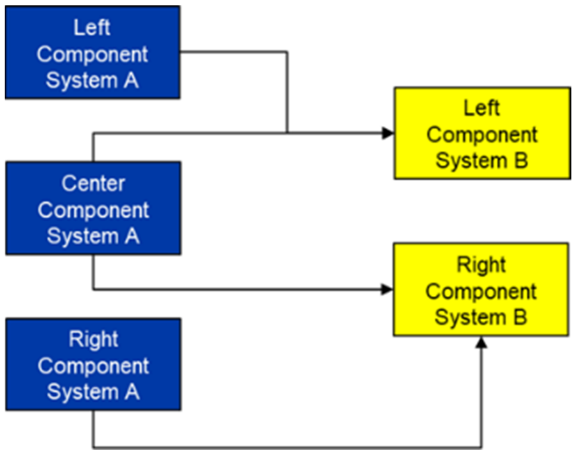


Figure 62. Architecture #1

An alternative architecture is shown in Figure 63. The left yellow component receives its required data from the left, center, and right blue components. The right yellow component receives its required data from the left, center, and right blue components.

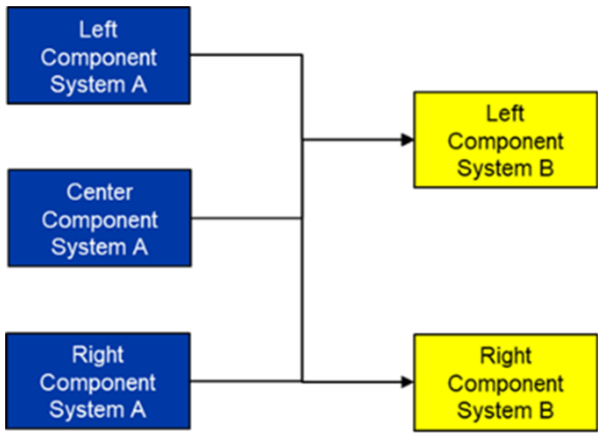


Figure 63. Architecture #2

It is important to ensure that the format of the data being passed between different systems is complete, correct, and consistent. Commercial aircraft use protocols (e.g., ARINC 429, ARINC 664) to specify the format of the data. Figure 64 provides a simple example: System 1 is publishing Signal 1; System 2 is subscribing to the Signal 1 that System 1 is publishing. To make this exchange of data possible, an interface needs to be both established and controlled between System 1 and System 2. This interface will include:

- Transmission path between the two signals.
- Consistent definition for the signal.

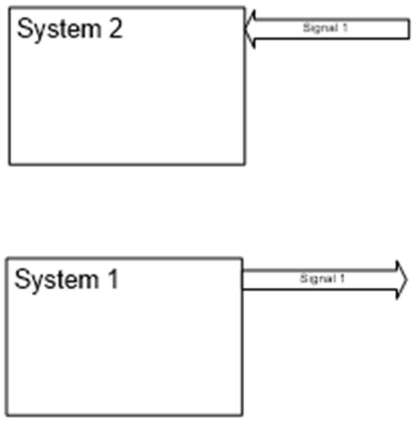


Figure 64. Interface protocols

The purpose of this simple example was to highlight how the required functionality can be achieved. It is important to clearly establish the required level of fidelity. To answer this, it is first important to understand why the model is being built. What purpose does it serve? How will it be used? Who will use it? How will the data be captured and managed? Do different models need to be integrated? The subsequent section goes into this in further detail, demonstrating how a single data model can be used to support different types of systems architectures.

6.2 Common understanding of system design model definition

It is important to understand what needs to be modeled and to have common understanding of the system design model definition. This is important because an airplane, similar to any other design, is a compromise of many competing objectives; it requires the knowledge and experiences of multiple design disciplines. (It should be noted that safety is the most important part of the design.)

Different design disciplines need to work together to determine the optimal aircraft design. If this does not occur, it is easy to optimize a particular aspect of the design, but to overall sub-optimize the design solution at the airplane level.

One of the benefits of modeling is to provide the opportunity to identify if a system is being sub-optimized. In order for this to occur, it is important to have a common framework so that the different models can be integrated. This is reflected in the principle of concordance, which can be defined as the ability to represent a single entity such that data in one view (or level of abstraction) matches the data in another view (or level of abstraction), when talking about the same thing.

Section 4.2 described the concepts of vertical, horizontal, and diagonal integration (shown in Figure 65).

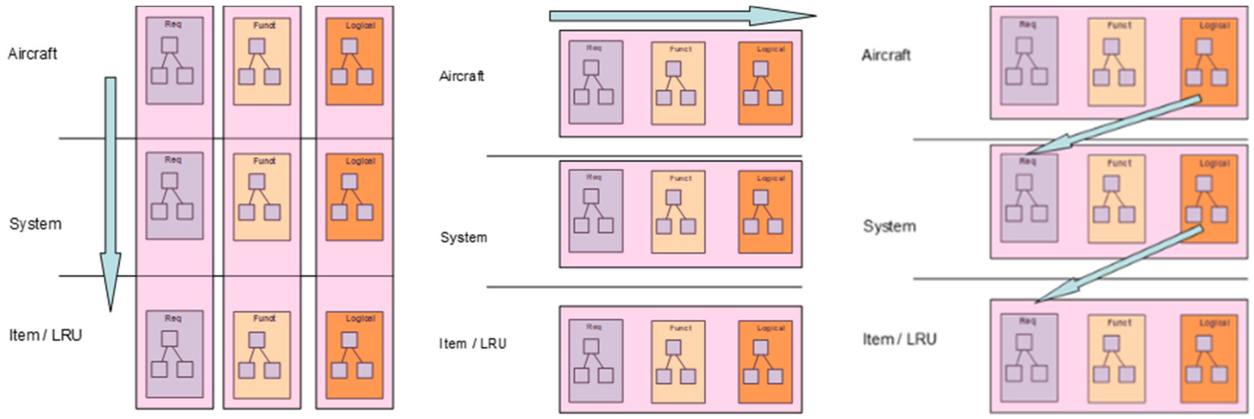


Figure 65. Vertical, horizontal, and diagonal integration

Vertical integration ensures the traceability and acceptability, as one moves from one abstraction level (airplane, system, item, LRU) within a given hierarchy (requirements, functional, logical, and physical). Horizontal integration ensures that there is an integrated product architecture at each abstraction level. Diagonal integration ensures the allocation/traceability from one abstraction level to another (e.g., from system to item/LRU). It helps ensure that the derived requirements from design decisions made at the higher abstraction level are identified and captured. The data model should support vertical, horizontal and diagonal integration.

6.2.1 Data model

To support the principle of concordance, it is important for the MBSE and MBSA activities to have a common data model. The data model captures the data objects, attributes, and relationships that need to be captured, as process and task steps are executed.

The following sections contain examples of data models for the systems architecture model, as described in Figure 66. They are meant for illustrative purposes only; they are not meant to imply that they are the only acceptable data models. It is also not meant to imply that these would be complete data models for a large, complex, highly integrated system.

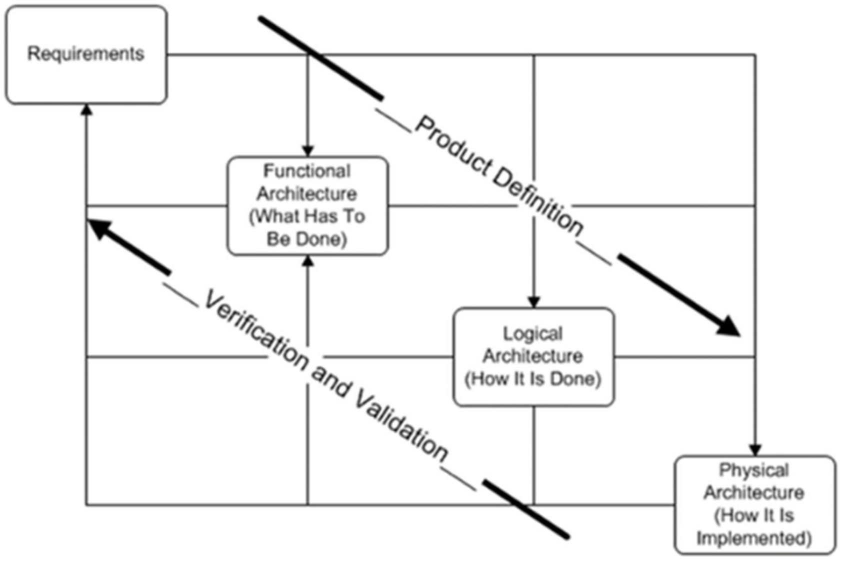

Requirements
Functional Architecture (What Has To Be Done)
Logical Architecture (How It Is Done)
Physical Architecture (How It Is Implemented)
Product Definition
Verification and Validation

Figure 66. Systems architecture model

Figure 67 provides an overview of UML diagrams, which consist of objects attributes and relationships. Figure 68 provides an overview on the types of relationships.

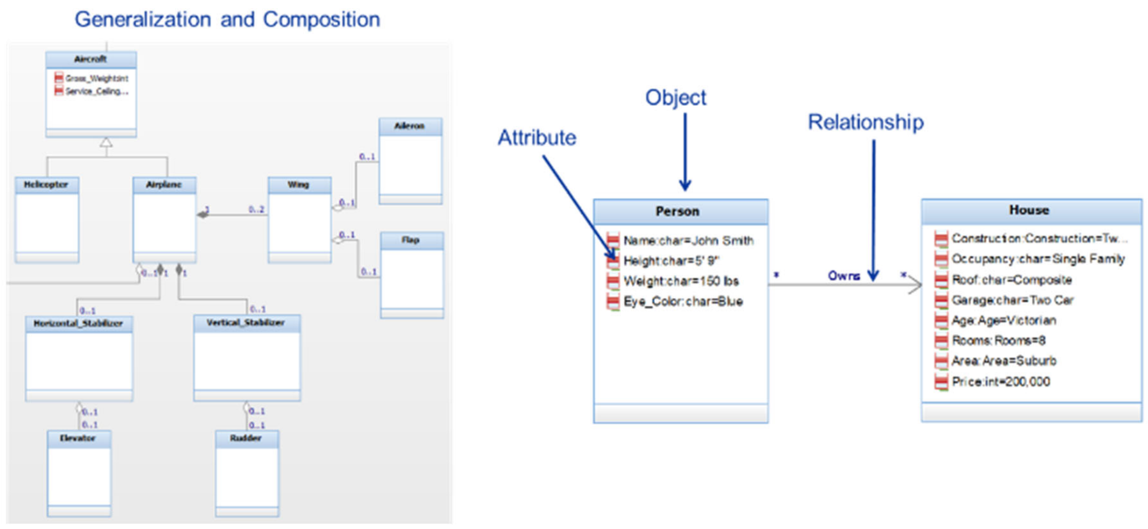


Figure 67. UML 2.0 Class diagrams

Relationship Icon	Relationship Name	Relationship Description
↑	Directed Association	Establishes a relationship between two objects and the reasons for the relationship. A directed association indicates that control flows from one object to another.
◇	Aggregation	A special type of association that organizes objects into an assembly. If the assembly is destroyed, the component objects still exist.
◆	Composition	A special type of aggregation where, if the assembly is destroyed, the component objects are also destroyed.
△	Generalization (Subtypes)	A relationship in which the definition of one model element is based on that of another model element, and is similar to, yet different from the referenced model element.

Figure 68. UML 2.0 Relationship icons

The subsequent sections will describe the different parts of the systems architecture model in UML.

6.2.1.1 Requirements architecture data model

The requirements architecture data model (as seen in Figure 69) shows that parent/child relationships can be created. The cardinality shows how many of each object type can be related. In this data model, there is a one-to-many relationship (i.e., every requirement can have many children requirements). This same process can be repeated at multiple levels to create a requirements hierarchy.

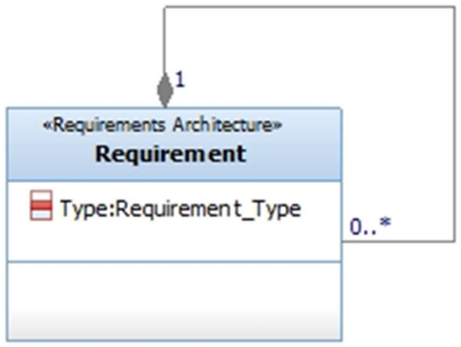


Figure 69. Requirements architecture data model

This requirements data model enables the creation of the requirements hierarchy. The requirements architecture, as shown in Figure 70, is a set of decomposed "Requirement" objects. Higher-level requirements can be decomposed into more specific lower-level requirements. The requirements can have subtypes, such as functional, performance, or design.

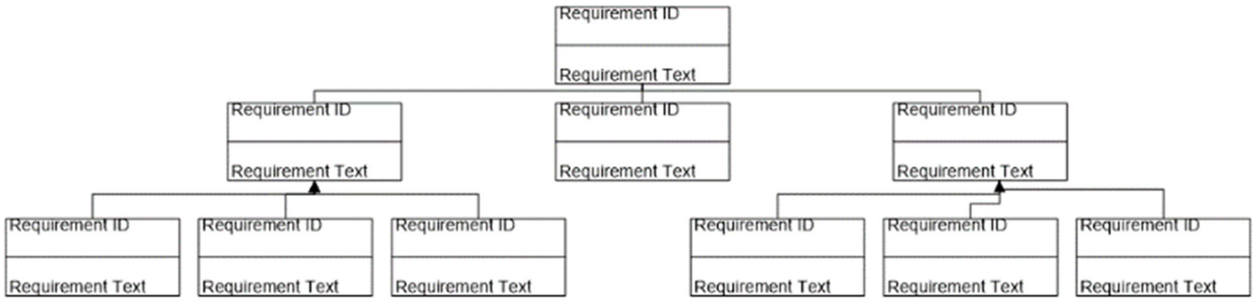


Figure 70. Requirements hierarchy

6.2.1.2 *Functional architecture data model*

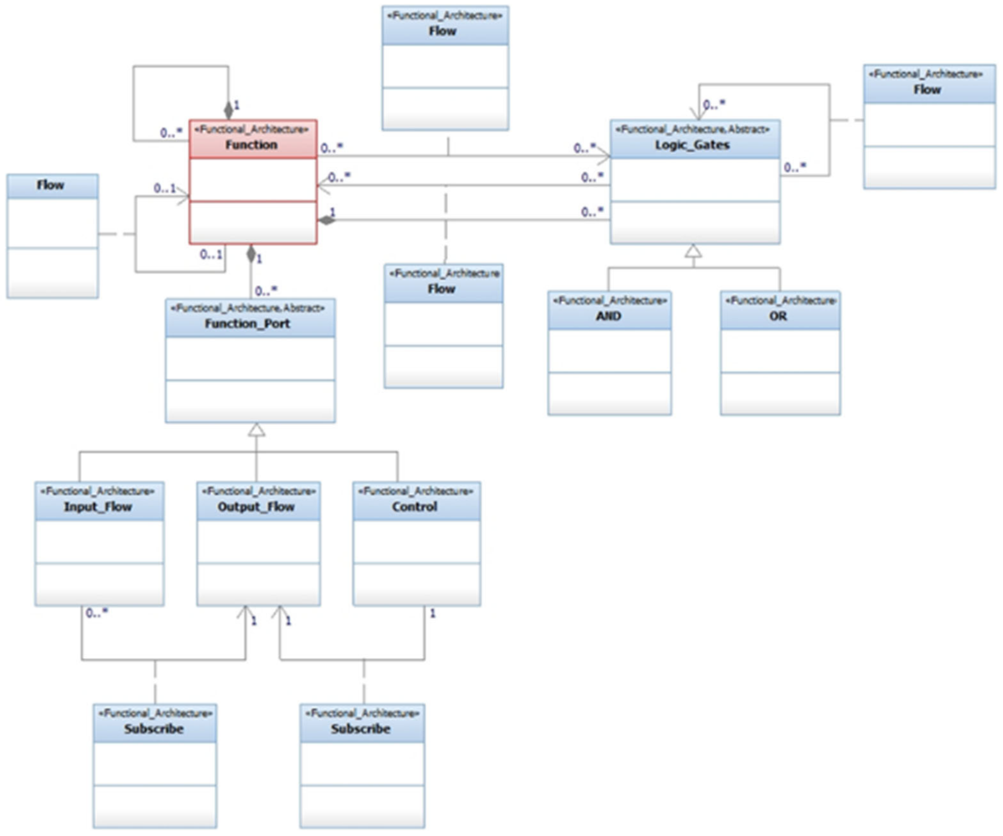
The functional architecture comprises a set of decomposed "Function" objects. Higher-level functions are decomposed into more specific lower-level functions.

The main elements of the functional architecture data model are shown in Figure 71, which are useful data categories. Objects can also have attributes, which define and describe an item in the data category. The functional architecture data model can include the following objects:

- Function.
- Function port.
- Functional data flows (input and output).

- Controls.
- Logic gates.

A function converts inputs to outputs. The input and output objects are children of the function object. Inputs comprise input flows and controls, and outputs comprise output flows. Input flows are resources needed by the function to execute. Controls are constraints on the execution of the function. Output flows are products of the function following execution.



«Functional_Architecture» Flow
«Functional_Architecture» Function
«Functional_Architecture.Abstract» Logic_Gates
«Functional_Architecture» Flow
Flow
«Functional_Architecture» Flow
«Functional_Architecture.Abstract» Function_Port
«Functional_Architecture» AND
«Functional_Architecture» OR
«Functional_Architecture» Input_Flow
«Functional_Architecture» Output_Flow
«Functional_Architecture» Control
«Functional_Architecture» Subscribe
«Functional_Architecture» Subscribe
0..*
1
0..1
0..*

Figure 71. Functional architecture data model

As shown in Figure 72, the connection operations between functions can be accomplished by creating data relationships (links) between output function ports and input function ports. The output function port is the source of the link and the input function port is the destination of the link. This functionality enables the creation of data flow diagrams.

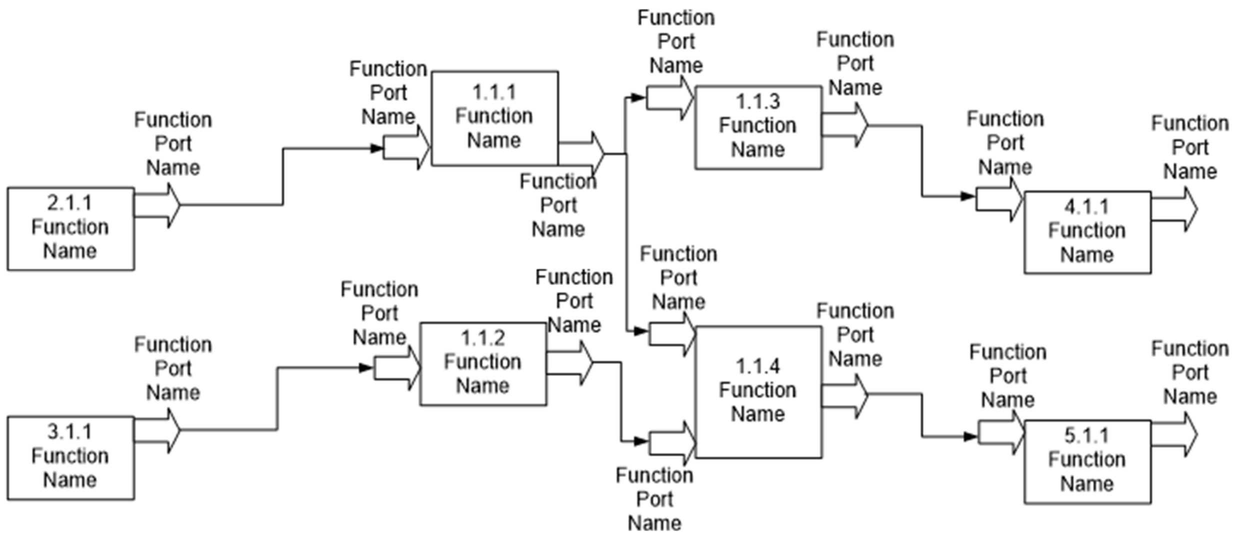


Figure 72. Sample data flow diagram

Logic gates and flow association can be used to establish sequence and execution logic between functions. Figure 73 also shows subtypes (generalization) associated with logic gates (for example, “AND” and “OR” generalize to logic gates). This functionality enables the creation of functional flow block diagrams (as shown in Figure 73).

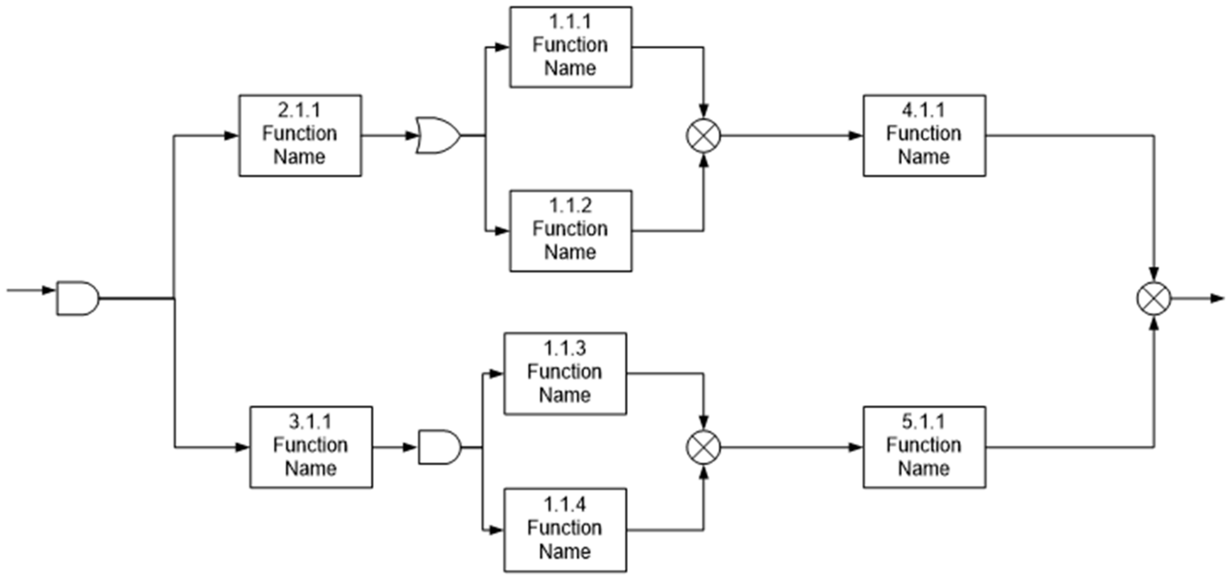


Figure 73. Sample functional flow block diagram

This functional data model enables the creation of the following:

- Functional hierarchy (parent/child relationships).
- Functional data flow diagrams (inputs and outputs for functions).
- Functional flow block diagrams (sequence).

6.2.1.3 *Logical architecture data model*

As shown in Figure 74, the logical architecture can consist of a system definition object, with subtypes of hardware and software. The data model contains a host association, which can be used to integrate hardware and software (i.e., software hosted on hardware). This data model enables the creation of a DIMA, in which, for example, an Avionics software package (such as Flight Management System) is hosted on hardware (such as a general processing module). The general process module can host multiple software applications.

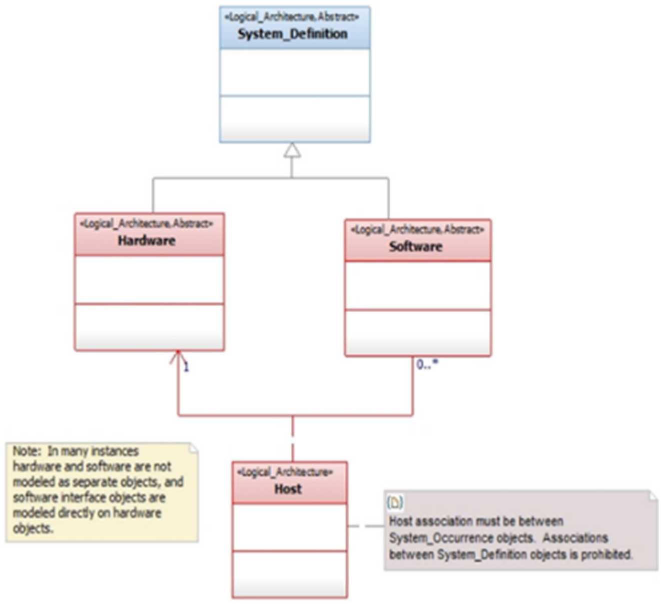

«Logical_Architecture.Abstract»
System_Definition
«Logical_Architecture.Abstract»
Hardware
«Logical_Architecture.Abstract»
Software
1
0..*
«Logical_Architecture»
Host
Note: In many instances hardware and software are not modeled as separate objects, and software interface objects are modeled directly on hardware objects.
Host association must be between System_Occurrence objects. Associations between System_Definition objects is prohibited.

Figure 74. Logical architecture data model - subtypes

One of the key reasons for modeling a system is to understand the intra-system and intersystem interfaces. This is critical to helping validate that there will be no anomalous behavior in both nominal and failure conditions.

Figure 75 shows a data model that can be used to create ports. The logical architecture data model has three major system port classes:

- Physical port.
- Logical port.
- Interface.

The physical port is used to model connections on hardware. The logical port is an interface point on a logical element; it is used to model signal transmission ports on software. Interfaces are used to model the signals transmitted on physical ports and logical ports.

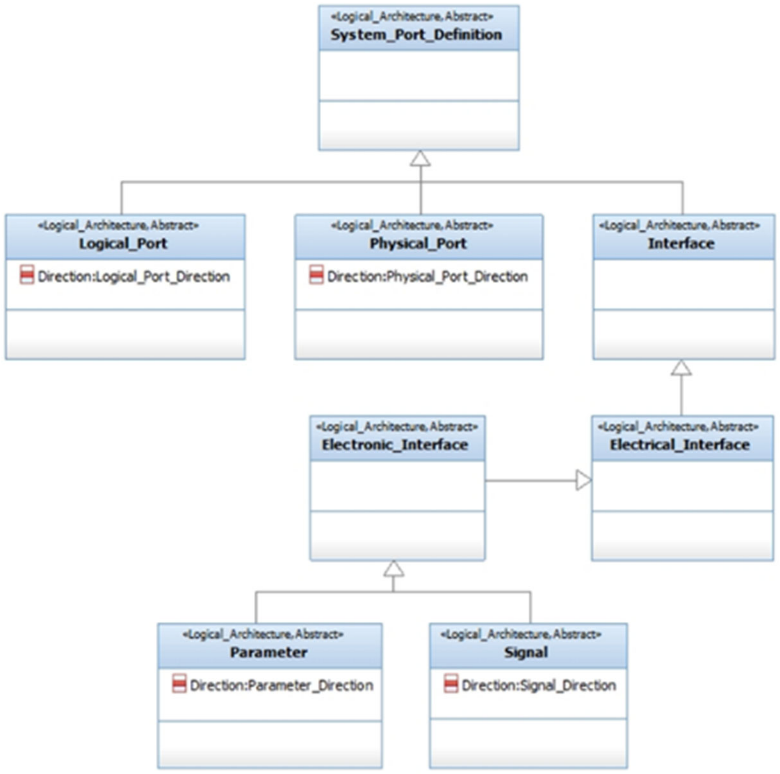

«Logical_Architecture.Abstract»
System_Port_Definition
«Logical_Architecture.Abstract»
Logical_Port
Direction:Logical_Port_Direction
«Logical_Architecture.Abstract»
Physical_Port
Direction:Physical_Port_Direction
«Logical_Architecture.Abstract»
Interface
«Logical_Architecture.Abstract»
Electronic_Interface
«Logical_Architecture.Abstract»
Electrical_Interface
«Logical_Architecture.Abstract»
Parameter
Direction:Parameter_Direction
«Logical_Architecture.Abstract»
Signal
Direction:Signal_Direction

Figure 75. Logical data model: system port subtypes

It is typical for a commercial airplane to use multiple ARINC protocols. The data model needs to support the creation of different signal formats. As an example, Table 11 describes the difference between ARINC 429, 629, and 664 signals.

Table 11. Signal characteristics of different ARINC protocols

ARINC Protocol	**Characteristic**
ARINC 429	Signals are single 32-bit data words. Signals transmitted from broadcaster to all receivers.
ARINC 629	Signals comprise up to 265 20-bit data words. Messages comprise up to 31 ARINC 629 signals. Signals broadcast to every LRU connected to the bus. Bus schedules are required to avoid signal clashes.
ARINC 664	Signals comprise multiple 32-bit data words, up to 1472 data bytes. ARINC 664 signals are not broadcast over entire network; message traffic is directed to subscribers through routers (switches).

Figure 76 represents a data model that can be used to support the creation of different signal formats.

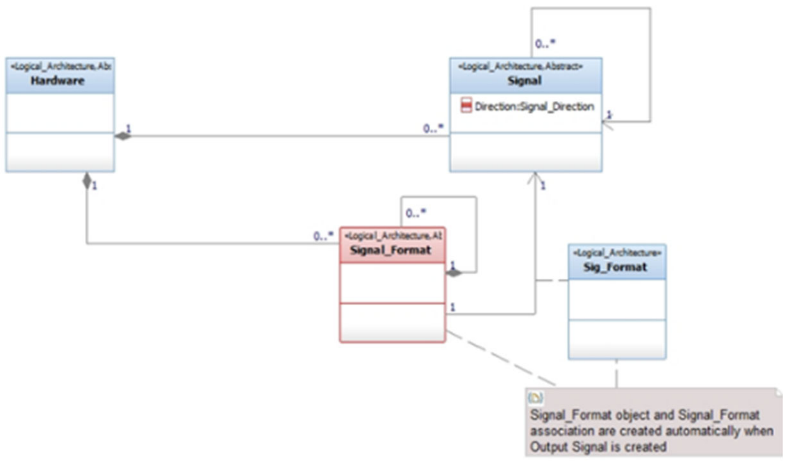

«Logical_Architecture.Ab
Hardware
«Logical_Architecture.Abstract»
Signal
Direction:Signal_Direction
0..*
1
0..*
1
1
1
«Logical_Architecture.Al
Signal_Format
0..*
0..*
1
1
«Logical_Architecture»
Sig_Format
Signal_Format object and Signal_Format association are created automatically when Output Signal is created

Figure 76. Logical architecture data model – signal format

Figure 77 shows a data model that can be used to support software and hardware integration, enabling software interfaces to be modeled directly on hardware. It should be noted that this data model can work for both DIMA (with hosted software) and more traditional architectures (with software in a standalone LRU component).

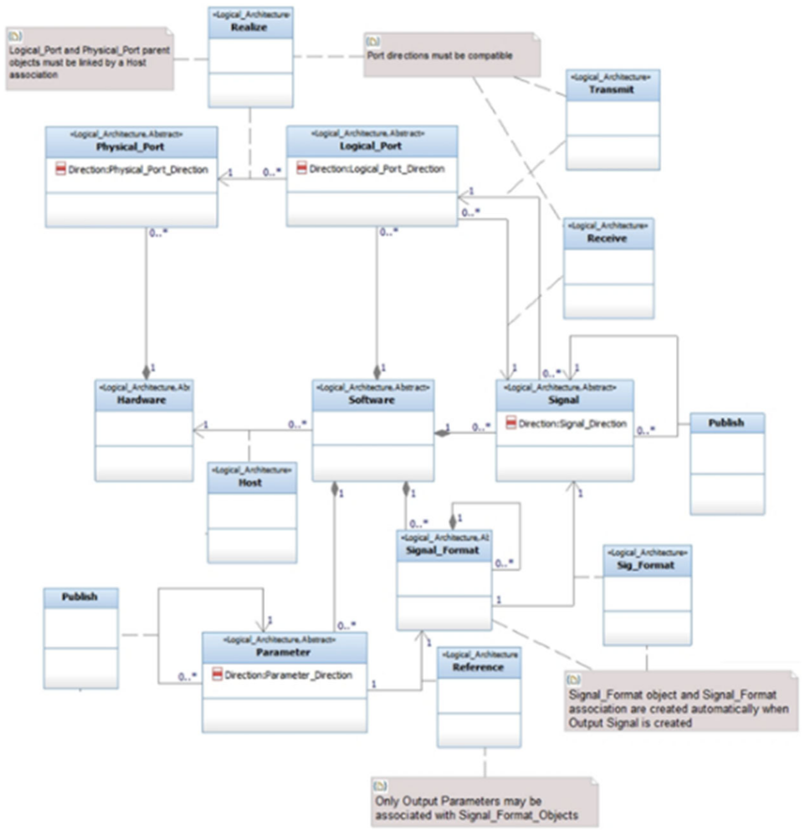

Logical_Port and Physical_Port parent objects must be linked by a Host association
Realize
Port directions must be compatible
Transmit
Physical_Port
Direction:Physical_Port_Direction
Logical_Port
Direction:Logical_Port_Direction
Receive
Hardware
Software
Signal
Direction:Signal_Direction
Publish
Host
Signal_Format
Sig_Format
Publish
Parameter
Direction:Parameter_Direction
Reference
Signal_Format object and Signal_Format association are created automatically when Output Signal is created
Only Output Parameters may be associated with Signal_Format_Objects

Figure 77. Software-hardware integration data model

6.3 System model-based design example

Figure 78 shows a sample architecture model, which is consistent with the theoretical systems architecture model described in Figure 63.

For simplicity, the assumption will be that higher-level architecture requirements and trade studies have already been conducted, identifying how to control the airplane. A lower-level function includes the ability to control airspeed.

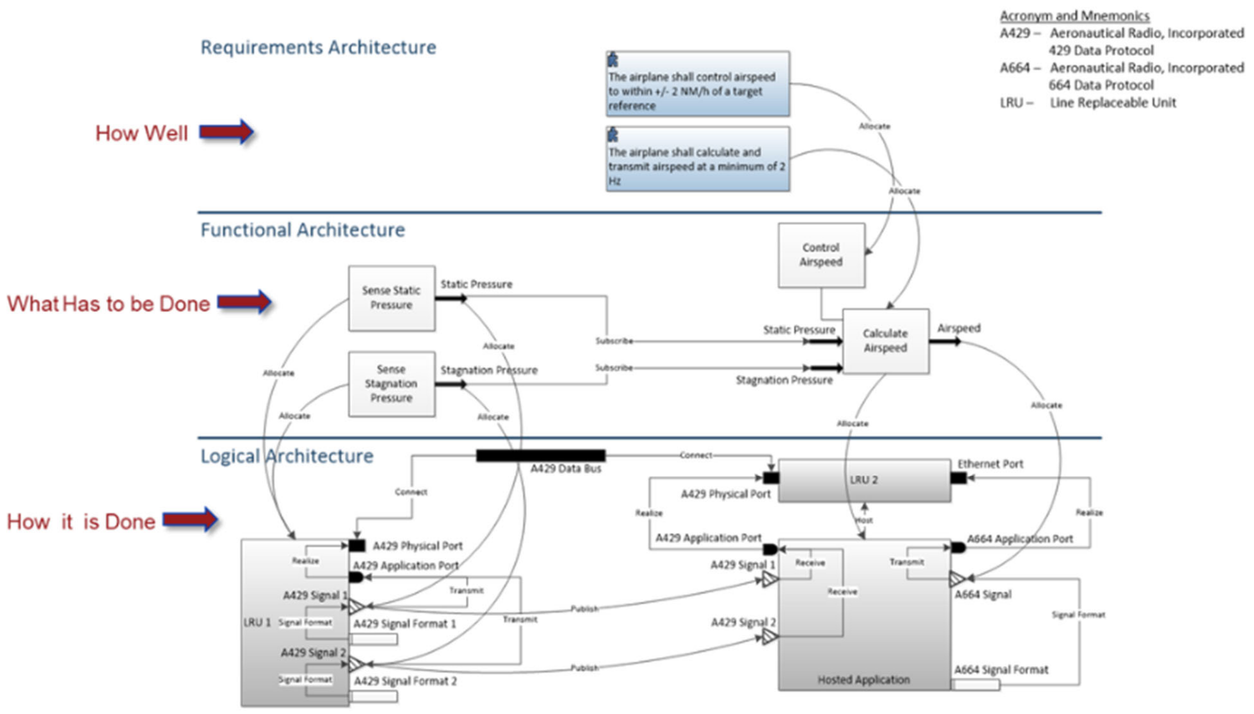


Figure 78. System architecture model example

The logical architecture describes “how it is done.” In this example, the calculate airspeed function will be implemented by a hosted application software that is part of a DIMA.

6.3.1 Requirements architecture modeling example

The requirements architecture data model (as shown in Figure 79) allows the creation of a requirements hierarchy.

The requirements architecture describes ‘how well,” including performance requirements and design constraints. The following requirements will be used for the example:

- The airplane shall control airspeed to within +/- 2 nautical miles per hour of a target reference.
- The airplane shall calculate and transmit airspeed at a minimum of 2 hertz.

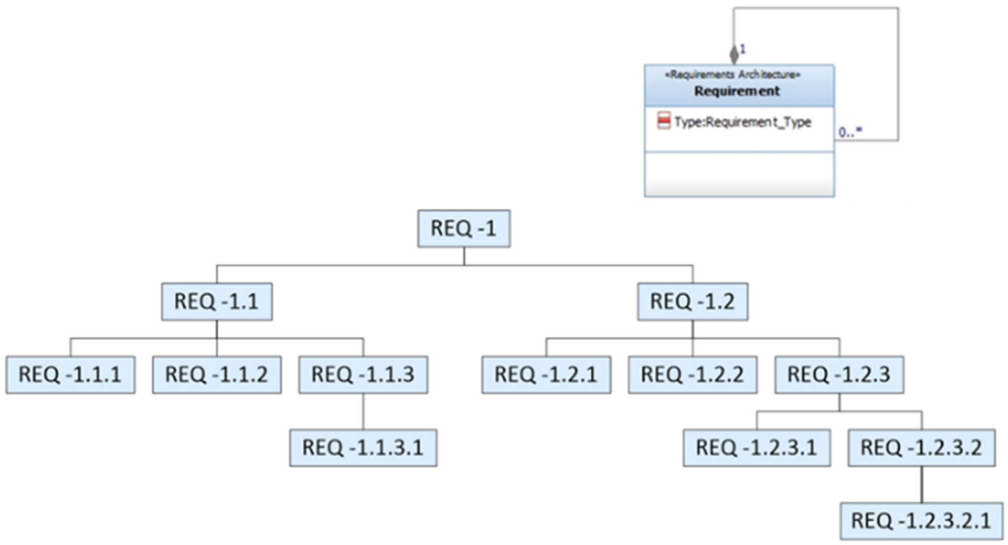


Figure 79. Requirements architecture

6.3.2 Functional architecture modeling example

The functional architecture identifies "what needs to be done." Functions could include the parent function Control Airspeed and the child function Calculate Airspeed.

Figure 80 shows how the function Control Airspeed would be created, per the data model.

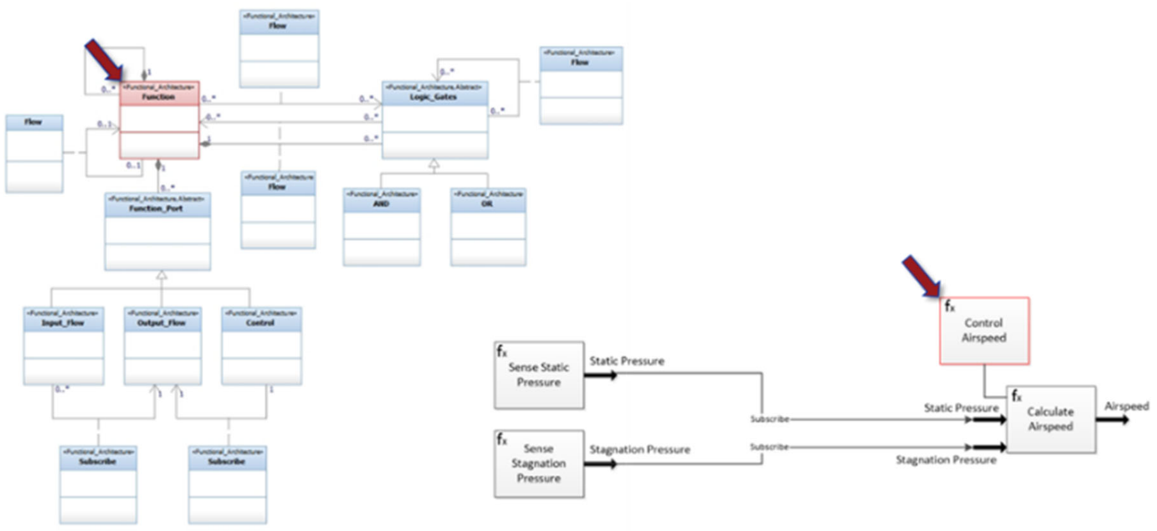


Figure 80. Functional architecture – functions

Figure 81 shows how the function Calculate Airspeed can be created as a child of the function Control Airspeed, per the data model.

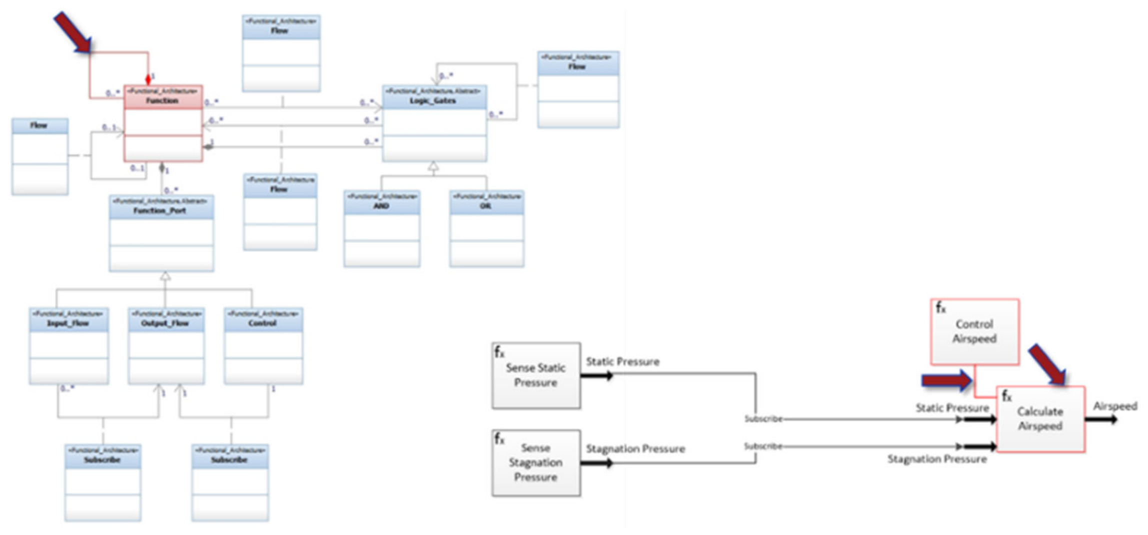


Figure 81. Functional architecture – parent/child functions

Figure 82 shows how the output port and output flow Stagnation Pressure can be created as child of the function Sense Stagnation Pressure, per the data model.

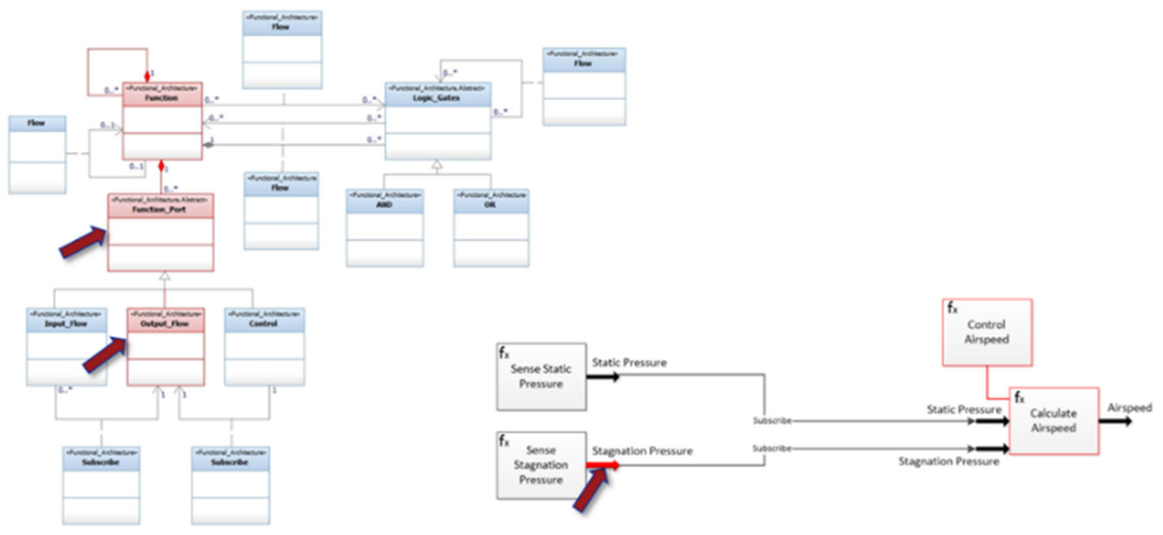


Figure 82. Function ports and output data flows

Figure 83 shows how the input port and input flow Stagnation Pressure can be created as child of the function Calculate Airspeed, per the data model.

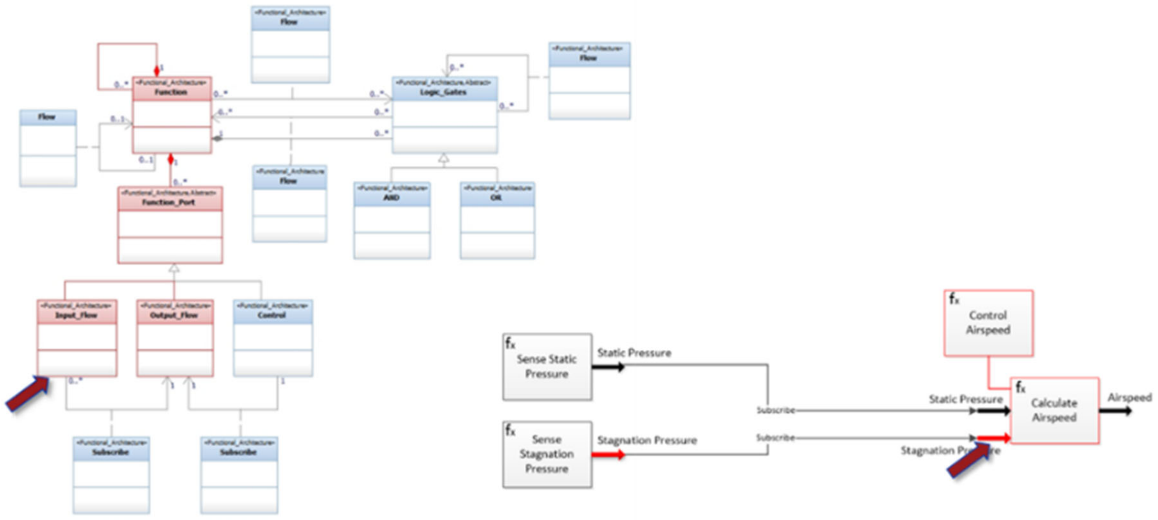


Figure 83. Functional input data flows

Figure 84 shows how the subscribe line can be created between the output port/and input flow Stagnation Pressure can be created as child of the function Calculate Airspeed, per the data model.

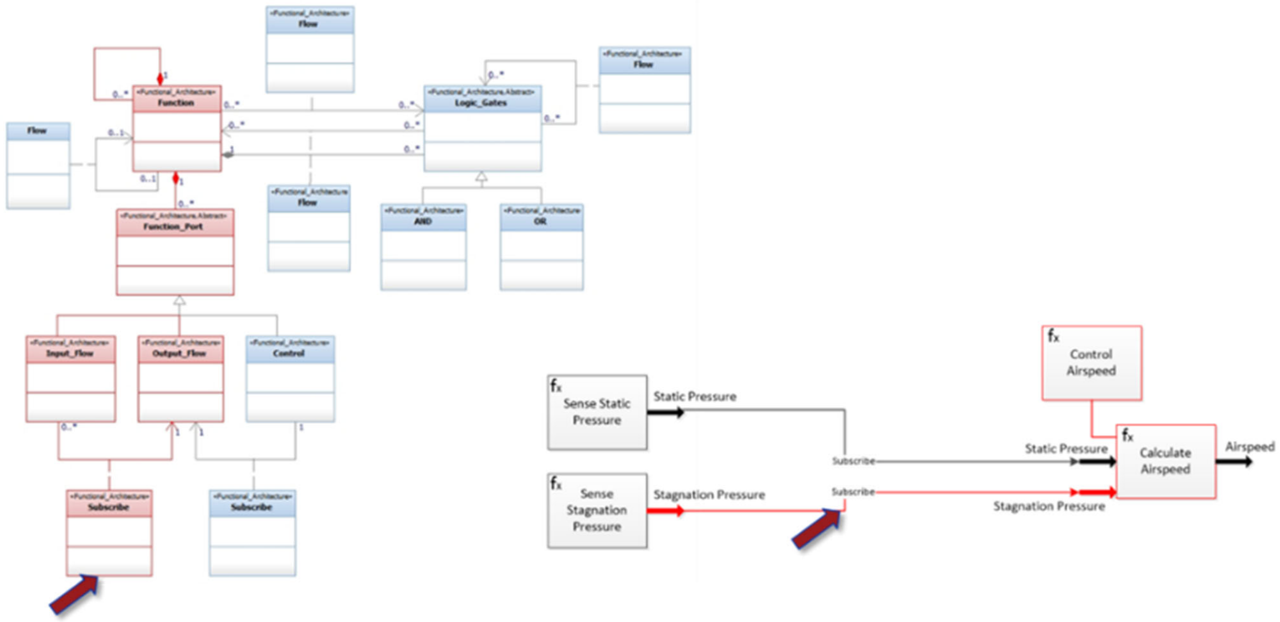


Figure 84. Functional subscription

6.4 Model-based design processes: systems level

The logical architecture is a view of the architecture depicting the organization of logical elements of a system, their relationships to each other, to other architectural elements, and to the system context. Two key elements are the system definition and ports. The logical ports are defined as children of systems (hardware or software); they represent messages received or sent from software. Physical ports are defined as children of systems (hardware) and represent the physical pins or group of pins on an LRU.

As shown in Figure 85, an ARINC 429 port can be created, per the data model.

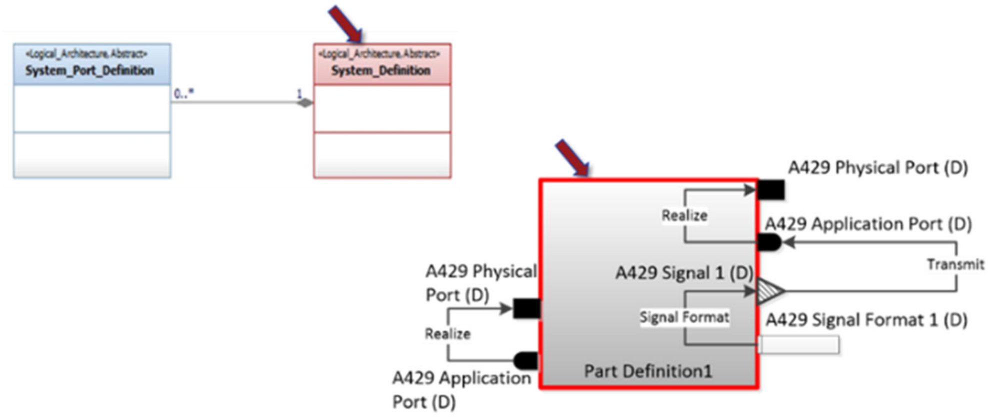


Figure 85. Systems and ports modeled

As shown in Figure 86, an ARINC 429 port can be created, per the data model. Systems objects communicate through system ports.

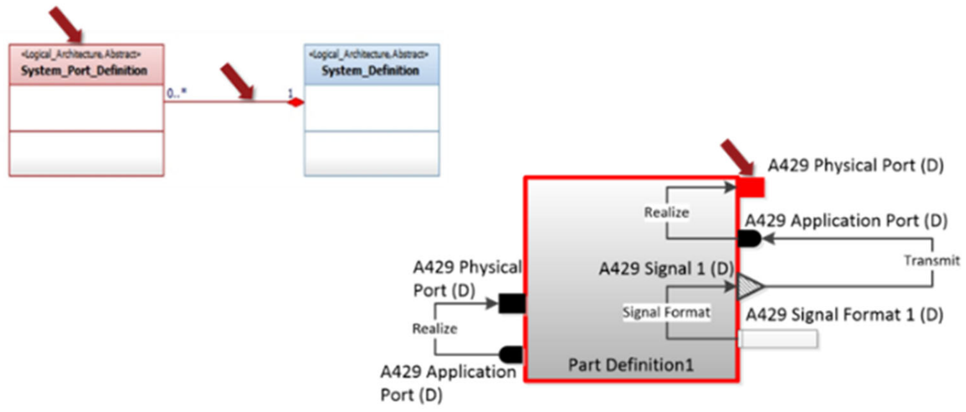


Figure 86. System-system communication via system ports

As shown in Figure 87, system connectivity can be created, per the data model. Physical port objects are used to physically interface system objects. In this example, the connectivity is created by ARINC 429 LRU 1 and ARINC 429 LRU 2.

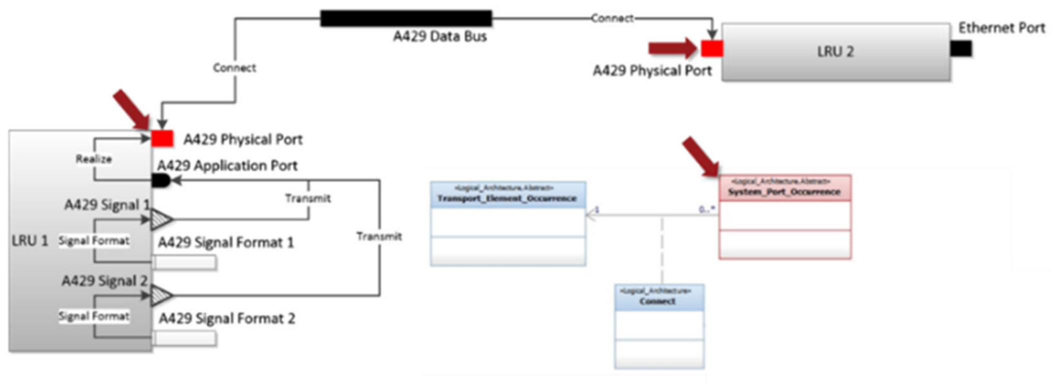


Figure 87. Systems connectivity

As shown in Figure 88, per the data model, systems connectivity is completed by connecting the physical ports to the transport element (in this example, the ARINC 429 data bus).

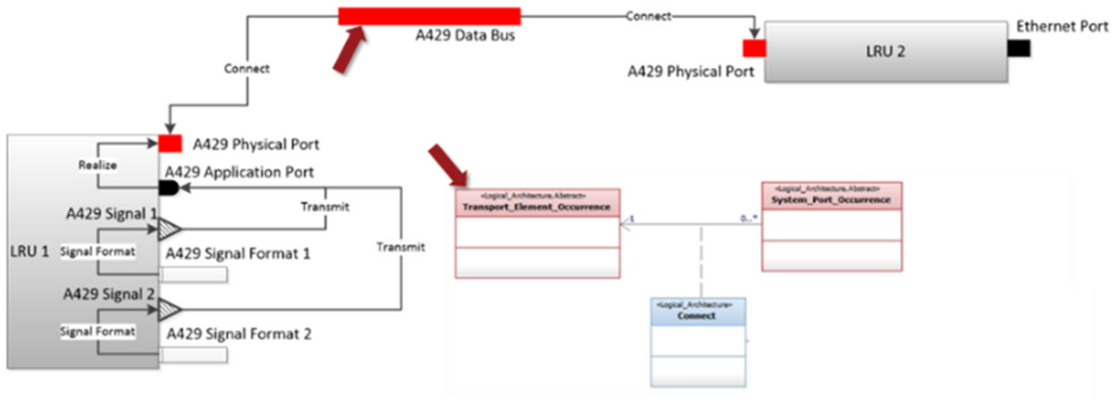


Figure 88. Physical ports and transport elements

6.5 MBD processes to integrate system and software/AEH models

The following examples will show how a data model can support the integration of the system and software/AEH models. Figure 89 shows how an LRU (hardware) can be created, per the data model.

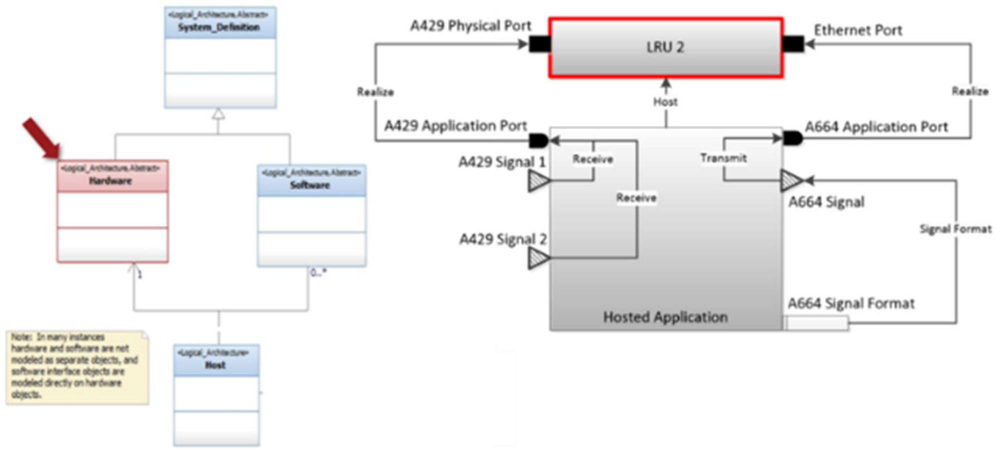

«Logical_Architecture.Abstract» System_Definition
«Logical_Architecture.Abstract» Hardware
«Logical_Architecture.Abstract» Software
«Logical_Architecture» Host
Note: In many instances hardware and software are not modeled as separate objects, and software interface objects are modeled directly on hardware objects.
A429 Physical Port
LRU 2
Ethernet Port
Realize
Host
A429 Application Port
A664 Application Port
A429 Signal 1
Receive
Transmit
A664 Signal
A429 Signal 2
Signal Format
Hosted Application
A664 Signal Format

Figure 89. Hardware

Figure 90 shows how a hosted application (software) can be created, per the data model.

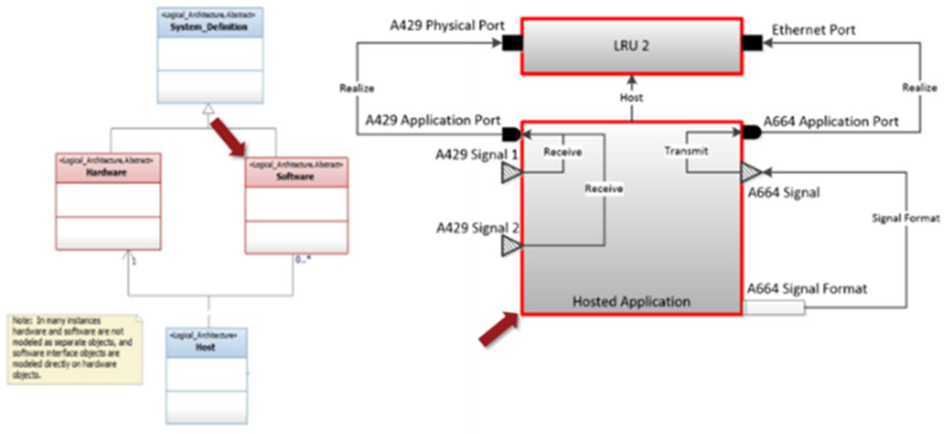
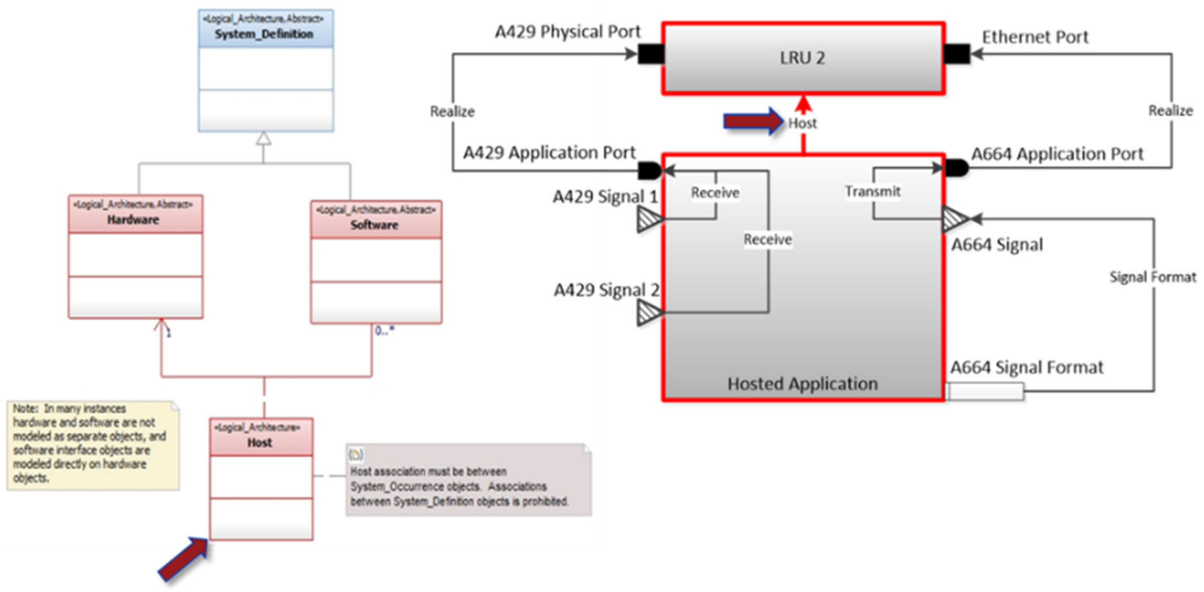

«Logical_Architecture.Abstract» System_Definition
«Logical_Architecture.Abstract» Hardware
«Logical_Architecture.Abstract» Software
«Logical_Architecture» Host
Note: In many instances hardware and software are not modeled as separate objects, and software interface objects are modeled directly on hardware objects.
A429 Physical Port
LRU 2
Ethernet Port
Realize
Host
A429 Application Port
A664 Application Port
A429 Signal 1
Receive
Transmit
A664 Signal
A429 Signal 2
Signal Format
Hosted Application
A664 Signal Format

Figure 90. Hosted application software

Figure 91 shows how a host relationship can be created between software and hardware, per the data model.

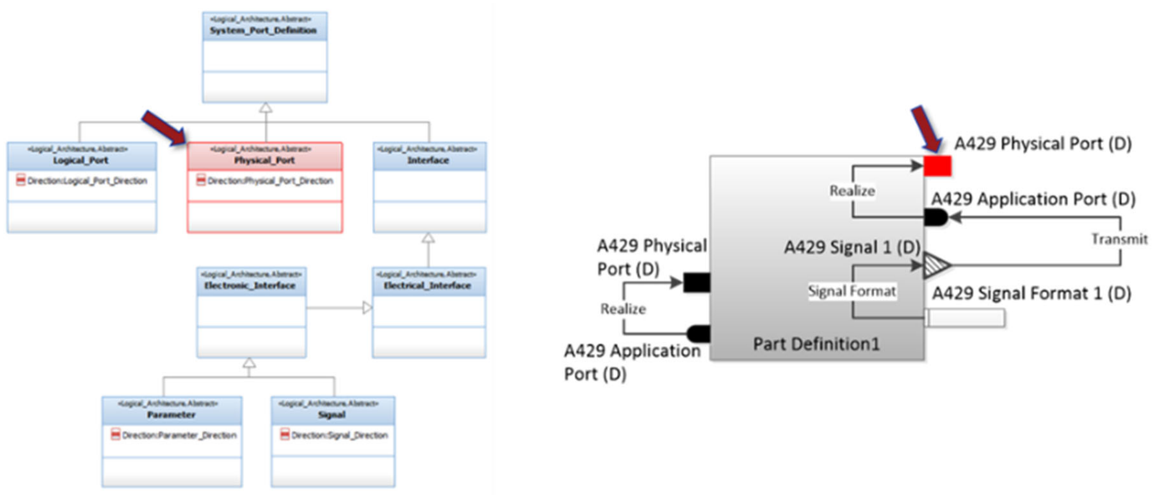


«Logical_Architecture.Abstract»
System_Definition
«Logical_Architecture.Abstract»
Hardware
«Logical_Architecture.Abstract»
Software
1
0..*
«Logical_Architecture»
Host
Note: In many instances hardware and software are not modeled as separate objects, and software interface objects are modeled directly on hardware objects.
Host association must be between System_Occurrence objects. Associations between System_Definition objects is prohibited.
A429 Physical Port
LRU 2
Ethernet Port
Realize
Host
Realize
A429 Application Port
A664 Application Port
A429 Signal 1
Receive
Transmit
Receive
A664 Signal
Signal Format
A429 Signal 2
A664 Signal Format
Hosted Application

Figure 91. Host relationship

The integration of software and hardware will involve the transmission of signals (which contain detailed parameters) via logical and physical ports.

Figure 92 and Figure 93 show how ARINC 429 physical and logical ports can be created, per the data model.



«Logical_Architecture.Abstract»
System_Port_Definition
«Logical_Architecture.Abstract»
Logical_Port
Direction:Logical_Port_Direction
«Logical_Architecture.Abstract»
Physical_Port
Direction:Physical_Port_Direction
«Logical_Architecture.Abstract»
Interface
«Logical_Architecture.Abstract»
Electronic_Interface
«Logical_Architecture.Abstract»
Electrical_Interface
«Logical_Architecture.Abstract»
Parameter
Direction:Parameter_Direction
«Logical_Architecture.Abstract»
Signal
Direction:Signal_Direction
A429 Physical Port (D)
Realize
A429 Application Port (D)
Transmit
A429 Physical Port (D)
A429 Signal 1 (D)
Signal Format
A429 Signal Format 1 (D)
Realize
A429 Application Port (D)
Part Definition1

Figure 92. System port subtype – physical port

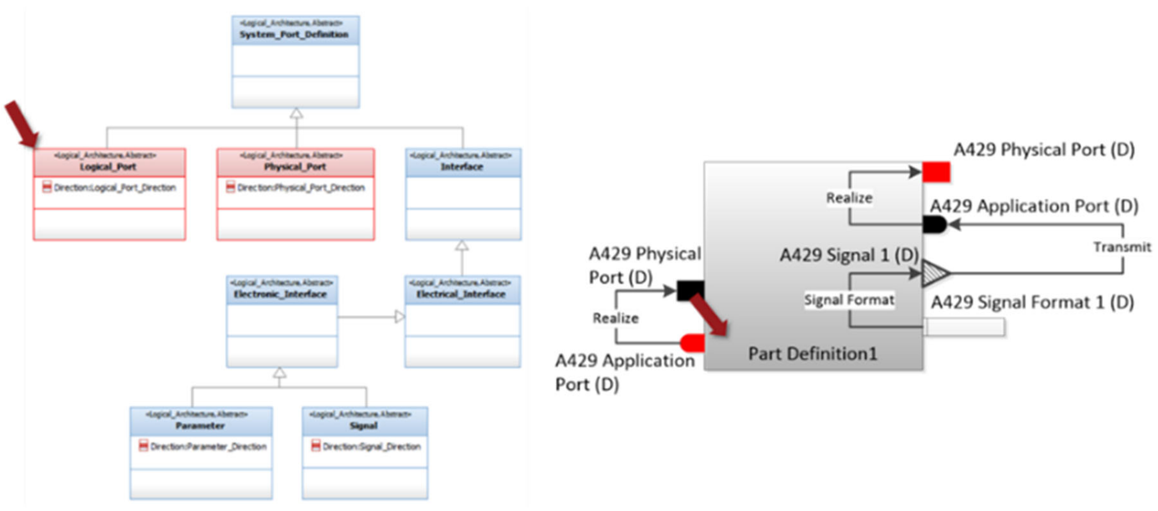


Figure 93. System port subtype – logical port

Figure 94 shows how an ARINC 429 signal can be created, per the data model.

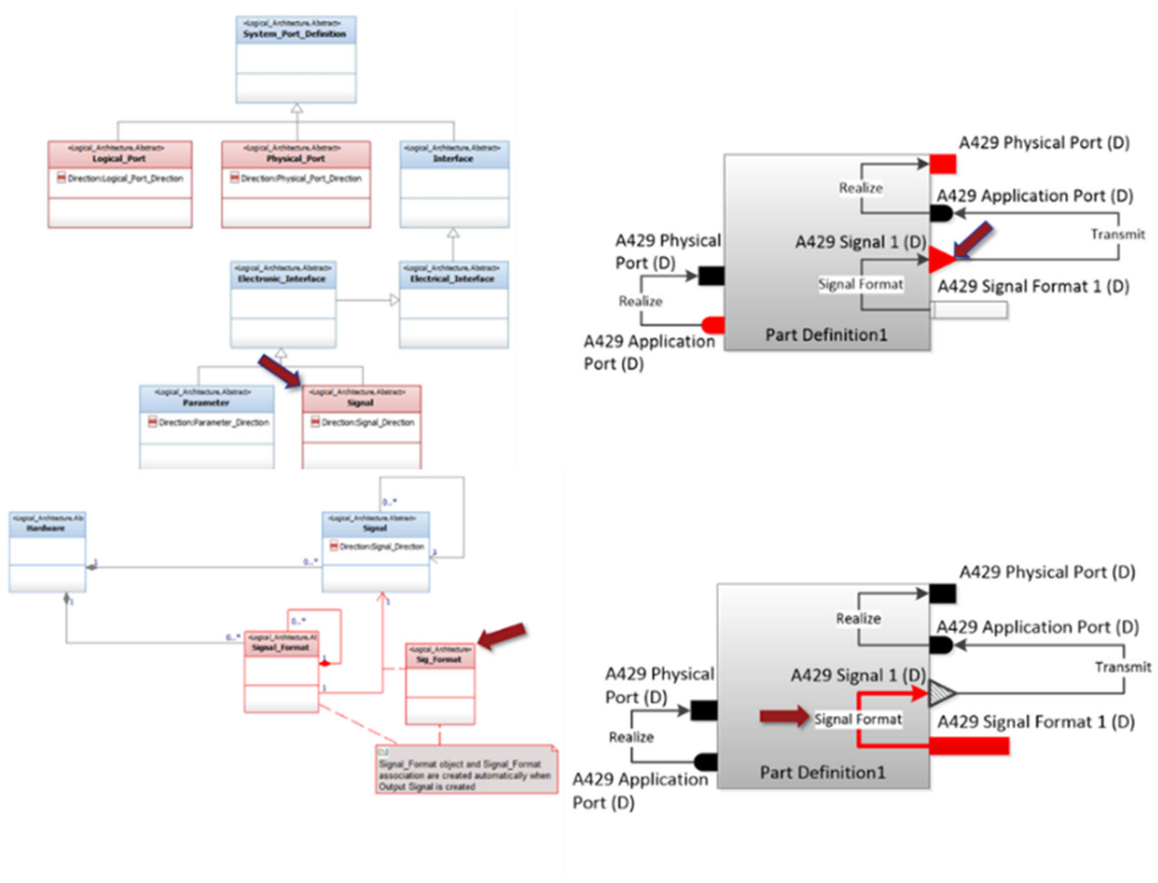


Figure 94. Signal

Figure 95 shows how the signal format and the signal format association can be automatically created when the output signal is created.

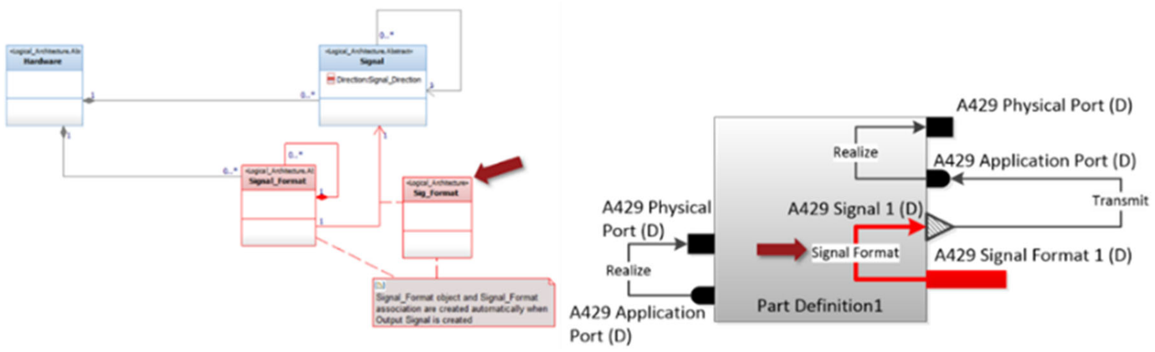


Figure 95. Signal format

Figure 96 shows how the airspeed parameter can be created, per the data model.

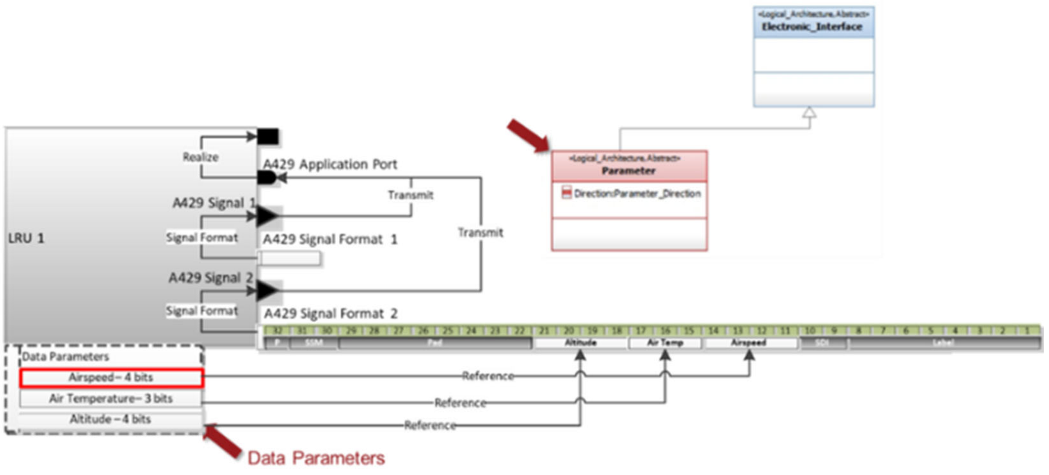


Figure 96. Data parameters

Figure 97 shows how the airspeed data parameter can be incorporated into data words, per the data model.

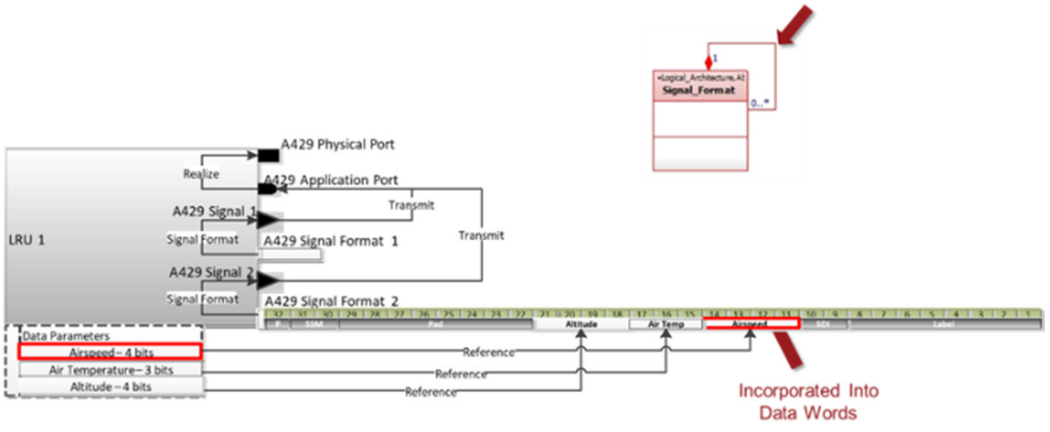


Figure 97. Data words

ARINC 429 signals are single 32-bit data words. Figure 98 shows the detailed signal format incorporation, per the data model.

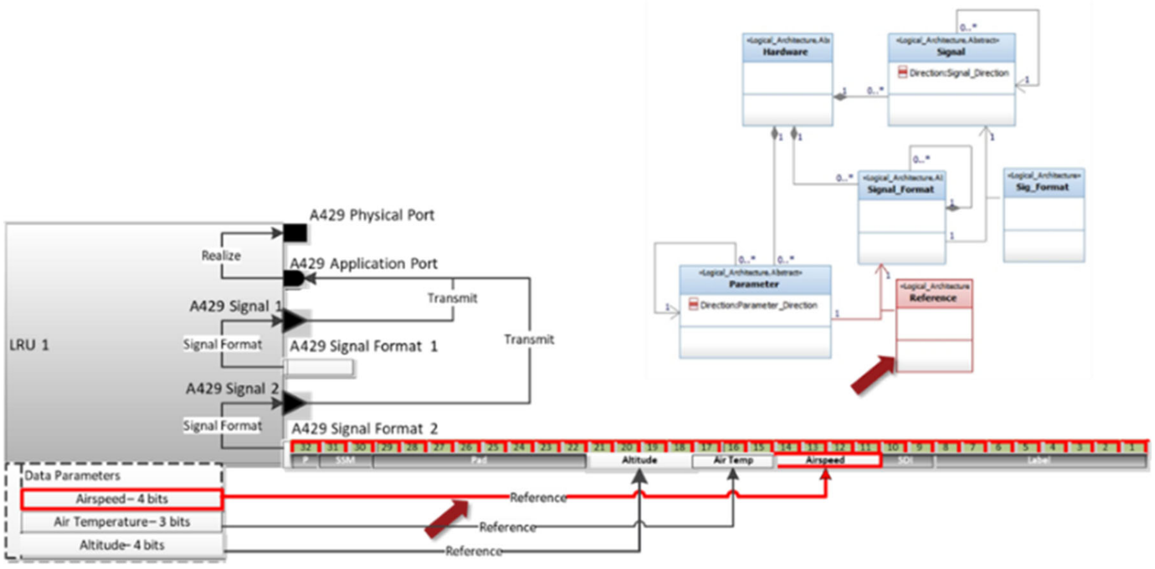


Figure 98. Detailed signal – signal format incorporation

Figure 99 shows how a signal can be created and transmitted, per the data model.

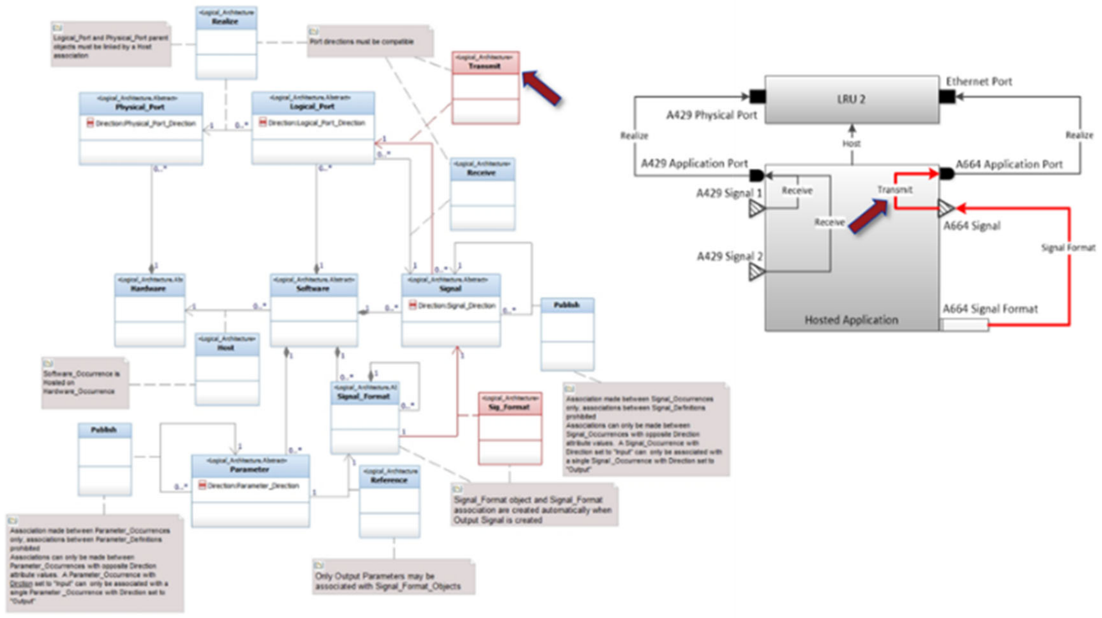
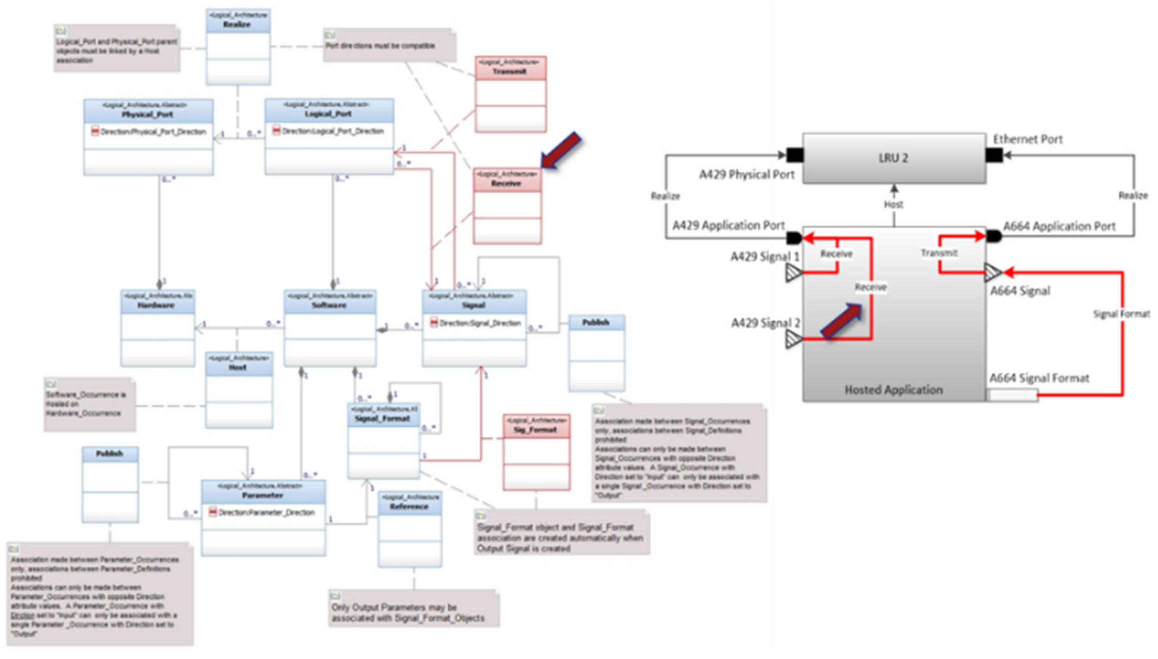


Figure 99. SW/HW integration – transmit

Figure 100 shows how a signal can be received, per the data model.



Realize
Transmit
Receive
Physical_Port
Logical_Port
Hardware
Software
Signal
Publish
Host
Signal_Format
Sig_Format
Parameter
Reference
Port directions must be compatible
Logical_Port and Physical_Port parent objects must be linked by a Host association
Software_Occurrence is Hosted on Hardware_Occurrence
Signal_Format object and Signal_Format association are created automatically when Output Signal is created
Only Output Parameters may be associated with Signal_Format_Objects
LRU 2
Ethernet Port
A429 Physical Port
Host
Realize
A429 Application Port
A664 Application Port
A429 Signal 1
A429 Signal 2
A664 Signal
Signal Format
A664 Signal Format
Receive
Transmit
Hosted Application

Figure 100. SW/HW integration – receive

Figure 101 shows how the logical port can be linked to the physical port, per the data model.

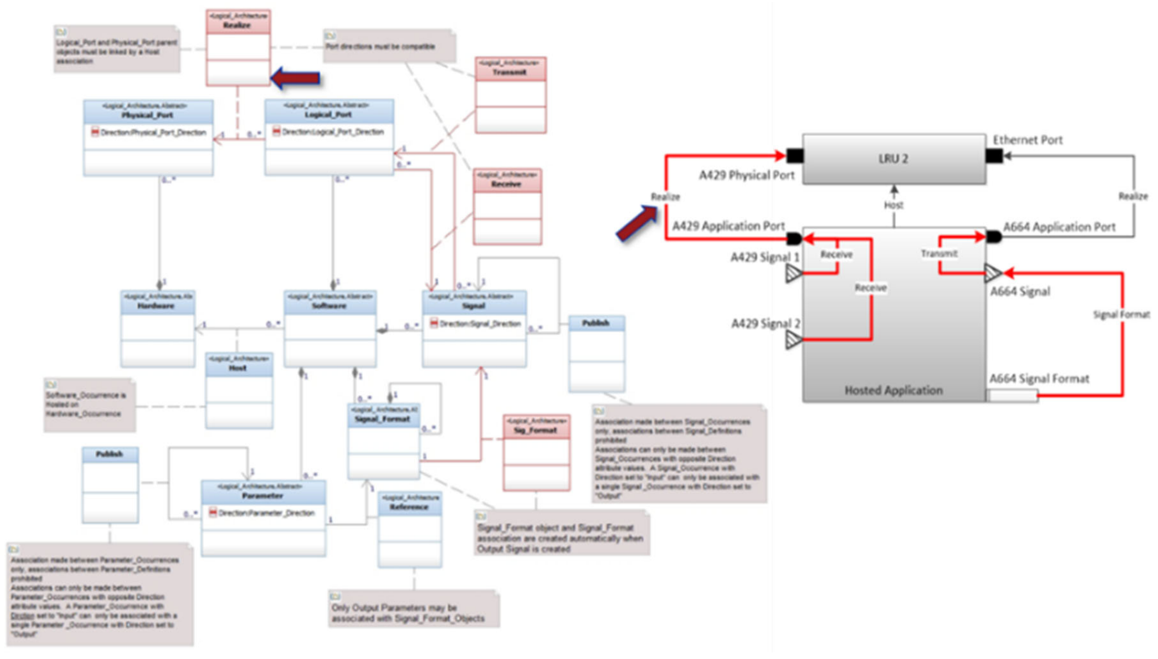


Figure 101. SW/HW integration – realize

Figure 102 show how the interfaces can be created and reconciled, per the data model.

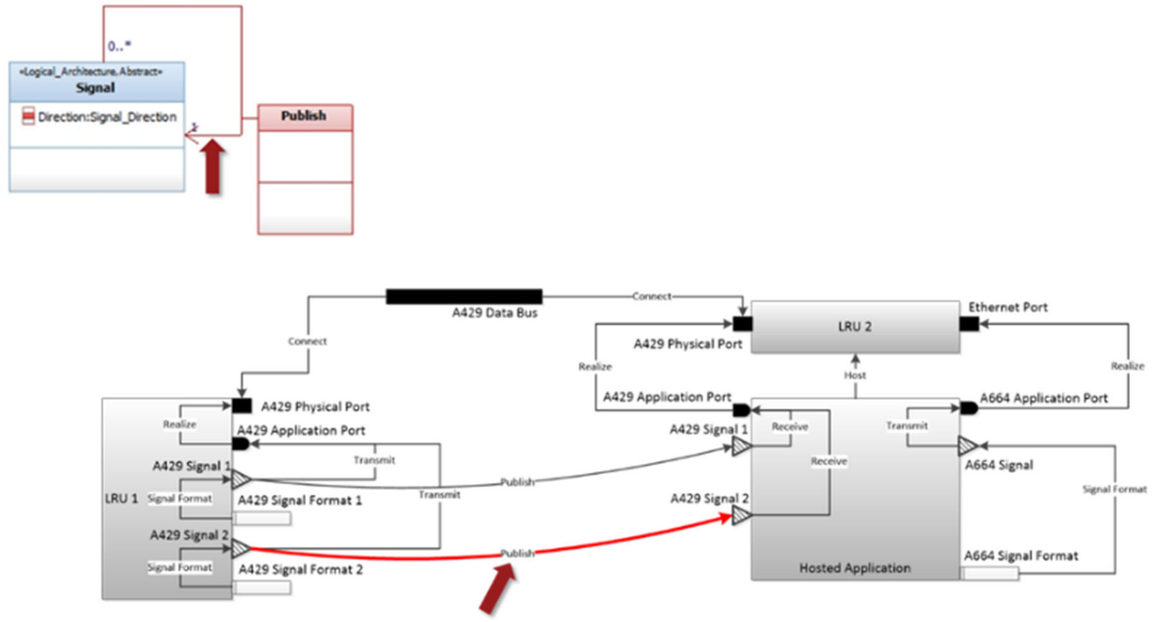


Figure 102. Interface reconciliation

These examples demonstrate how, using a data model, the requirements architecture, functional architecture, and logical architecture can be developed and integrated. This includes the development of system-level, software-level, and hardware-level models, including their respective interfaces and integration.

6.6 Model coverage analysis

RTCA DO-331 [6], Model-Based Development and Verification Supplement to DO-178C and DO-278A, identifies the need to conduct model coverage analysis when the design models are used in the software development process.

The purpose of the model coverage analysis is to ensure the completeness and thoroughness of the verification activities. The models are developed, based on the requirements; the model coverage analysis determines which requirements (expressed by the model) are (or are not) covered by verification cases. The verification cases can consist of one or more of the following:

- Simulation cases.
- Test cases.
- Cases produced through other verification techniques such as formalisms.

The system-level model coverage analysis criteria should include the following:

- All input variables that are driven by a requirement are exercised.
- Each functional block that is exercised.
- Coverage of all the transitions between states that are exercised.
- Each decision or Boolean gate that is exercised.

Both a tool-based and manual process can be used for model coverage analysis:

- Tool-based process – There are industry verification coverage tools that can be used.
- Manual process – Reviews of test procedures, written for textual requirements, against the model to ensure that the procedure exercised all elements in the model.

Model coverage analysis may identify deficiencies. DO-331 [6] identifies the following activities that may need to be conducted, in order to address verification deficiencies discovered during the model coverage analysis. Per DO-331:

- Shortcomings in the requirements-based verification cases or procedures – The verification cases should be supplemented or verification procedures changed to provide the missing coverage. The method(s) used to perform the coverage analyses, based on the requirements from which the Design Model was developed may need to be reviewed.
- Inadequacies or shortcomings in requirements from the Design Model was developed – The requirements from which the Design Model was developed should be modified and additional verification cases developed and verification procedures executed.
- Derived requirements expressed by the Design Model – Appropriate verification cases should exist or they should be developed for the derived requirements and verification procedures should be executed to provide the missing coverage. Previously unidentified derived requirements should be identified, justified, and provided to the system processes, including SSA process.
- Deactivated functionality expressed by the Design Model – Deactivated functionality expressed by the Design Model should be justified. For deactivated functionality expressed by a Design Model that is not intended to be realized in any configuration used within an aircraft or engine, a combination of analysis, simulation and testing should show that its realization is prevented, isolated or eliminated. For deactivated functionality expressed by a Design Model that is only intended to be realized in certain approved

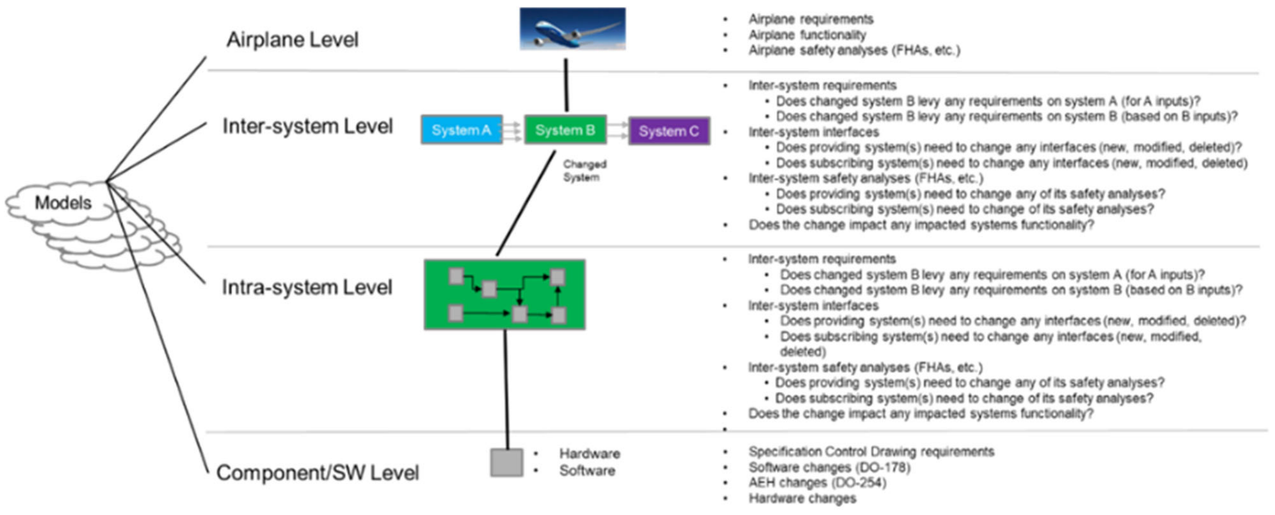
configurations used within an aircraft or engine, the operational configuration needed for normal realization of these requirements should be established and additional verification cases and verification procedures developed to satisfy the required coverage objectives.

- Unintended functionality expressed by a Design Model – Unintended functionality indicates an error is present and the functionality should be removed from the Design Model.

As indicated above, model coverage analysis activities can help identify if there are model deficiencies.

6.7 MBD touchpoints with V&V process

Figure 103 shows the modeling activities occurring at multiple hierarchical levels. As a result, the touchpoints with the V&V process can similarly occur at all levels.



- Airplane requirements
- Airplane functionality
- Airplane safety analyses (FHAs, etc.)

- Inter-system requirements
  - Does changed system B levy any requirements on system A (for A inputs)?
  - Does changed system B levy any requirements on system B (based on B inputs)?
- Inter-system interfaces
  - Does providing system(s) need to change any interfaces (new, modified, deleted)?
  - Does subscribing system(s) need to change any interfaces (new, modified, deleted)
- Inter-system safety analyses (FHAs, etc.)
  - Does providing system(s) need to change any of its safety analyses?
  - Does subscribing system(s) need to change of its safety analyses?
- Does the change impact any impacted systems functionality?

- Inter-system requirements
  - Does changed system B levy any requirements on system A (for A inputs)?
  - Does changed system B levy any requirements on system B (based on B inputs)?
- Inter-system interfaces
  - Does providing system(s) need to change any interfaces (new, modified, deleted)?
  - Does subscribing system(s) need to change any interfaces (new, modified, deleted)
- Inter-system safety analyses (FHAs, etc.)
  - Does providing system(s) need to change any of its safety analyses?
  - Does subscribing system(s) need to change of its safety analyses?
- Does the change impact any impacted systems functionality?

- Specification Control Drawing requirements
- Software changes (DO-178)
- AEH changes (DO-254)
- Hardware changes

Figure 103. V&V touchpoints

The following figures provide examples of systems architecture validation. Figure 104 shows requirements allocation, which helps validate a complete and correct set of requirements.

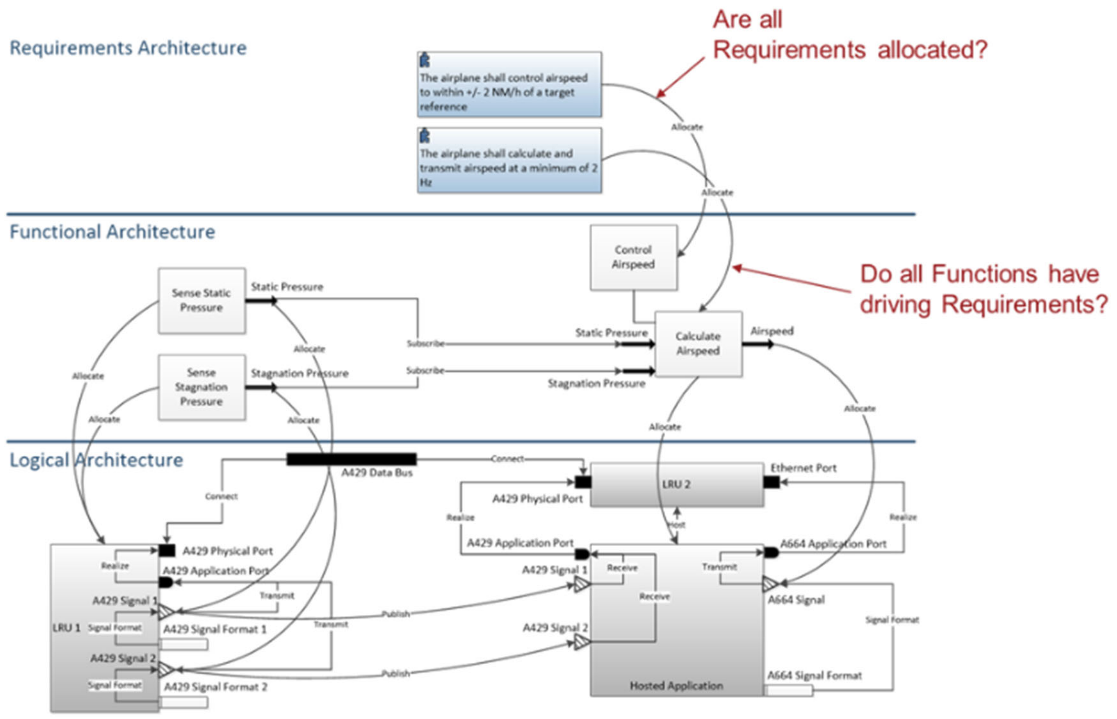


Figure 104. Requirements allocation

Figure 105 shows some of the validation activities for validating that the interfaces are complete and correct.

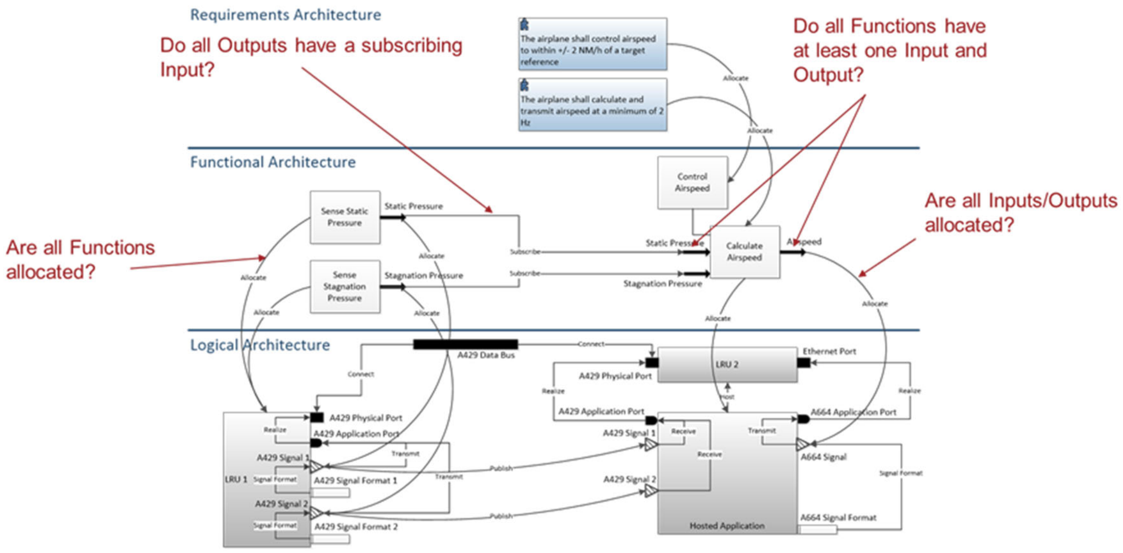
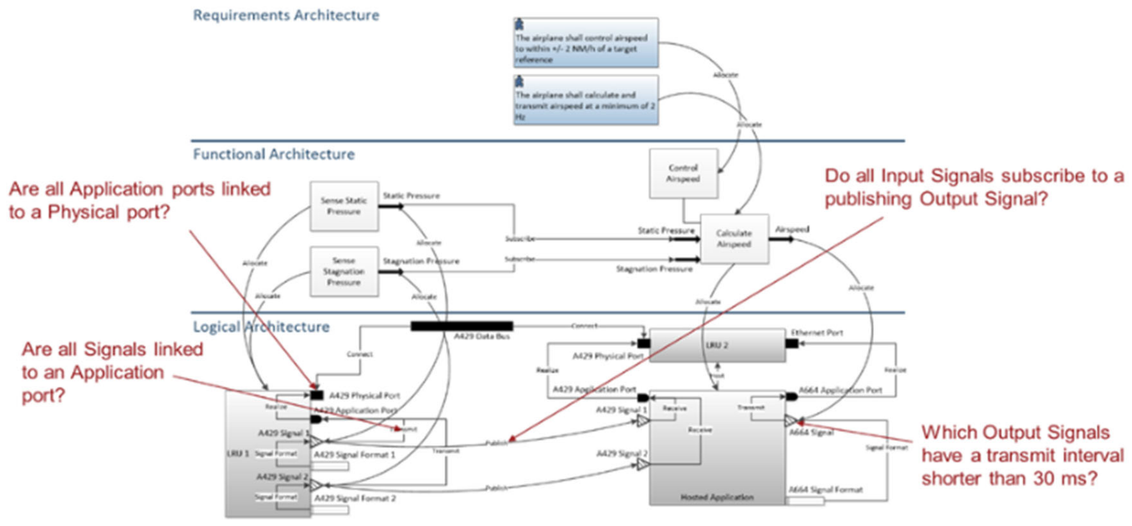

Requirements Architecture
Do all Outputs have a subscribing Input?
Do all Functions have at least one Input and Output?
Are all Inputs/Outputs allocated?
Are all Functions allocated?
Functional Architecture
Logical Architecture

Figure 105. Interface management

Figure 106 shows some of the validation activities for validating that the signals are complete and correct.



Requirements Architecture
The airplane shall control airspeed to within +/- 2 NM/h of a target reference
The airplane shall calculate and transmit airspeed at a minimum of 2 Hz
Functional Architecture
Are all Application ports linked to a Physical port?
Do all Input Signals subscribe to a publishing Output Signal?
Sense Static Pressure
Sense Stagnation Pressure
Control Airspeed
Calculate Airspeed
Static Pressure
Stagnation Pressure
Airspeed
Logical Architecture
Are all Signals linked to an Application port?
A429 Data Bus
LRU 2
Ethernet Port
A429 Physical Port
A429 Application Port
A664 Application Port
A429 Signal 1
A429 Signal 2
A664 Signal
A429 Signal Format 1
A429 Signal Format 2
A664 Signal Format
LRU 1
Hosted Application
Which Output Signals have a transmit interval shorter than 30 ms?

Figure 106. Signal validation

The system architecture needs to be verified to ensure that it is integrated and adequately implements the intent of the requirements.

Table 12 lists modeling objects that can be the touchpoints to the verification process.

Table 12. Verification objects

Verification Object	**Description**
Verification Case	The primary mechanism for defining and documenting how the verification and validation are accomplished and for identifying which portions of the system architecture and system definition or product are being verified.
Verification Procedure	The verification procedure is a detailed description of how the verification will be performed. It may depend on the verification environment.
Verification Model	This model or simulation is developed to support the verification and validation of the system and/or the product. It includes modeling of requirements, functions, functional interfaces, systems, physical parts, and transport elements.
Verification Result	This is the data that is generated as the result of executing the verification procedure.
Verification Record	This is the evaluation of the verification results, with respect to the anticipated results or the expected results.
Verification Environment	This is the definition of the infrastructure needed to perform the verification. For example, product verification and validation may be conducted using a standalone test bench. The standalone test bench would be considered the verification environment.

7 Assess the latest solutions, tools and technologies

This section assesses the latest solutions, tools, and technologies. Table 13 describes the approach to address these research objectives.

Table 13. Latest solutions, tools, and technologies

Research Objectives	**Detailed Work Description**
Assess the latest solutions, tools, and technologies	Identify state of the art for MBSE and MBSA. Identify how MBSE/MBSA implementation experience has shown that errors and omissions with potential of adverse effects on safety can be found and resolved earlier in the design cycle and with change impact analysis, resulting in less effort and disruption to industry and regulatory agencies than with traditional methods. Identify how MBSE/MBSA techniques can be one means of addressing potential gaps in existing guidance related to redundancy and capability in IMA architectures to enable more efficient means of achieving product safety standards. Identify required functionality needed for MBSE/MBSA tools to be effective (tool agnostic). Investigate how model-based design approaches, using formalisms, could mitigate integration and safety issues for process execution problems. Identify MBSE and MBSA as approaches to design that emphasize the injection of additional formalism into pre-existing design processes. These formalisms can be introduced at a variety of levels of abstraction of design. At each level, the formalisms enable a number of new analytic capabilities: Analyses of individual systems (e.g., demonstrating their conformance to requirements). Analyses of interactions between systems (e.g., demonstrating their conformance to communication protocol requirements). Analyses of the emergent behavior created by the combination of individual systems into a greater system of systems (e.g., demonstrating properties of the system of systems given properties of the systems themselves). Identify integration touchpoints of MBSE/MBSA with V&V processes.

7.1 MBSE/MBSA state of the art

MBSE helps transition from the traditional systems engineering, which consisted of data in static documents. Often, the same data was required in multiple documents, which increased data synchronization efforts. MBSE can improve efficiency by having the data as set of object models, and then creating relationships between the different objects within the model. As a result, users are able to create different views of the same data – as opposed to re-creating the data multiple times to create different views.

Having a common single source of truth, with multiple viewers looking at the same data (as opposed to creating multiple overlapping data, which is required to be continuously synchronized and harmonized), provides great benefits.

This "single source of truth" improves efficiency, because all users can make real-time updates. It also enables virtual systems integration and test. The ability to do early systems integration and test helps identify potential design problems earlier, which reduces product development costs. Capturing and managing architecture data in a single data environment can help ensure product quality and enable affordability. The probability of "getting it right the first time" is increased when all the required information is contained in a single data model. The following benefits can be achieved:

- Single data environment ensures completeness and consistency of design data. The users are looking at the same data but from different perspectives, which helps ensure the accuracy. In the spirit of the famous analogy of the blind man and the elephant, all users can see both their "part," as well as the larger picture. In addition, all users are working with the latest data. This improves the fidelity of the data.
- Rich database permits multi-user input and immediate synchronization, improving efficiency and productivity. This helps with intangibles, such as minimizing the time associated with interpretation of intent. It also minimizes the re-creation of data for downstream processes.
- Traceability through model elements enables efficient change/impact analysis, enabling a more adaptable system. This helps ensure that modifications to the models are clearly communicated to all potential impacted customers. This is particularly true if the model is set up in a "publisher-subscriber" framework. Users can immediately be notified if there are any interface changes.
- Use of a single data environment results in data availability throughout program life cycles

- Robust query engine allows rapid assessment of the integrated database, finding anomalies early, thus preventing rework. Queries can be used to identify missing, incorrect, or incompatible data. Examples of errors are unallocated requirements and functions; system elements with no allocated driving requirements or functions; system attribution not matching requirements specifications; unsubscribed published data (and vice-versa); publisher-subscriber data definition discrepancies; published data not physically routed to subscribers properly; bandwidth overutilization; timing noncompliance; and, detecting where actual values of measured performance parameters exceed allocations.

MBSE/MBSA implementation experience has shown that errors and omissions with potential adverse effects can be found and resolved earlier in the design cycle and with change impact analysis. It is important to emphasize the existing processes are safe. The main benefits seen to date with MBSE/MBSA are improved efficiencies in identifying (and fixing) potential problems earlier.

Products are comprised of components, which are often produced by multiple suppliers. The components are often procured through contracts, which impose requirements and specifications, which include interface definitions. Ensuring the correctness and completeness of the specifications will increase the likelihood that the components will properly integrate when delivered. This reduces rework, which mitigates schedule and cost risk. Because models can be integrated into an overall system functional model, this increases the ability for first pass quality. Models also enable emulation. This can be particularly helpful if not all of the components will be delivered concurrently with the same level of fidelity (which is typical during the early stages of a development program).

The more complete, accurate, and properly cross-referenced the specifications are, the better the components integrate when delivered, thereby reducing the cost and schedule impacts of rework. Specification discrepancies not discovered in models are destined for the test bench or physical system where their resolution is costly. Reconciling requirements and interface definition across documents is arduous and error prone. It can be more effectively accomplished using model-based methods, as depicted in Figure 107.

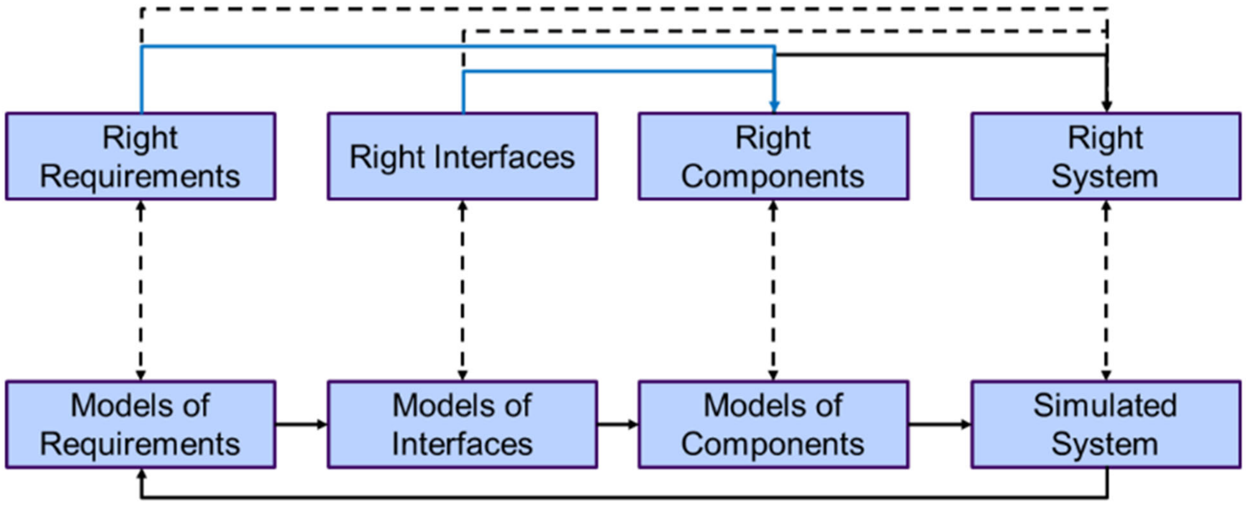


Figure 107. Modeling validation

Boeing experienced some of these benefits when it used model-based engineering when developing the Air Data Reference Function (ADRF) for the 777X. The ADRF is a key avionics function, processing signals from pressure and temperature probes and computing aircraft state parameters, such as airspeed and altitude. Using an MBSE approach gave the engineers the opportunity to work faster, reaching higher first-time quality through model-based engineering. Early integrated testing of the model in the simulator helped to verify and correct the system's interfaces and behavior, long before such testing could be achieved on the design through conventional development. The ADRF project for the 777X proved that a MBSE approach could improve quality and communication, de-risk the program, and significantly reduce development cost and time. Requirements could be validated at a very early stage in the program, avoiding ambiguities, erroneous information, and other specification flaws.

Boeing also experienced positive results in achieving flight test stability as a result of using modeling. MBSE models and, specifically system architecture models, can help minimize errors during the systems development and design. It has been shown that modeling can result in increased flight test stability on current commercial aircraft programs, compared to equivalent programs in the 1990s. Referencing Figure 108 below, there was increased design stability during the flight test program, even though there were increased interfaces. The resultant benefit realized from this additional modeling effort is an order of magnitude decrease in required ICD changes during test on current programs compared to earlier programs.

The ability to ensure the model is complete and accurate is greatly enhanced through rule embedding, user definable queries, real time input/output discrepancy checking, and other analysis features incorporated within the MBSE environment. These features greatly accelerate system architecture model maturity, thereby allowing aggressive development schedules to be maintained. The number of interfaces for current commercial airplane digital systems can be an order of magnitude more than systems designed in the 1990s. However, the interface errors experienced during test are now an order of magnitude lower than in previous generations of commercial airplane. Hundreds of thousands of interface errors are now discovered in system architecture models rather than in integration labs. The use of model-based methods has also significantly reduced the flow time required to validate a new airplane ICD revision.

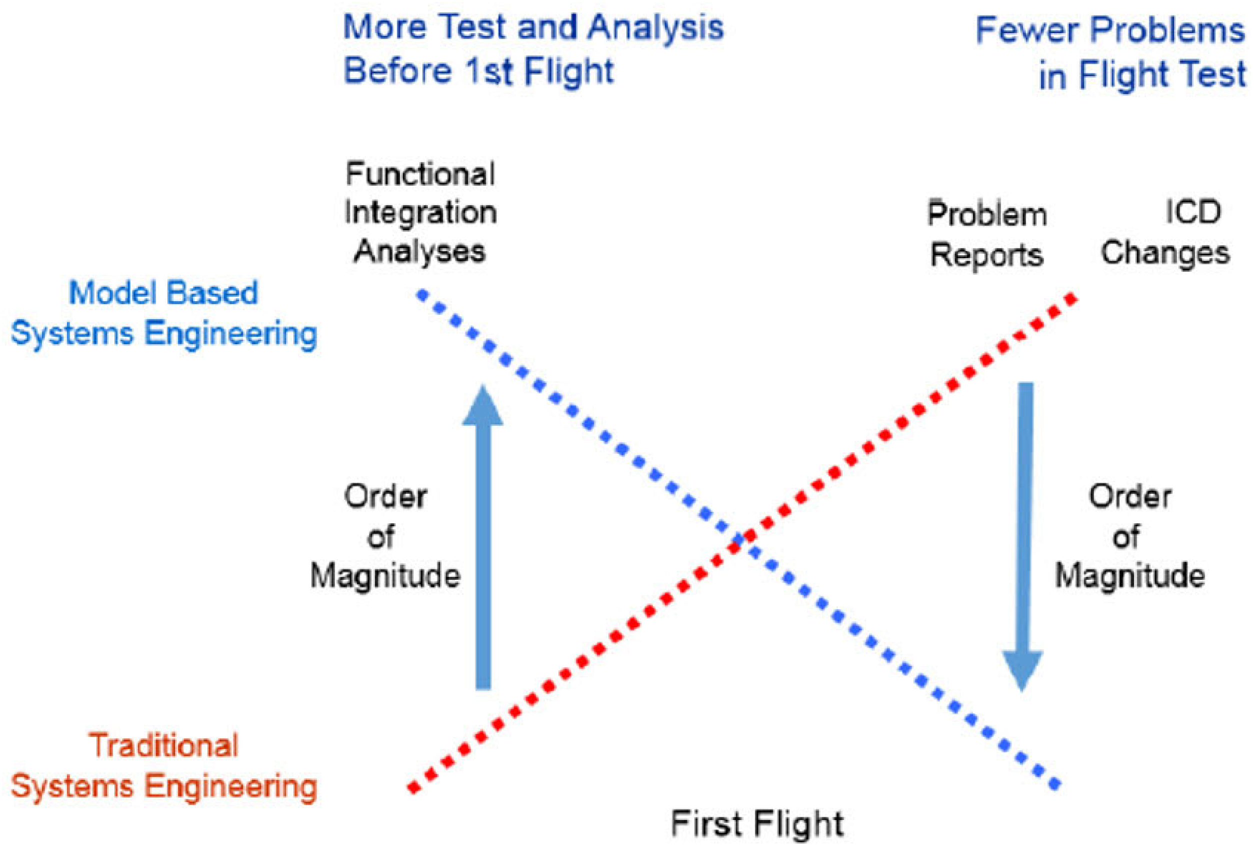

More Test and Analysis Before 1st Flight
Fewer Problems in Flight Test
Functional Integration Analyses
Problem Reports
ICD Changes
Model Based Systems Engineering
Order of Magnitude
Order of Magnitude
Traditional Systems Engineering
First Flight

Figure 108. Modeling to reduce flight test discoveries

Boeing has achieved other benefits from modeling. By assigning timing, latency, data size, and other attributes to interface objects within the model, the data networks can be demonstrated to be compliant through analyses, such as bandwidth utilization and end-to-end latency. This is very helpful in being able to help show that a network is bandwidth and timing compliant.

7.2 Functionality needed for MBSE/MBSA tools

The INCOSE SE Vision 2020 defines MBSE as “the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycles.”

Multiple MBSE tools exists. Table 14 is a subset and is not meant to be an exhaustive list of MBSE tools.

Table 14. MBSE tools

Tool	**Publisher**	**Modeling Standard**
Cameo Systems Modeler	NoMagic	SysML
Rhapsody	IBM	SysML and DoDAF
Core	Vitech	SysML and DoDAF
Modelica and Dymola	Dassault Systemes	Modelica
Innoslate	Spec Innovations	SysML and DoDAF
Enterprise Architect	SPARX Systems	SysML
Capella	PolarSys	ARCADIA

The intent of this paper is to remain tool agnostic. As a result, there will not be a detailed comparison of different MBSE tools.

As shown in Figure 109, the MBSE modeling tools should be built to support the model-based processes.

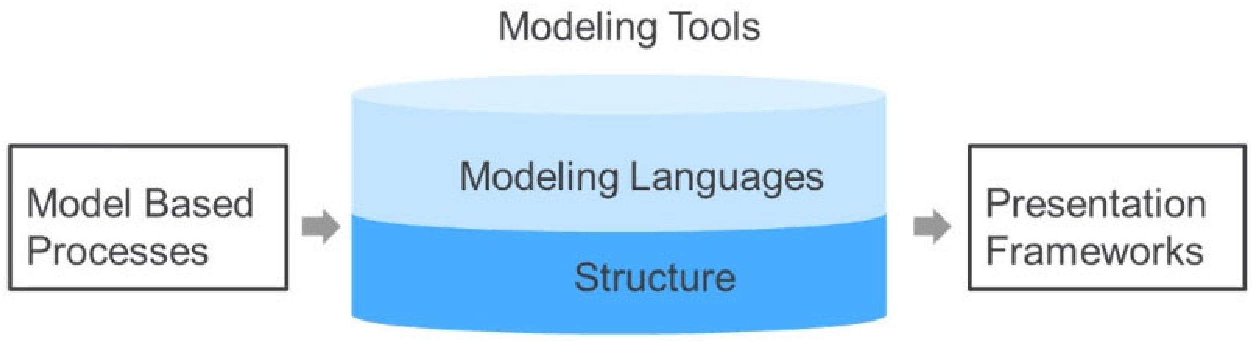


Figure 109. Model based tools

It is important to re-evaluate the existing document-based processes and determine what, if any, changes need to be made to support MBSE. Simply digitizing the existing processes and artifacts will often not achieve the desired MBSE benefits, particularly if the inefficiencies of the paper-based approach are simply included in the digital realm.

An MBSE tool consists of modeling languages (such as UML and SysML) and structure, which defines the relationships between the different modeled objects. (Reference section 6 for additional information.)

The modeling tools will support presentation frameworks. Examples include Department of Defense Architecture Framework (DoDAF) and Unified Architecture Framework (UAF), which provide visualization for different stakeholders.

A diagramming interface is a necessary component of an MBSE environment to allow human comprehension of the model. A graphical representation of a model can be an improvement over simply text-based descriptions ("a picture is worth a thousand words"). A diagram is a good starting point for stakeholder discussions, ensuring that there is a consistent and complete understanding of the model.

However, a graphical representation of a model is not necessarily sufficient to be able to define and analyze the model, particularly depending on the complexity of the model. A complex model may consist of millions of objects and relationships. It is not practical to be able to analyze millions of objects and their interrelationships for completeness and correctness. The inability to do this could impact the integrity of the model (and its resultant benefits). Figure 110 shows the transition that occurs from text-based document to diagrams/visualization to query capabilities.

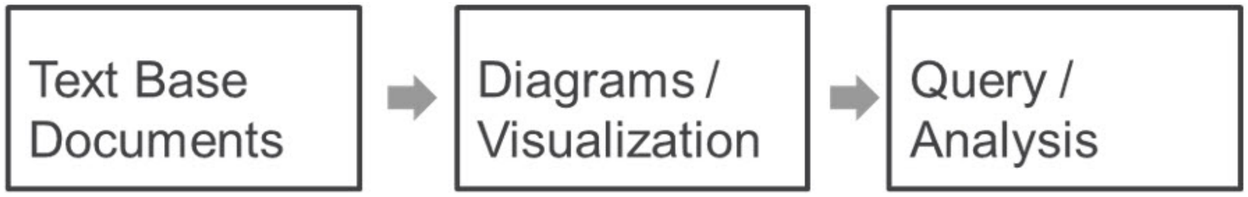


Figure 110. Transition from text to diagrams to queries

Figure 111 shows the relationships between diagramming and visualization, as the design matures and becomes increasingly more detailed. This highlights the importance for a query engine to support analyses, helping validate the completeness and correctness of the models. As the model size increases, the importance of a query engine becomes more significant than the visualization capabilities. In Boeing's experience, as the models mature and grow, most experienced users focus less on graphical viewing utilities and work directly in the database. Diagrams are initially used to establish and visualize the high-level architecture. As the architecture models are decomposed and defined in greater detail, however, the ability to comprehend the models by visual review becomes limited.

Although diagramming interfaces still have utility at later stages in a program, visualization of the model ceases to be the goal. The goal becomes to establish model completeness, correctness, and accuracy, and this is best accomplished through queries and reports. Queries are particularly important to be able to analyze current commercial airplane digital network models.

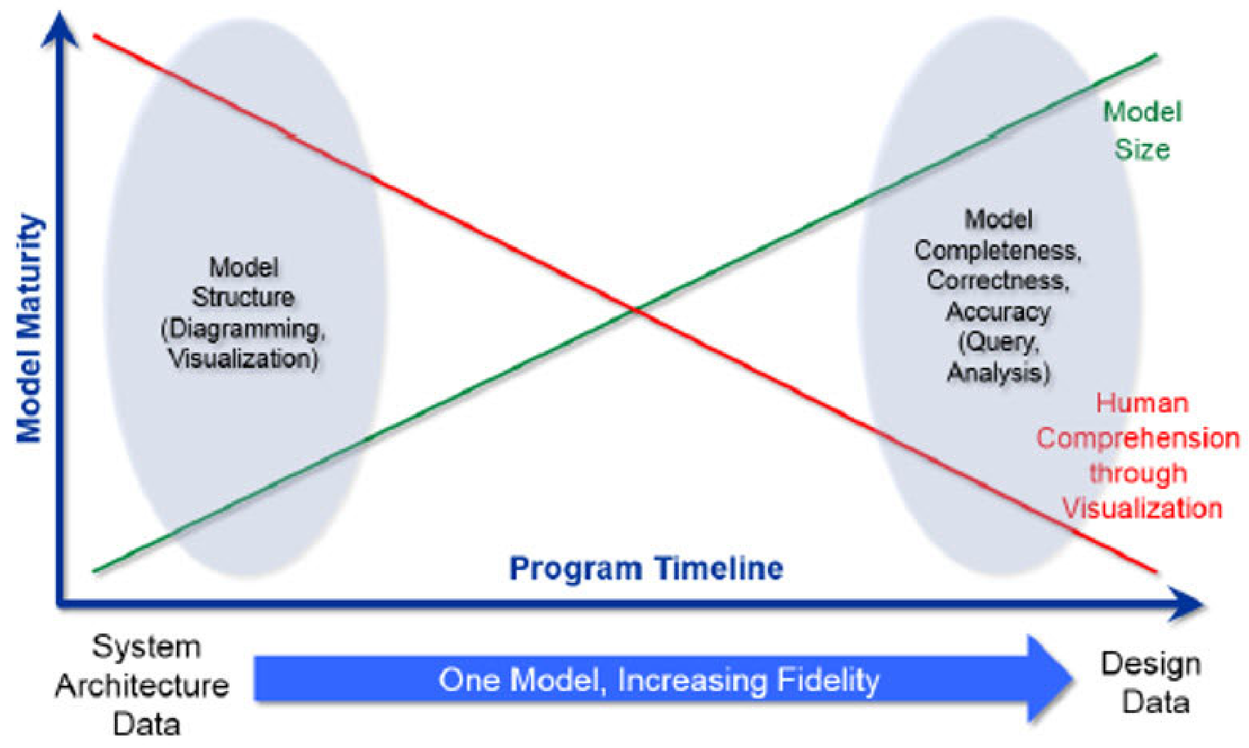


Figure 111. Visualization and queries

An MBSE/MBSA tool should be able to support the following:

- Requirement management – process of documenting, analyzing, tracing, and prioritizing requirements.
- Functional architecture – an architectural model that identifies system function and their interactions.
- Logical architecture – defining how the system will realize the required functionality.
- Interface control definition – describing the interface or interfaces between subsystems or to a system or subsystem.
- Verification and validation – independent procedures for checking that the system meets requirements and specifications and fulfills its intended purpose.
- Content management – ability of the application to publish, edit, control change, and modification of content.
- Scalability – ability of a system to expand to meet business needs.
- Extendibility – systemic measure of the ability to extend a system and the level of effort required to implement the extension.

To be effective in doing so, an MBSE/MBSA tool would require the following:

- Intuitive user interfaces.
- Creating architecture.
- Data model.
- Query engine.
- Access to tool suite(s).
- Scalability and database management.
- Support for data reference libraries and data reuse.
- Configuration and version control of all objects.
- Bulk import/export/update capabilities.

Table 15 identifies potential criteria for evaluating an MBSE/MBSA tool.

Table 15. Potential criteria for evaluating an MBSE/MBSA tool

MBSE/MBSA Tool Functionality	**Definition**
Intuitive user interfaces	Ability to support systems design innately or by interacting/importing information from other tools or data formats.
Creating architecture	Ability to represent architecture hierarchies, objects, attributes, and relationships. This architecture should be viewable in a tree structure. There should be an ability to copy and reference data.
Data model	Ability to create custom subtype objects and custom attributes. User defined business rules for objects and links are supported.
Query engine	Ability to perform customizable queries and to have robust batch processing capabilities. It is important to be able to query the model early and continuously to identify missing design elements.
Access to tool suite	Requirements for roles to be customizable and user groups are supported. The tool should be able to support hundreds of globally distributed users. To protect intellectual properties, data, attribute, objects, and links should be accessible by granted permissions and controllable.
Scalability and database management	Ability to work in a load balanced and high availability environment. Ability to support multiple databases and a configurable and modifiable schema. Consistent performance when managing large quantities of data and users.

MBSE/MBSA Tool Functionality	**Definition**
Configuration and version control of all objects	Ability to perform change management capabilities on the model, such as baselining, versioning, and managing links. Ability to branch/merge and support variants.
Bulk import/export/update capabilities	Support bulk creation, deletion, and modifications of objects, links and attributes.
Baseline/Change/State tracking	Ability to identify and manage changes against a baseline.

It is important for the tool to support attributes that state the version and state of the elements. Baseline capability alone will often not be adequate, because it does not describe the required level of granularity. After a baseline is taken, changes will occur. Unless there is enough granularity, the information is baselined again with a new identifier. Often, a tool is run to identify the differences between the new and a previously approved baseline and the changes are re-reviewed. This can make the process cumbersome and difficult to manage, especially if the repository is distributed between geographically distributed teams. A much better approach is to build in a system with fine granularity that allows the extraction of the changes from the repository, and to review the changes and corresponding state transitions in the repository. The tool can automatically perform impact analysis, if the dependent nodes are connected. The elements in a model repository should have a version/baseline/extraction and state tracking capabilities built in, otherwise the benefits may not be realized.

A software tool needs to provide, at a minimum, this level of required functional to support MBSE/MBSA.

7.3 Formalisms at different abstraction levels

Model-based design approaches, using formalisms, can mitigate integration and safety issues for process execution problems. Formalism can be used to help with the following:

- Organize to support the use of MBSE.
- Use the model as the authority (not the documents –which reflect the model).
- Ensure appropriate access to the data.
- Use the model (architecture) to perform SE.
- Use data elements for functional and operational analysis to derive requirements.

- Use data elements to determine system trade studies to meet evolving needs.
- Derive verification activities from the data elements.
- Generate customer artifacts directly from the model.
- Use the data to ensure quality (i.e., the “right design”).
- Query the model early and continuously to identify missing design elements.
- Query the model to prevent overdesign and unnecessary costs.
- Use the model throughout the entire lifecycle.
- Share across various development and support organizations.

MBSE/MBSA, with its formalisms, can enable the integration of the different models at different abstraction levels. Design is recursive and iterative, but iteration should be occurring at increasing lower levels of details over time (i.e. difference between modifying existing message between LRUs and adding new LRUs). This highlights the importance of having formalisms at the different abstract levels.

Figure 112 shows examples of formalisms at different abstraction levels, which can include:

- Aircraft level.
- Intersystem level.
- Network (for DIMA architectures).
- Intra-system.
- Component.
- Dynamic modeling.

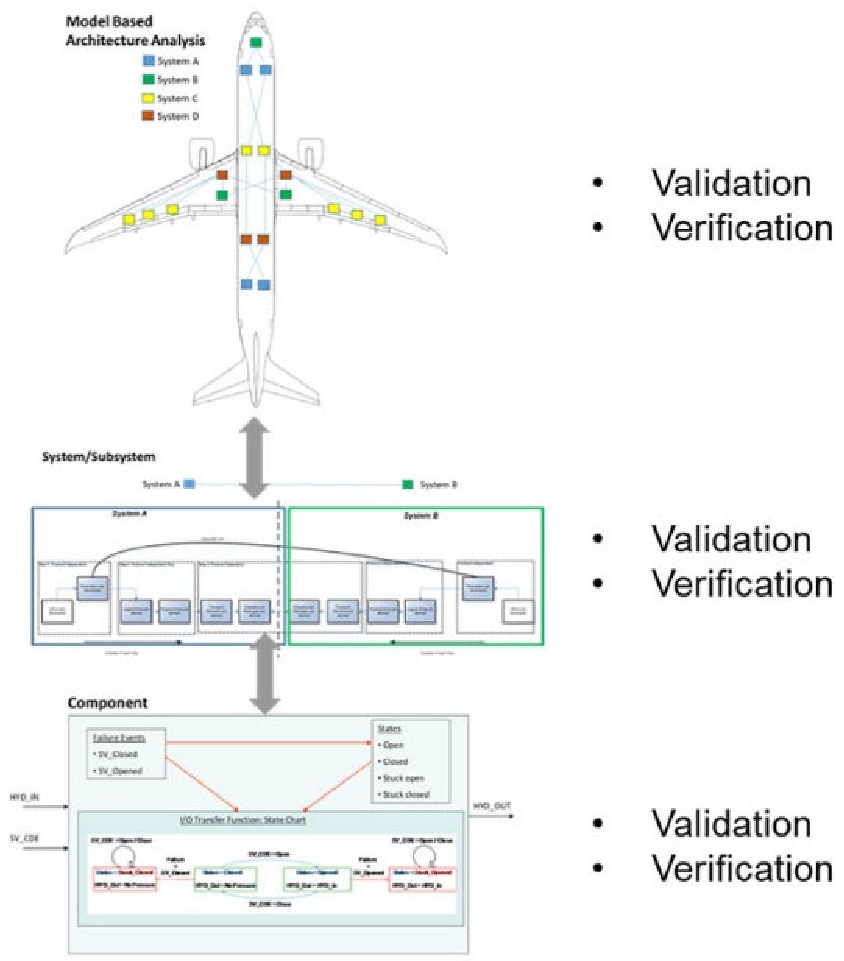


Figure 112. Formalism at different abstraction levels

It is important to clearly identify the required level of modeling fidelity at the different abstraction levels. Models will become more detailed at progressively lower abstraction levels. For example, one is usually not concerned whether a component has a queuing or sampling port when modeling at the airplane/systems architecture level. Formalism will also enable the models to be integrated across the different levels. This can help ensure that changes to lower-level models will not "violate" the higher-level models.

It is axiomatic but it bears mentioning: it is important to know what the model is going to be used for; this will drive the required formalism. Examples of modeling objectives at the architecture level (shown in Figure 113) include:

- Validate functionality and performance.
- Validate that failure conditions identified in the aircraft level FHA have been addressed and that corresponding safety requirements are met.

- Validate that the interactions of system functions, their interdependencies, independence, separation, and their contribution to associated failure conditions have been appropriately identified and assessed.
- Validate Functional Hazard Assessment hazard categorizations.

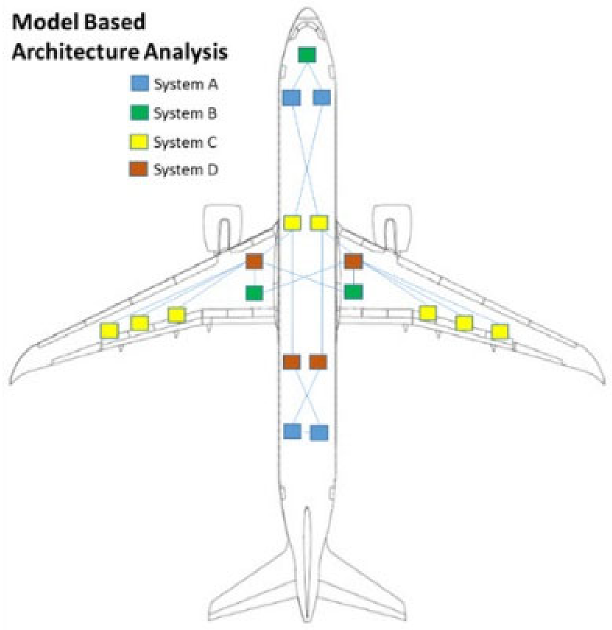


Figure 113. Formalism at the architecture level

Examples of modeling objectives at the system/subsystem level (shown in Figure 114) include:

- Validate completeness, sufficiency, and accuracy of systems architecture.
- Validate that the definitions that compromise your system are complete, accurate, and sufficient.
- Validate that the usages (hardware and software) are complete, accurate, and sufficient. The number of usages is consistent with, for example, the required redundancy management, reliability, and availability.
- Validate that each system is providing what is required by other systems.
- Validate that each system is receiving what is needed (from the correct system).
- Validate that the parameters, logical ports, physical ports, and dataset and messages have passed the audits.

- Validate that system passed any audits conducted by the system that are in addition to the audits published by IMA (ensuring that the data not only satisfies IMA audits, but also satisfies system's requirements).
- Validate design robustness.

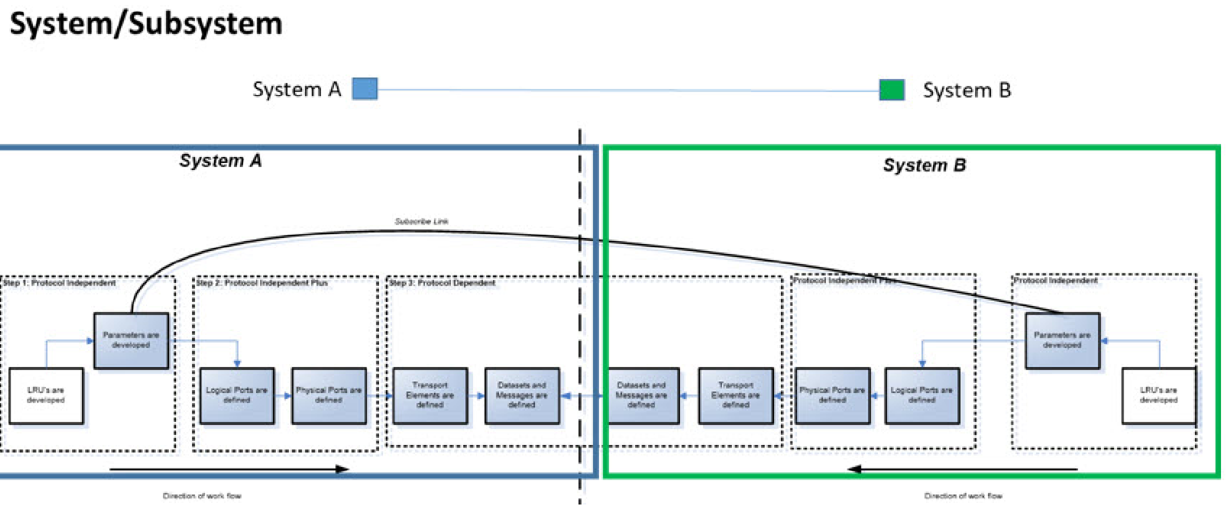

System/Subsystem
System A
System B
System A
System B
Subscribe Link
Step 1: Protocol Independent
Step 2: Protocol Independent Plus
Step 3: Protocol Dependent
Protocol Independent Plus
Protocol Independent
LRU's are developed
Parameters are developed
Logical Ports are defined
Physical Ports are defined
Transport Elements are defined
Datasets and Messages are defined
Datasets and Messages are defined
Transport Elements are defined
Physical Ports are defined
Logical Ports are defined
Parameters are developed
LRU's are developed
Direction of work flow
Direction of work flow

Figure 114. Formalism at system/subsystem level

Examples of modeling objectives at the DIMA level (shown in Figure 115) include:

- Validate system to network platform completeness, sufficiency, and accuracy.
- Host system on a general processing module.
- Validate hosted applications to IMA platform compatibility.
- Complete analyses to ensure services provided by IMA are sufficient for the system.
- General processing module (GPM) allocation.
- Network path definition (timing scenarios).
- Network bandwidth analysis (validate network bandwidth allowances are not violated by the systems).
- Arrange for systems to communicate directly to Common Data Network (CDN).
- Validate applications not hosted on IMA but communicating directly to the CDN.
- Validate end-system configuration files created by IMA are sufficient for the system to access the common data network and fulfill its system/airplane function.
- Arrange communication across CDN via Remote Data Concentrator (RDC).

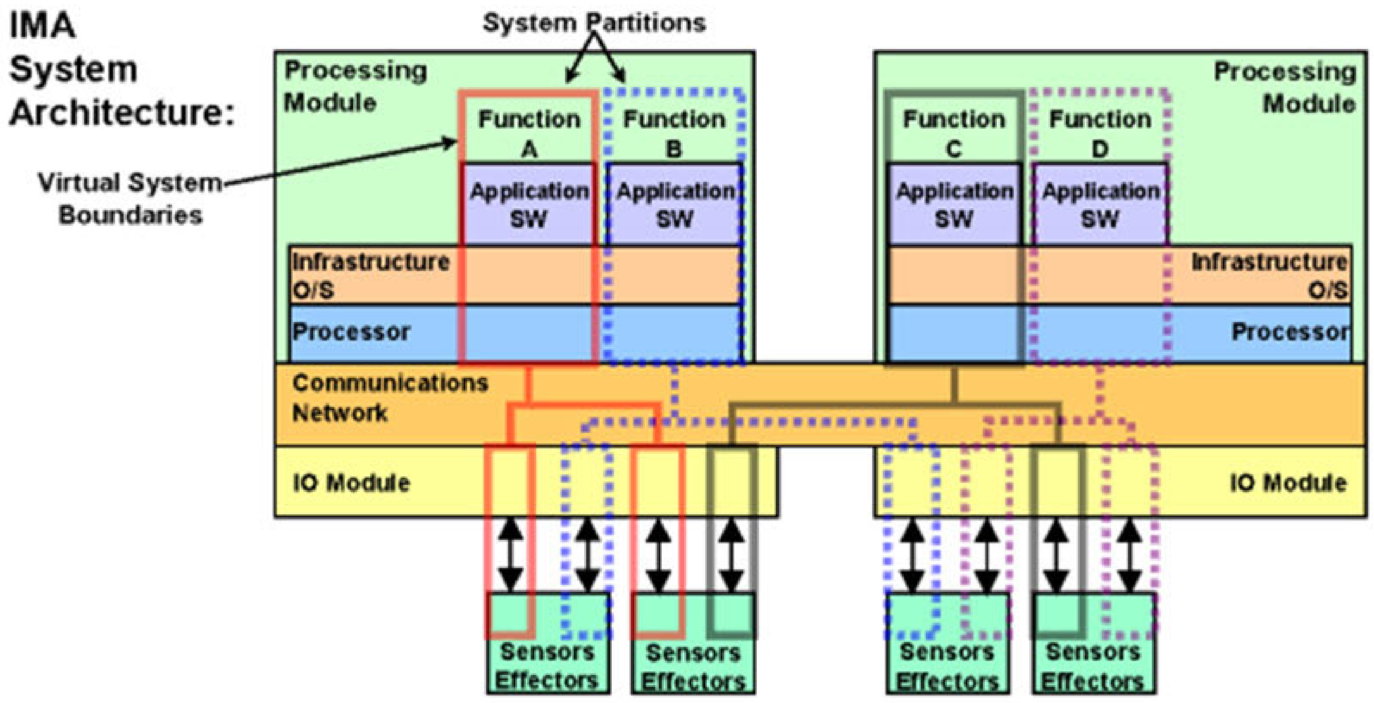


Figure 115. Formalism for network architecture analysis (system to platform)

In an MBE/MBSE environment, it is important to ensure that the models are complete and correct. This is accomplished by a recursive, iterative validation of the models. One of the first steps is to have and enforce a common modeling language and a standard data model. This helps ensure that the models are being created in a consistent manner, which is critical for integration. The standard data model is then constrained by embedded rules that help achieve data integrity. These rules preclude an individual from certain modeling errors, such as linking an input port to an input port (as opposed to an output port).

Query engines can enable data audits and checks. Examples of audit reports could include:

- Ensuring analog discrete parameters and digital output parameters have the required attributes populated.
- Checking the attributes against the data format types for a digital output parameter.

A query can be run to check the relationships between certain attributes for mathematical and functional correctness (e.g., functional range maximum > functional range minimum and < full scaled range – upper bound).

Model-based systems engineering intends to centralize all information about the system in a model, often called the "single source of truth." The models can support the system's entire life cycle, from requirements documentation to validation, and verification exercises to maintenance and training purposes. Different stakeholders (including suppliers) can access the models at different views and levels of detail. This enables the ability to access the data according to their needs, ensuring consistency of the information. Establishing the formalisms at the different abstraction levels is an integral part to enabling the integration of the different models.

7.4 Mitigation of integration and safety issues for process execution

Model-based design approaches, using formalisms, can potentially help mitigate integration and safety issues for process execution problems.

Note: This does not imply that the existing processes are insufficient; this is more about gaining efficiencies and mitigating schedule and cost risks.

The safety analysis starts with assessing the airplane-level functions and system-level functions and evaluating for hazard assessments.

The Functional Hazard Assessment (FHA) is a comprehensive qualitative evaluation of airplane or systems functions, the failure conditions associated with each function, the airplane effects, and hazard classification of those effects. FHAs are performed at the airplane-level (i.e., AFHA) and at the systems level (i.e., SFHA).

The FHAs are foundational for all safety analyses. The FHA results drive safety criteria, system architecture, development assurance levels, HIRF/lightning test levels, and other qualification test criteria.

The following examples show how formalisms can potentially help mitigate integration and safety issues for process execution problems. Form follows function. The first step is to identify "what" needs to be done (i.e., functions). The next step is to identify "how well" (i.e., performance requirements).

A requirement can be allocated to multiple functions; a function can have multiple requirements allocated to it. As shown in Figure 116, a requirements to function allocation matrix can provide a framework for identifying and validating functional requirements, identifying missing requirements, developing derived requirements, identifying unrealistic and conflicting requirements, and eliminating poorly written requirements.

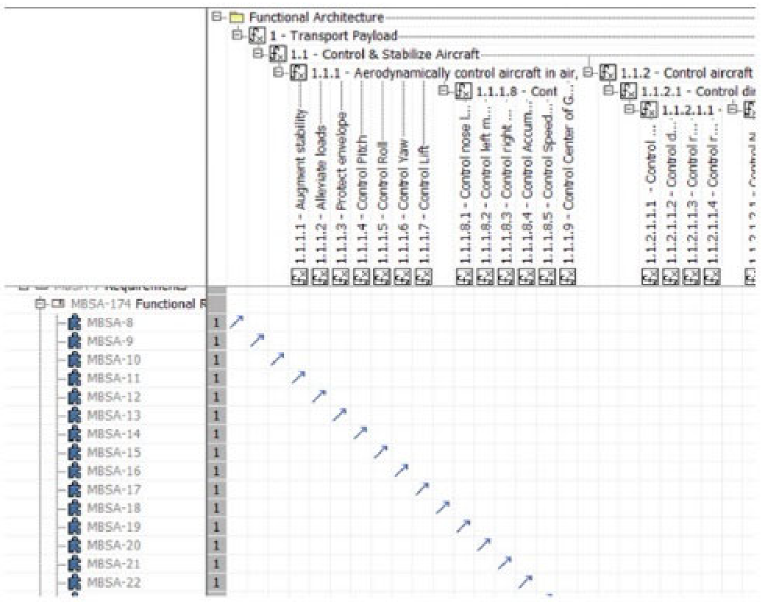
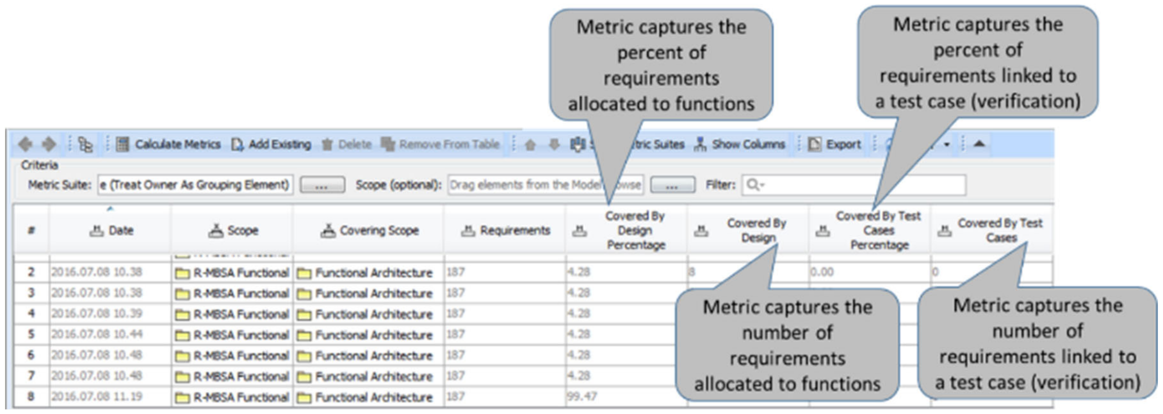

Functional Architecture
1 - Transport Payload
1.1 - Control & Stabilize Aircraft
1.1.1 - Aerodynamically control aircraft in air,
1.1.2 - Control aircraft
1.1.1.8 - Cont
1.1.2.1 - Control di
1.1.2.1.1
1.1.1.1 - Augment stability
1.1.1.2 - Alleviate loads
1.1.1.3 - Protect envelope
1.1.1.4 - Control Pitch
1.1.1.5 - Control Roll
1.1.1.6 - Control Yaw
1.1.1.7 - Control Lift
1.1.1.8.1 - Control nose L...
1.1.1.8.2 - Control left m...
1.1.1.8.3 - Control right ...
1.1.1.8.4 - Control Accum...
1.1.1.8.5 - Control Speed...
1.1.1.9 - Control Center of G...
1.1.2.1.1.1 - Control ...
1.1.2.1.1.2 - Control d...
1.1.2.1.1.3 - Control r...
1.1.2.1.1.4 - Control r...
MBSA-174 Functional R
MBSA-8
MBSA-9
MBSA-10
MBSA-11
MBSA-12
MBSA-13
MBSA-14
MBSA-15
MBSA-16
MBSA-17
MBSA-18
MBSA-19
MBSA-20
MBSA-21
MBSA-22

Figure 116. Requirements allocation to functions

One key aspect of development assurance is to minimize requirements and design errors that could adversely affect safety. Formally documenting the requirements to function allocation can help analyze requirements completeness and correctness (which helps mitigate the risk of requirements errors). The following are some of the analyses and reports that can be generated when formalisms are used:

- Unallocated requirements.
- Unallocated design requirements.
- Unallocated functional and performance requirements.
- Systems without allocated requirements.
- Functions without allocated requirements.
- Requirements allocated to functions.
- Requirements allocated to systems.
- Requirements allocation sheet (logical, functional, and requirements view).

Figure 117 shows an example of allocation metrics, which can help ensure the completeness and correctness of the model.


Metric captures the percent of requirements allocated to functions
Metric captures the percent of requirements linked to a test case (verification)
Metric captures the number of requirements allocated to functions
Metric captures the number of requirements linked to a test case (verification)
Calculate Metrics
Add Existing
Delete
Remove From Table
Show Columns
Export

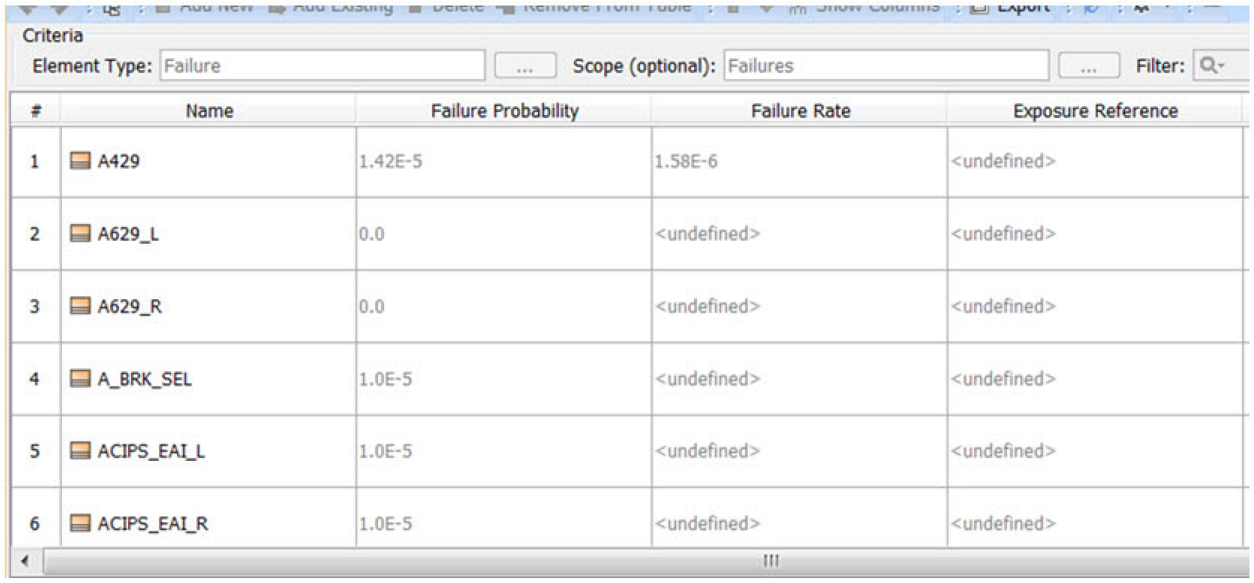
Criteria

Metric Suite: e (Treat Owner As Grouping Element) ... Scope (optional): Drag elements from the Model Browse ... Filter:

#	Date	Scope	Covering Scope	Requirements	Covered By Design Percentage	Covered By Design	Covered By Test Cases Percentage	Covered By Test Cases
2	2016.07.08 10.38	R-MBSA Functional	Functional Architecture	187	4.28	8	0.00	0
3	2016.07.08 10.38	R-MBSA Functional	Functional Architecture	187	4.28			
4	2016.07.08 10.39	R-MBSA Functional	Functional Architecture	187	4.28			
5	2016.07.08 10.44	R-MBSA Functional	Functional Architecture	187	4.28			
6	2016.07.08 10.48	R-MBSA Functional	Functional Architecture	187	4.28			
7	2016.07.08 10.48	R-MBSA Functional	Functional Architecture	187	4.28			
8	2016.07.08 11.19	R-MBSA Functional	Functional Architecture	187	99.47			

Figure 117. Allocation metrics

Figure 118 shows a modeling example of failure probability, failure rate, and exposure reference. This helps create the equipment fault model, which is comprised of all of the components in the systems architecture necessary to represent each subsystem and their fault behaviors to the required level of abstraction. Each component in the model has attributes (e.g., dependency logic, failure rate). This type of modeling can be used to understand how component failures in the system architecture affect their supported functions.



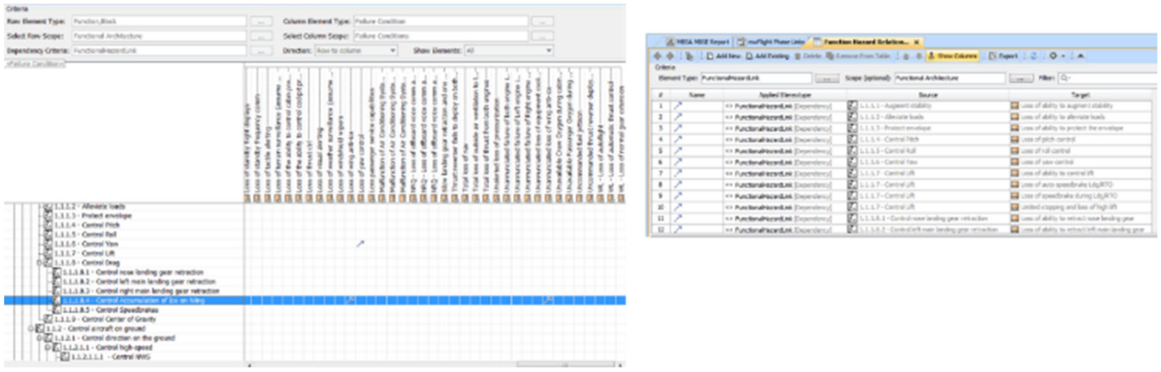
Criteria

Element Type: Failure ... Scope (optional): Failures ... Filter:

#	Name	Failure Probability	Failure Rate	Exposure Reference
1	A429	1.42E-5	1.58E-6	<undefined>
2	A629_L	0.0	<undefined>	<undefined>
3	A629_R	0.0	<undefined>	<undefined>
4	A_BRK_SEL	1.0E-5	<undefined>	<undefined>
5	ACIPS_EAI_L	1.0E-5	<undefined>	<undefined>
6	ACIPS_EAI_R	1.0E-5	<undefined>	<undefined>

Figure 118. Failure probability, failure rate and exposure reference

Figure 119 shows a modeling example of associating functions to FHA. The functional decomposition is comprised of the aircraft and system level functions and their hierarchy. The FHAs describe the loss of certain functions that have an associated severity at the aircraft level. Severity of the loss of function is captured for each flight phase of operation, allowing increased granularity in the analyses to cover scenarios that may only occur at specific times in a flight. The hazards are linked to the function and their resource dependencies.



Criteria

Row Element Type: Function_Block
Select Row Scope: Functional Architecture
Dependency Criteria: FunctionalHazardLink
Column Element Type: Failure Condition
Select Column Scope: Failure Conditions
Direction: Row to column
Show Elements: All

1.1.1.2 - Alleviate loads
1.1.1.3 - Protect envelope
1.1.1.4 - Control Pitch
1.1.1.5 - Control Roll
1.1.1.6 - Control Yaw
1.1.1.7 - Control Lift
1.1.1.8 - Control Drag
1.1.1.8.1 - Control nose landing gear retraction
1.1.1.8.2 - Control left main landing gear retraction
1.1.1.8.3 - Control right main landing gear retraction
1.1.1.8.4 - Control Accumulation of Ice on Wing
1.1.1.8.5 - Control Speedbrakes
1.1.1.9 - Control Center of Gravity
1.1.2 - Control aircraft on ground
1.1.2.1 - Control direction on the ground
1.1.2.1.1 - Control high-speed
1.1.2.1.1.1 - Control NWS

MBSA MBSE Report | myFlight Phase Links | Function Hazard Relation...

Criteria

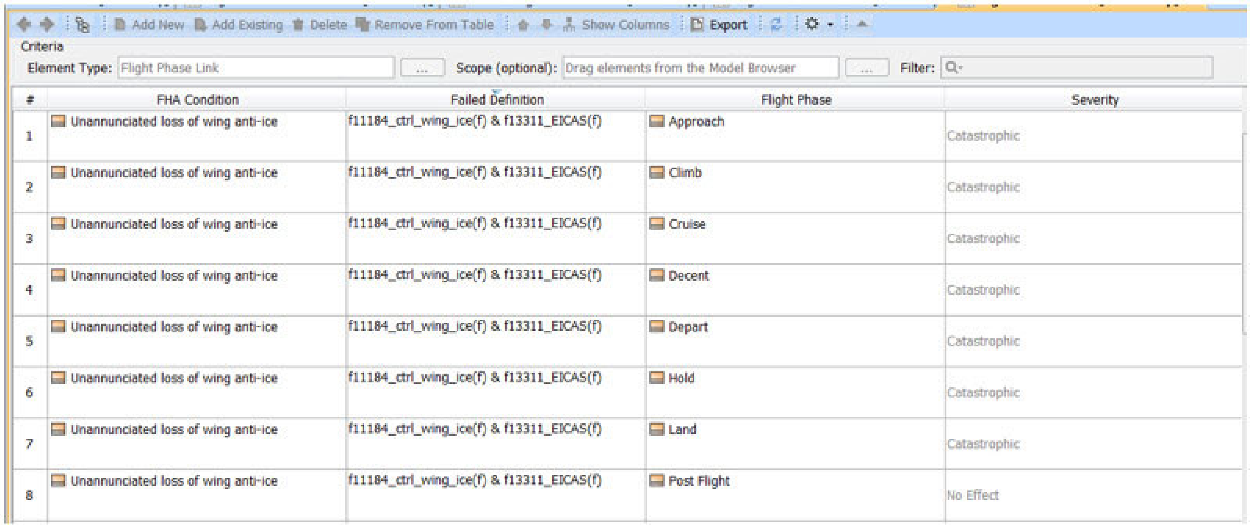
Element Type: FunctionalHazardLink
Scope (optional): Functional Architecture
Filter:

#	Name	Applied Stereotype	Source	Target
1		«» FunctionalHazardLink [Dependency]	1.1.1.1 - Augment stability	Loss of ability to augment stability
2		«» FunctionalHazardLink [Dependency]	1.1.1.2 - Alleviate loads	Loss of ability to alleviate loads
3		«» FunctionalHazardLink [Dependency]	1.1.1.3 - Protect envelope	Loss of ability to protect the envelope
4		«» FunctionalHazardLink [Dependency]	1.1.1.4 - Control Pitch	Loss of pitch control
5		«» FunctionalHazardLink [Dependency]	1.1.1.5 - Control Roll	Loss of roll control
6		«» FunctionalHazardLink [Dependency]	1.1.1.6 - Control Yaw	Loss of yaw control
7		«» FunctionalHazardLink [Dependency]	1.1.1.7 - Control Lift	Loss of ability to control lift
8		«» FunctionalHazardLink [Dependency]	1.1.1.7 - Control Lift	Loss of auto speedbrake Ldg/RTO
9		«» FunctionalHazardLink [Dependency]	1.1.1.7 - Control Lift	Loss of speedbrake during Ldg/RTO
10		«» FunctionalHazardLink [Dependency]	1.1.1.7 - Control Lift	Limited stopping and loss of high lift
11		«» FunctionalHazardLink [Dependency]	1.1.1.8.1 - Control nose landing gear retraction	Loss of ability to retract nose landing gear
12		«» FunctionalHazardLink [Dependency]	1.1.1.8.2 - Control left main landing gear retraction	Loss of ability to retract left main landing gear

Figure 119. Associating functions to FHAs

Creating an N2 diagram and table views (which show the functional hazard links from functions to FHAs) can help identify if there are any missing FHAs.

Figure 120 shows a modeling example of associating the FHAs with flight phases. The hazard categorization may differ, depending on the flight phase. Failure condition effects may vary, depending upon the phase of flight at the time of failure condition occurrence. This type of modeling can be used to help ensure that all flight phases are considered; this can help ensure the completeness and correctness of the FHAs.



Criteria

Element Type: Flight Phase Link
Scope (optional): Drag elements from the Model Browser
Filter:

#	FHA Condition	Failed Definition	Flight Phase	Severity
1	Unannunciated loss of wing anti-ice	f11184_ctrl_wing_ice(f) & f13311_EICAS(f)	Approach	Catastrophic
2	Unannunciated loss of wing anti-ice	f11184_ctrl_wing_ice(f) & f13311_EICAS(f)	Climb	Catastrophic
3	Unannunciated loss of wing anti-ice	f11184_ctrl_wing_ice(f) & f13311_EICAS(f)	Cruise	Catastrophic
4	Unannunciated loss of wing anti-ice	f11184_ctrl_wing_ice(f) & f13311_EICAS(f)	Decent	Catastrophic
5	Unannunciated loss of wing anti-ice	f11184_ctrl_wing_ice(f) & f13311_EICAS(f)	Depart	Catastrophic
6	Unannunciated loss of wing anti-ice	f11184_ctrl_wing_ice(f) & f13311_EICAS(f)	Hold	Catastrophic
7	Unannunciated loss of wing anti-ice	f11184_ctrl_wing_ice(f) & f13311_EICAS(f)	Land	Catastrophic
8	Unannunciated loss of wing anti-ice	f11184_ctrl_wing_ice(f) & f13311_EICAS(f)	Post Flight	No Effect

Figure 120. Flight phases and FHAs

Using the “single source of truth” can help ensure the consistency between the MBSE and MBSA activities. As shown in Figure 121, the MBSE tool can be used to create logic diagrams, which can help explain how the system works in nominal conditions; it can also be used to create safety dependencies, which can help validate that cascading failure effects are acceptable in the presence of failures.

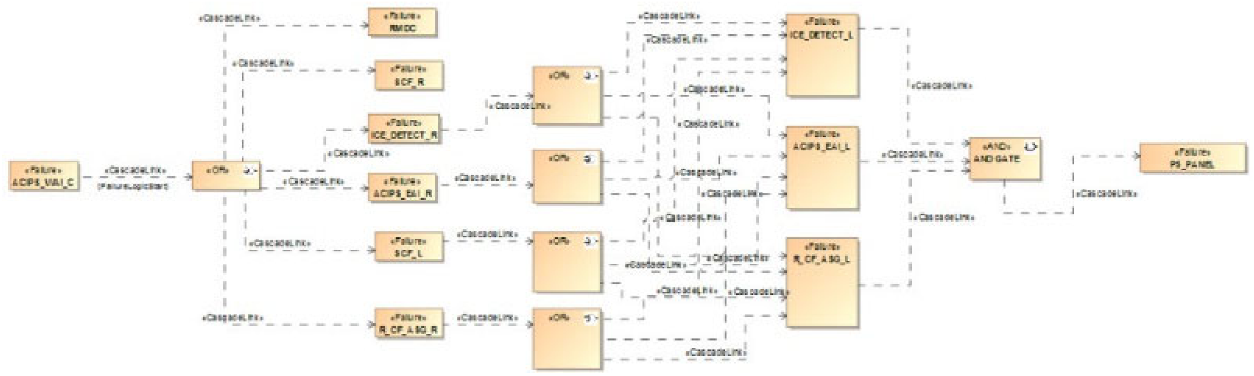


Figure 121. MBSE/MBSA integration

Components are typically procured through contracts imposing specifications. The specifications often include interface definitions. The more “right” the specifications are, the better the components integrate when delivered; this will reduce the cost and schedule impacts of rework. Models (for both MBSE and MBSA) can help enable getting the specifications “right.” Models also enable emulation, which can be beneficial because often not all components are delivered concurrently. This allows earlier integration, which is helpful to identify potential problems.

7.5 Touchpoints with validation and verification processes

Section 6.7 showed the touchpoints between the validation and verification processes.

Some the key elements of MBSE/MBSA involve:

- Modeling and creating the data.
- Validating the data (ensuring completeness and correctness).
- Validating the systems will work together (nominal and failure).
- Developing the software and hardware.
- Simulating and testing.
- Measuring data quality and managing configuration (including updates).

Validation activities need to be conducted after the model has been created. This will help ensure that the data is correct, particularly from an integrated, end-to-end perspective. Both horizontal and vertical integration analyses need to be conducted for validation. These include:

- Component level analyses.
- Intra-system analyses.
- Intersystem analyses.
- Airplane level analyses.

Models can help improve validation activities by supporting multiple views: system to system, subsystem to subsystem, component to component, message to message, parameter to parameter perspective. This helps validate that the models were complete and correct.

With this validation in place, software and hardware can be developed. (As one benefit, with proper procedures in place, software can be autocoded directly from the models.)

With the hardware and software developed, verification can take place at the different levels, such as component, system, or airplane. During this verification, any problem reports should be documented. These problem reports get collected and will be fixed (as needed) in the next blockpoint. Change impact analysis should be conducted as changes are implemented. Due to the iterative nature of design, the process repeats itself.

Section 7.3 discussed the formalism at different abstraction levels. Similarly, validation and verification will occur at the different abstraction levels.

8 Lessons learned/recommendations from MBSE/MBSA implementation

This section describes the criteria, mitigation approaches, recommendations, training material, and position papers. Table 16 describes the approach to address these research objectives.

Table 16. Current guidance and research approach

Research Objectives	**Detailed Work Description**
Develop criteria, mitigation approaches, recommendations, training material, or position papers	Identify the prerequisites/criteria before MBSE and MBSA can be applied. Identify the appropriate levels of abstraction of the design where additional formalism can be applied. Appropriate formalisms must be chosen for each of these target abstraction levels. Once the appropriate formalisms have been selected, it must be possible to obtain the required information, in order to construct the enhanced models at each level.

Research Objectives	Detailed Work Description
	The more complete the formalism, the greater the detail that is required. Identify the limitations/challenges that these prerequisites introduce: Consistency among the models at different levels of abstraction can be both necessary and difficult to obtain. The degree to which the representations of the various models are substantially equivalent can more easily satisfy this need. Assessing the completeness of any particular model can be difficult. This limitation can be eased somewhat to the degree that the models themselves become the requirements for the system. Verifying the correctness of the individual models themselves can be challenging. More formal representations carry more opportunities for guaranteeing at least the consistency of the model.

8.1 Prerequisites/criteria for MBSE and MBSA

Systems engineering and safety have long established processes. The potential benefits of transitioning from a document-focused to a model-focused approach are well understood. It is more beneficial to have a single source of data, which can be used to generate multiple documents. Having a consistent, single source of data improves the efficiency in configuration management and change impact analysis. As shown in Figure 122, in a document-based approach, the same data can be included in multiple documents. When a change is made (for example to "Data 1"), all three of the documents need to be individually updated. A model-based approach precludes the need for synchronization associated with the configuration management of the same data when it is captured in individual documents (each as the "authoritative" source).

A model-focused approach directly improves some of the limitations of a document-based approach, which can impact efficiency. In a document-based approach, each individual document is the authoritative source of data. However, the same data is often duplicated in different documents. This creates the need for data synchronization between the different documents (where the same data is duplicated). In addition to ensuring the data is consistent when the documents are initially created, it is also important to keep the data in the different documents synchronized for subsequent changes.

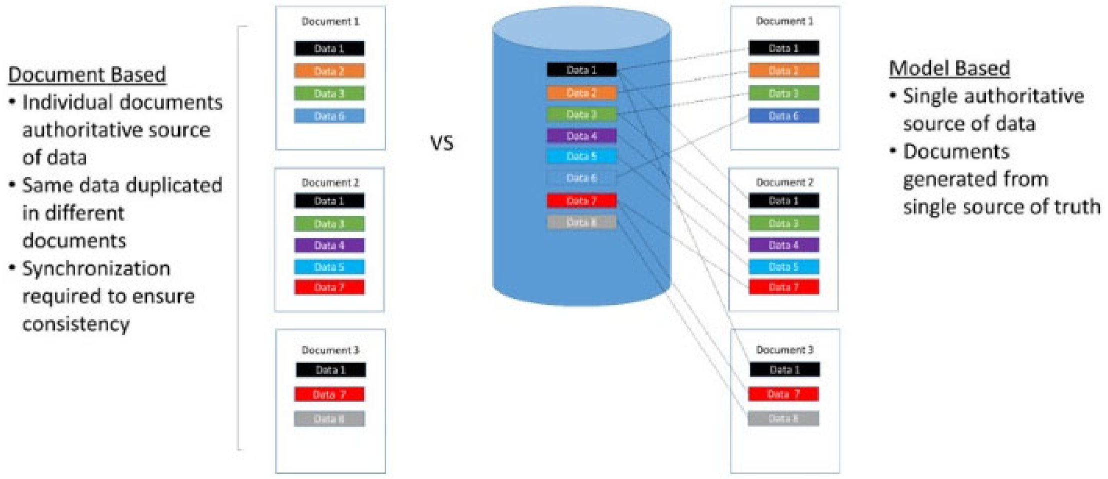

Document Based
• Individual documents authoritative source of data
• Same data duplicated in different documents
• Synchronization required to ensure consistency
Document 1
Data 1
Data 2
Data 3
Data 6
Document 2
Data 1
Data 3
Data 4
Data 5
Data 7
Document 3
Data 1
Data 7
Data 8
VS
Data 1
Data 2
Data 3
Data 4
Data 5
Data 6
Data 7
Data 8
Model Based
• Single authoritative source of data
• Documents generated from single source of truth

Figure 122. Document approach versus model-based approach

Because of the improvements in efficiency resulting from not having to manually synchronize the same data in different documents, industry is moving from static documents to model-based documents.

Models have been used as part of document-based SE approach for many years, and include functional flow diagrams, behavior diagrams, schematic block diagrams, N2charts, performance simulations, and reliability models, to name a few.

However, the use of models has generally been limited in scope to support specific types of analysis or selected aspects of system design. The individual models have not been integrated into a coherent model of the overall system.—
A Practical Guide to SysML. Friedenthal, Moore and Steiner

While modeling is not new, one of the goals of MBSE is to have a shared systems model with discipline-specific models providing their information in a mathematically rigorous format. All disciplines then "view" a consistent system model.

There are certain prerequisites that need to be in place in order to transition from traditional SE and safety to MBSE and MBSA:

- Taxonomy and ontology.
- Tools.
- People/training/culture/implementation.

8.1.1 Taxonomy and ontology

Taxonomies and ontologies standardize terms and the relationships between terms. They are used in every branch of science and engineering. In biology, species are classified by taxonomies, for example. Standard terms, and standard relationships between terms, are necessary for consistent information sharing and interpretation.

Taxonomy is a scheme of classification. Figure 123 shows an example of an MBE taxonomy across the product lifecycle. To achieve the full benefits, modeling can be set up to support complete lifecycle (including production and support and services). The focus of this report is on the airplane vehicle, which is mostly depicted by the blue boxes. The production system (modeling the factory as a system) can also benefit from an MBSE/MBSA approach. Similarly, support products (e.g., maintenance manuals) and services (e.g., predictive reliability) can also be modeled. Having these different models integrated allows for improved optimization (understanding tradeoffs between design, production and support/services). These additional elements are included to show that MBSE/MBSA can be applicable throughout the entire product lifecycle.

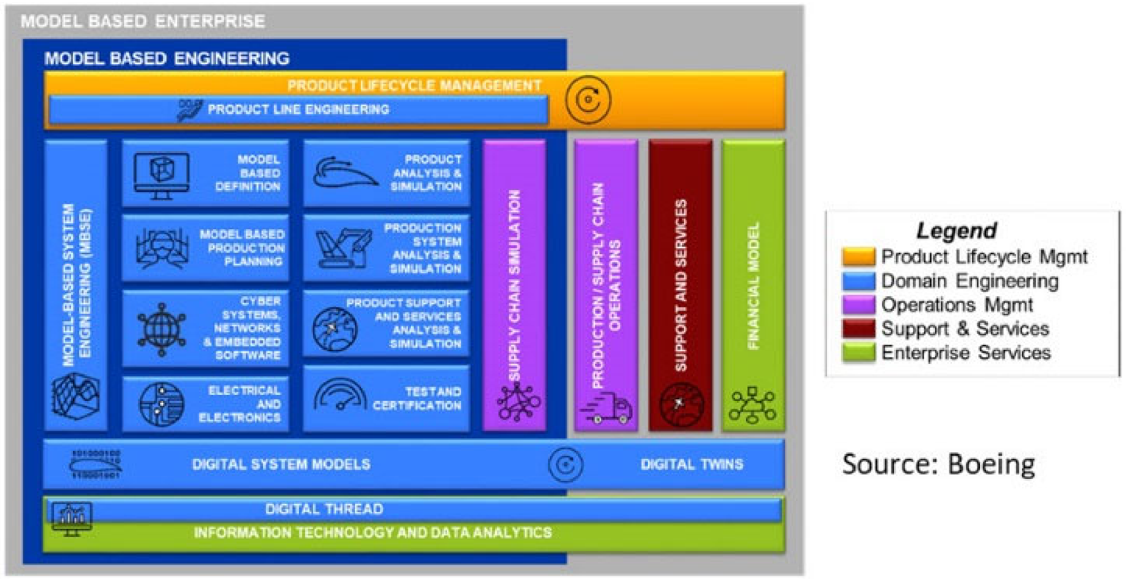

MODEL BASED ENTERPRISE
MODEL BASED ENGINEERING
PRODUCT LIFECYCLE MANAGEMENT
PRODUCT LINE ENGINEERING
MODEL-BASED SYSTEM ENGINEERING (MBSE)
MODEL BASED DEFINITION
PRODUCT ANALYSIS & SIMULATION
MODEL BASED PRODUCTION PLANNING
PRODUCTION SYSTEM ANALYSIS & SIMULATION
CYBER SYSTEMS, NETWORKS & EMBEDDED SOFTWARE
PRODUCT SUPPORT AND SERVICES ANALYSIS & SIMULATION
ELECTRICAL AND ELECTRONICS
TEST AND CERTIFICATION
SUPPLY CHAIN SIMULATION
PRODUCTION / SUPPLY CHAIN OPERATIONS
SUPPORT AND SERVICES
FINANCIAL MODEL
DIGITAL SYSTEM MODELS
DIGITAL TWINS
DIGITAL THREAD
INFORMATION TECHNOLOGY AND DATA ANALYTICS
Legend
Product Lifecycle Mgmt
Domain Engineering
Operations Mgmt
Support & Services
Enterprise Services
Source: Boeing

Figure 123. MBE components (taxonomy)

Ontologies detail the objects, attributes, and relationships describing a domain, akin to a data model. MBSE tools should be architecturally driven by formal data models that ensure data integrity. A data model is a type of ontology, just more concrete and specific.

Data models are essential to enabling digital transformation/digital thread/digital twin activities. The concept of the digital twin is to create a digital informational construct of a physical system as an entity on its own. This digital information would be a “twin” of the information that is embedded within the physical system and be linked with that physical system through the entire lifecycle of the system. The digital twin concept is intended to allow manufacturers to create models of the physical production systems and processes using historical and real-time data from smart sensors for near-real-time analysis and system performance improvement.

Figure 124 shows validation and verification in the context of the SE Diamond. Section 6.2.1 provided detailed information on the data models required to support MBSE and MBSA.

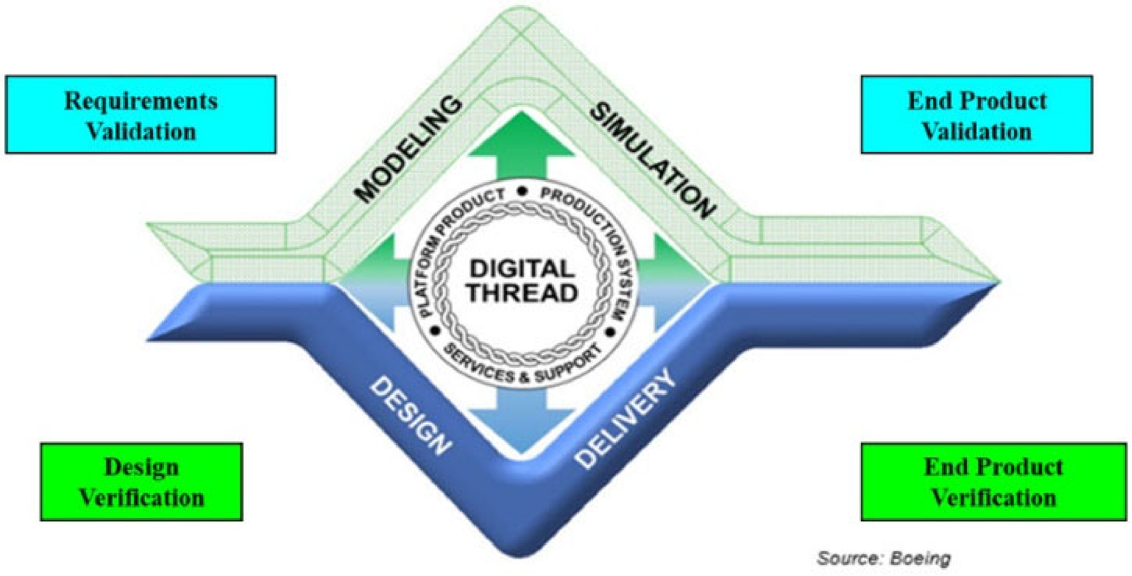


Figure 124. Validation and verification in SE diamond

8.1.2 Tools

Tools are also an integral part of enabling MBSE and MBSA. The tools need to support the following functionality:

- Creating models.
- Storing models.
- Configuration managing models.
- Integrating different models.
- Analyzing models.
- Validating rules and constraints.
- Querying.

- Supporting data analytics.
- Generating documents from models (i.e., autogenerating the required artifacts, certification deliverables, and other documents directly from the models).
- Generating products from models.
- Validating products and model integrity.
- Supporting evolutions of aerospace products, systems, models, support, test, training, for example.

Tools can be used to help with both validation and verification. As shown in Figure 124, validation and verification occur on both sides of the diamond.

Requirements validation consists of ensuring that requirements are correct, complete, and integrated. End-product validation consists of ensuring that the product performs as intended in the operational environment (as judged by the end users). Design verification consists of ensuring that the design meets the requirements. The verification method for design verification is almost always analysis. This is because there is no implemented design at this point to inspect, test, or demonstrate. End-product verification consists of ensuring that the product is built per design and that the product meets the requirements.

There are interrelationships between these different activities. For example, it can be the case that the design verification effort concludes that the requirements are infeasible (for whatever reason). Feasibility is a validation criterion, so in this situation, the actions of that design verification would contribute to update and correct the requirements validation.

Tools can be used to support the validation and verification in the lifecycle and gain a better understanding of a system's behavior, in both nominal and failure modes. One of the goals of MBSE/MBSA is the early identification of undesired and unintended behavior, whether it is the result of a failure, an error, or unanticipated consequences of interactions. The desire to identify these types of emergent behavior will drive the required level of modeling and fidelity. The tools need to support the required level of modeling to help identify these types of emergent behavior.

There can be differences in tool usage at the different abstraction levels, such as system level versus component/item level.

- Systems-level tools support decision making on what the system should be. System-level tools do not directly create something intended to be implemented in the actual system.

- Item-level tools support decision making on how the individual items provide their function in the context of the larger system. The item tool may directly provide the physical implementation.

Because the decisions made at the systems level will guide the development at the item level, it is important that the tools support the ability to integrate the models at different abstraction levels.

Section 8.2 discusses the challenges with integration of different abstraction levels.

8.1.3 People/training/culture/implementation

The transition from a document-based approach to a model-based approach can have cultural, as well as technical challenges.

It is important to acknowledge and address the required upfront investment MBSE and MBSA require. This includes investment in tools, model development, and training.

The transition to a model-based approach may introduce some organizational challenges, including:

- Individuals with the greatest domain knowledge may not be as familiar with modeling techniques.
- Individuals who are the best modelers may not have the greatest domain knowledge.

This can usually be addressed by pairing the different skill sets and having them collaborate. Training is a prerequisite to successful MBSE and MBSA. It is important to have both knowledge of modeling techniques and engineering domain knowledge. Modeling knowledge, in and of itself, will not be adequate.

8.2 Abstraction levels and formalisms

Models should support the development and design process and not just end up documenting the final answer. This means that the models need to support the iterative nature of design of going from higher-level abstraction to lower-level models with more details. Figure 125 shows how the formalism occurs at different abstraction levels (airplane, system, component level).

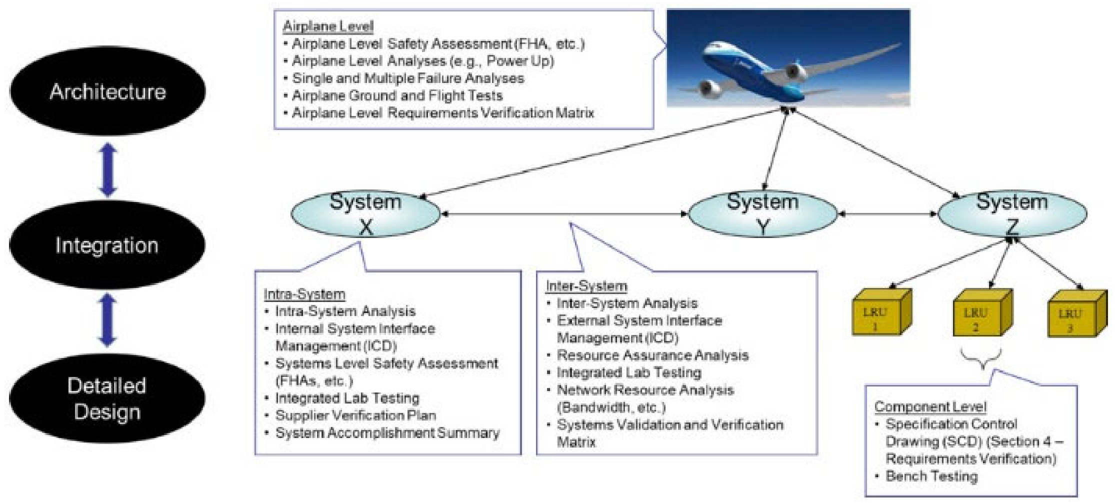

Architecture
Integration
Detailed Design
Airplane Level
• Airplane Level Safety Assessment (FHA, etc.)
• Airplane Level Analyses (e.g., Power Up)
• Single and Multiple Failure Analyses
• Airplane Ground and Flight Tests
• Airplane Level Requirements Verification Matrix
System X
System Y
System Z
Intra-System
• Intra-System Analysis
• Internal System Interface Management (ICD)
• Systems Level Safety Assessment (FHAs, etc.)
• Integrated Lab Testing
• Supplier Verification Plan
• System Accomplishment Summary
Inter-System
• Inter-System Analysis
• External System Interface Management (ICD)
• Resource Assurance Analysis
• Integrated Lab Testing
• Network Resource Analysis (Bandwidth, etc.)
• Systems Validation and Verification Matrix
LRU 1
LRU 2
LRU 3
Component Level
• Specification Control Drawing (SCD) (Section 4 – Requirements Verification)
• Bench Testing

Figure 125. Formalism at different abstraction levels

Formalism needs to exist at each abstraction level (e.g., airplane, system, subsystem). It also needs to exist to enable the integration of the different models (created at different abstraction levels). This includes the ability to support both top-down (higher-level abstractions to lower-level, more detailed models) and bottom-up (lower-level, more detailed models to higher-level abstractions).

8.2.1 MBSE and system model-based design

There are two aspects of MBE that readily apply to product design development: MBSE and System Model-Based Design (SysMBD).

As shown in Figure 126, MBSE can be considered the application of modeling to support system requirements, design, analysis, verification and validation throughout the development lifecycle. System Model-Based Design (SysMBD) is the mathematical representation of design functions, behavior, and software interactions. It is important to be able to integrate the MBSE and SysMBD models and approaches.

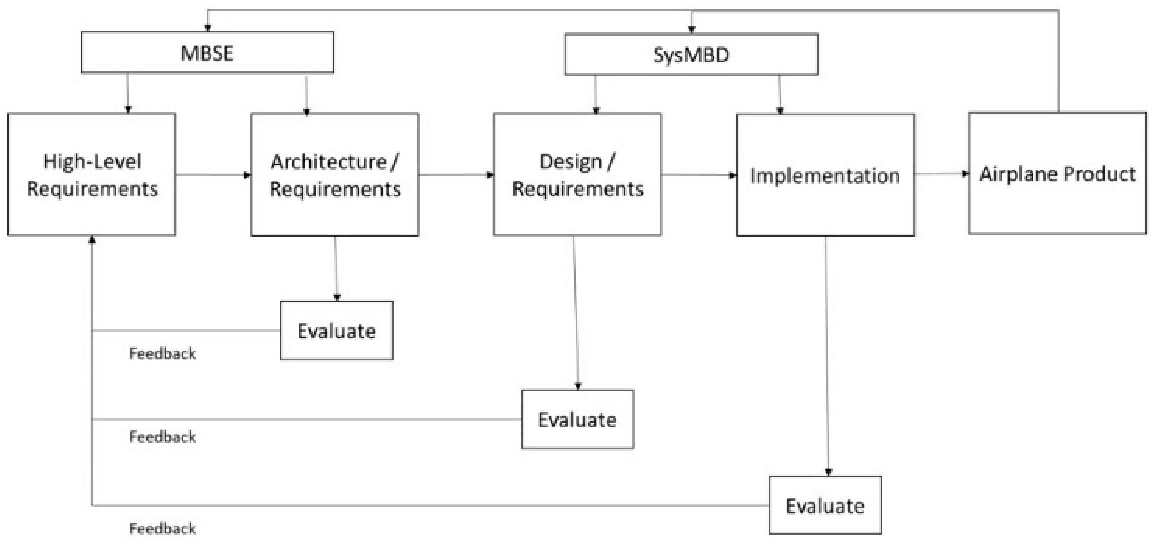


Figure 126. Integration of MBSE and SysMBD

While MBSE and SysMBD are independent, to increase effectiveness, they should be interactive. The combination of both strategies enable better traceability across the design, more robust validation methods, and reuse of the models for various purposes.

MBSE provides a model-based approach to the SE discipline. MBSE enables early architecture development. MBSE excels in capturing descriptive models of the architecture aspects of a system that can be statically analyzed through its traceability.

SysMBD provides a model-based approach to rapidly prototype and test different functional/behavioral designs and validate intended system behavior. SysMBD is used to define functionality later in the development cycle (after the architecture development). It excels in capturing details of system functionality and interfaces that can be dynamically analyzed in simulations.

DO-331 [6], which is a supplement to DO-178, discusses the use of model-based development and verification in the software lifecycle for software that is produced in accordance with DO-178C. It is also applicable to the models, developed in the system process, that define the software requirements or software architecture. DO-331 identifies two types of models, which are mutually exclusive:

- Specification models.
- Design models.

Specification models "represent high-level requirements that provide an abstract representation of functional, performance, interface or safety characteristics of software components" [6].

Design models “include low-level requirements and/or architecture” [6]. Unlike a specification model, a design model does define detailed software design, such as software component internal data structures, data flow, and/or control flow.

Figure 127 shows the interactions between lower-level design models, which are part of DO-331 [6], and analysis (evaluation models), which are not. The models developed for simulation and analysis as part of verification process are not in scope of DO-331. Analysis models represent system functionality that is conveyed in textual requirements. Simulation models are design or analysis models that are used to support a simulation with another design or analysis model.

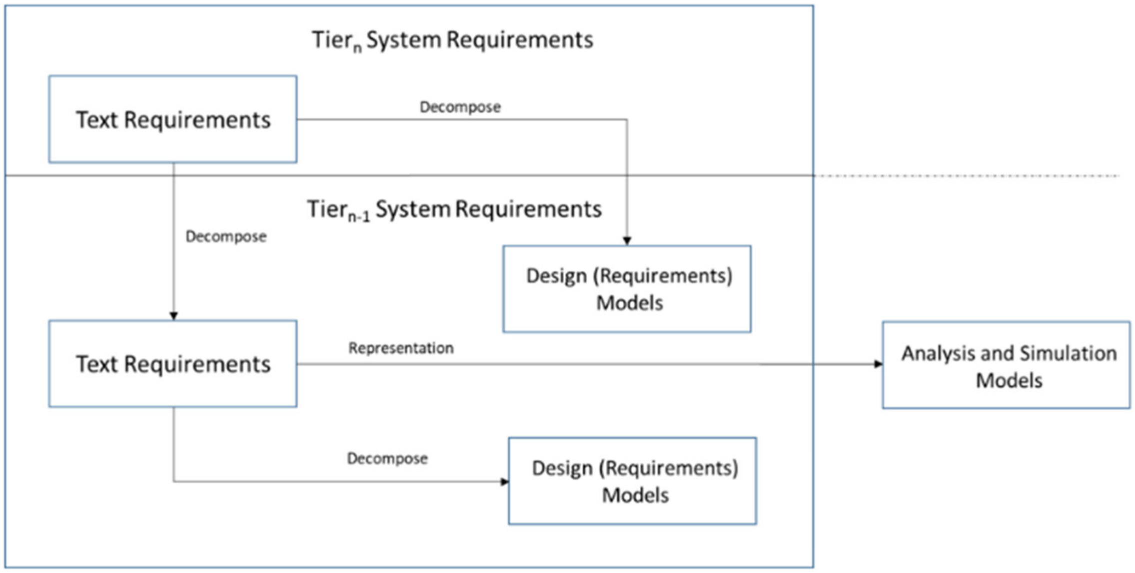

Tier$_n$ System Requirements
Text Requirements
Decompose
Tier$_{n-1}$ System Requirements
Decompose
Design (Requirements) Models
Text Requirements
Representation
Analysis and Simulation Models
Decompose
Design (Requirements) Models

Figure 127. SysMBD models

A design model directly conveys a subset of the intended functional and behavioral requirements of the system. Design models often represent the control and monitoring aspects of the system, and they are often allocated to software and/or AEH components of the system. A set of traditional requirements should exist for the design model. Additionally, requirements-based tests (RBTs) written against the requirements for the design model and executed on the design model in simulation can provide early validation of those requirements, as well as verification of the model, when model coverage metrics are captured. Traditional requirement types that are good candidates to be converted to design models include state charts, logic diagrams, textual logic statements, and other requirements that convey concise and specific behaviors.

An analysis model represents a whole or partial system and potentially the system’s environment, which may be used to validate the technical correctness of textual behavioral or performance requirements or verify that the system meets its requirements. An analysis model may represent requirements to support verification or validation activity, but does not convey requirements. They are often mathematical in nature and executable through simulation, representing the behavioral aspects of physics-based domains of the system. Analysis models must be validated to ensure their fitness to support the purposes for which they were created (e.g., correlation to actual performance of the system or environmental characteristics, which the model represents). An example of an analysis model is a control surface actuator model that is used to help refine and validate textual requirements defining specification characteristics for the actuator.

A simulation support model provides an accurate representation of the portion of the environment or system that behaviorally interacts with a design or analysis model in simulation to support the usage of the design or analysis model; it is used to convey, verify, or validate any requirements it represents in the context it is used. A mature and complete design or analysis model in one context may be reused as a simulation support model in another context.

8.2.2 Formalism at different levels

Once the appropriate formalisms have been selected, it must be possible to obtain the required information in order to construct the enhanced models at each level. The more complete the formalism, the greater the detail that is required.

Figure 128 shows the different abstraction levels, each requiring formalism. It is a slightly different view of the model discussion in the preceding sections. Specification models can be at different levels, such as airplane and systems levels. Design models can be generated during software development. The analysis and simulation models, coupled with testing, may drive change requests, which would be part of the configuration management process.

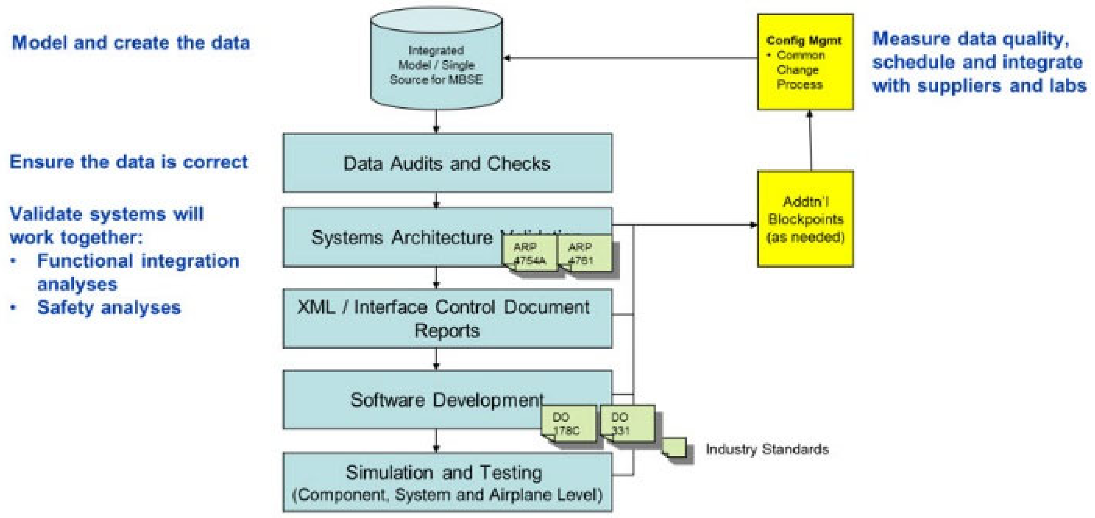

Model and create the data
Integrated Model / Single Source for MBSE
Config Mgmt + Common Change Process
Measure data quality, schedule and integrate with suppliers and labs
Ensure the data is correct
Data Audits and Checks
Validate systems will work together:
• Functional integration analyses
• Safety analyses
Systems Architecture Validation
ARP 4754A
ARP 4761
Addtn'l Blockpoints (as needed)
XML / Interface Control Document Reports
Software Development
DO 178C
DO 331
Industry Standards
Simulation and Testing (Component, System and Airplane Level)

Figure 128. Formalism at different levels

MBSE tools should be architecturally driven by formal data models that ensure data integrity. Data audits and checks can be conducted to ensure that the model is correct. This level of formalism needs to be adjusted for each abstraction level. For example, the level of modeling required to support PASA evaluations does not need to be as detailed as the SysMBD models for dynamic, functional behavior. However, one would want to be able to create a component once – and then use it in different contexts (e.g., PASA MBSA, SysMBD detailed model).

8.3 Limitations/challenges of formalism prerequisites

The prerequisites for MBSE and MBSA introduce limitations or challenges, including:

- Consistency among the models at different levels of abstraction can be both necessary and difficult to obtain. To the degree that the representations of the various models are substantially equivalent, this need can be more easily satisfied.
- Assessing the completeness of any particular model can be difficult. This limitation can be eased somewhat to the degree that the models themselves become the requirements for the system.
- Verifying the correctness of the individual models themselves can be challenging. More formal representations carry more opportunities for guaranteeing at least the consistency of the model.

8.3.1 Model consistency at different abstraction levels

One benefit of MBSE is achieved if the models are consistent and can be used downstream (i.e., lower level, more detailed models) without being recreated.

Figure 129 shows the importance of maintaining consistency between different abstraction levels. The MBSE architecture can include data input/output, connectivity, and requirements allocations. The SysMBD can include functional behavior, performance parametric analysis, and code generation (from design models).

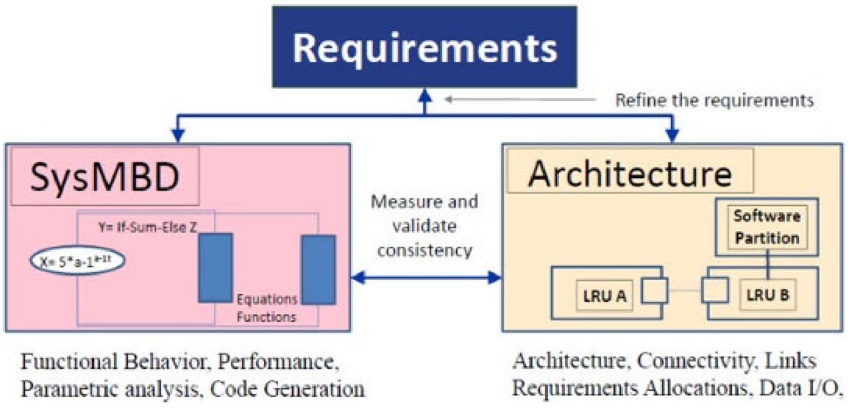

Requirements
Refine the requirements
SysMBD
Y= If-Sum-Else Z
X= 5*a-1
Equations
Functions
Measure and validate consistency
Architecture
Software Partition
LRU A
LRU B
Functional Behavior, Performance, Parametric analysis, Code Generation
Architecture, Connectivity, Links Requirements Allocations, Data I/O,

Figure 129. Consistency between SysMBD and architecture

To avoid having to recreate the MBSE and SysMBD models, it is important to be able (within the capability supported by tools and processes) to have the same modeled object used for different contexts. This includes having the ability to both:

- Create usages of new objects with the same base attributes.
- Allow the new objects to have unique properties (e.g., values, locations, parameters).

This eliminates the need to have to recreate the same base object. It also allows the models to differentiate (if needed).

The definition/usage/occurrence terminology, which is shown in Figure 130, will be used to illustrate the following example: an airplane with a left and right cabinet for its IMA/DIMA.

Note: The terminology is not meant to imply industry consensus. It is being used to convey the concepts, which are universally applicable.

These concepts will be described in further detail, but the high-level explanation is as follows:

- A usage is an instantiation of a definition.
- An occurrence is an instantiation of a usage.

Figure 130 shows an example of a left cabinet in an IMA airplane consisting of the following:

- Fiber Optic Translator (FOX).
- Graphic Generator (GG).
- Power Conditioning Module (PCM).
- Remote Data Concentrator (RDC).

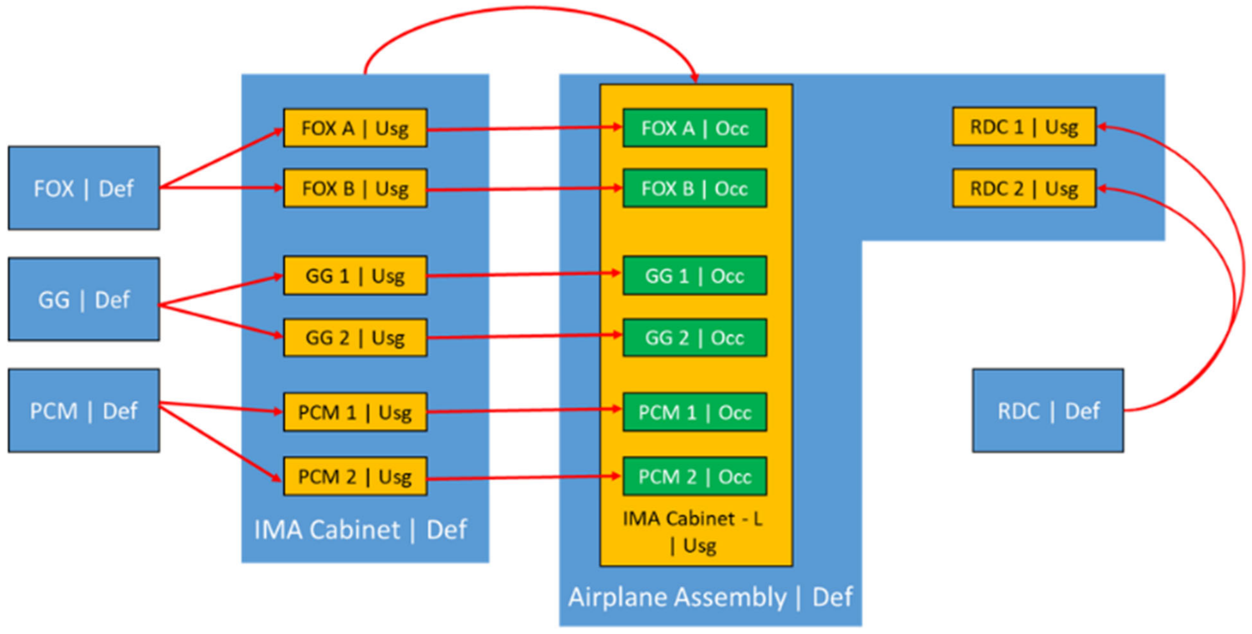


Figure 130. Definition/usage/occurrence in IMA airplane

A system definition is not an actual LRU or software application. A system definition provides a template, which can be realized when it is installed on the airplane. The system definition provides a template for common analog and digital interfaces, including:

- Common electrical characteristics.
- Common connectors and pins.
- Common transmitted and received digital data.

An example of a systems definition is a High Frequency (HF) radio. It is a mechanism to allow creation and reuse of design data.

A system usage is an LRU (or software application) installed on the airplane at a specific location (such as left, center, right). The interface characteristics and connection allocations are inherited from the definition. A usage is a component created, based on an existing definition. It can be used to build the definition of a new nested system/assembly. Each installed usage is wired independently to create system interfaces. An example of a usage is the Left HF Radio.

A system occurrence could be considered to be the usage of a usage. When a usage of a system is created on the airplane that is based upon a system definition with nested component usages, the nested component usages occur inside the installed system usage and are called “occurrences.” For example, when the module is installed on the airplane as a usage, the two internal processors become occurrences.

This is meant to be illustrative and not a comprehensive modeling of an IMA cabinet on an airplane. It is important to emphasize that an airplane can have both usages and occurrences.

The occurrence model allows users to construct assemblies and reuse data. This provides significant time savings, as compared to managing multiple system instantiations separately. When a definition is used, any of its child usages will also propagate to occurrences.

When the system designer changes the definition, the tool should automatically propagate the changes to all usages. In this example, there were two usages on the airplane (a left and right cabin usage). However, systems could have thousands of children that would benefit from a single edit and propagation to its usages. This is particularly true for parameters used in detailed interfaces.

8.3.2 Assessing model completeness (validation)

Peer reviews of the models should be used to validate that the models are conveying the correct intended system functional behavior and that the textual requirements for the model encompass constraints on the model. While conducting peer reviews, the following are examples of questions/criteria that can help ensure the correctness and completeness of the model and its textual requirements:

- Does the model accurately capture the behavior to support its requirements?
- Do the requirements sufficiently guide how this function should be modeled, or are requirement updates needed?
- Are the assumptions regarding the external system functions correct and documented in the requirement set?
- Are all parameters that are relevant to the behavior of the model exposed in the graphical representation of the model?
- Are the performance requirements identified (e.g., rate, loads, stability, gains, accuracy)?

- What should the function do if the input signals are unavailable, invalid, or corrupted?
- Are the constraints identified (e.g., availability, integrity, prohibited behavior)?
- Is the model functionally integrated (e.g., input/output signals, range, accuracy, rate)?
- Does the model accurately depict required robustness/sensitivity (e.g., expected and/or critical levels and combinations of input bias, jitter, noise, oscillation, skew, drift, asymmetry)?

Figure 131 shows a model-based development lifecycle process flow. It is important to recognize the feedback loops that occur throughout the development. After the models have been developed and validated, it is important to update models (as needed), based on feedback from simulation or testing.

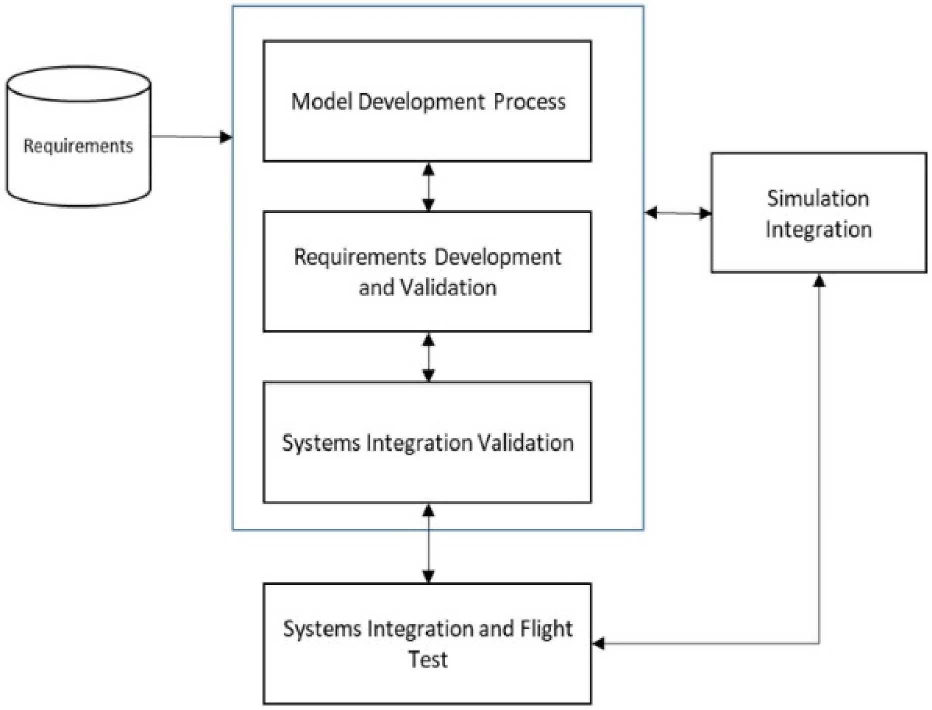

Requirements
Model Development Process
Requirements Development and Validation
Systems Integration Validation
Simulation Integration
Systems Integration and Flight Test

Figure 131. Model-based development lifecycle process flow

The subsequent section will discuss how to ensure model consistency during the iterative design process.

8.3.3 Assessing model consistency

Occurrence modeling can help ensure consistency of models across different abstraction levels. As shown in Figure 132, the occurrence modeling can be used for many types of definitions, such as LRUs, software, ports, parameters, messages, and transport elements.

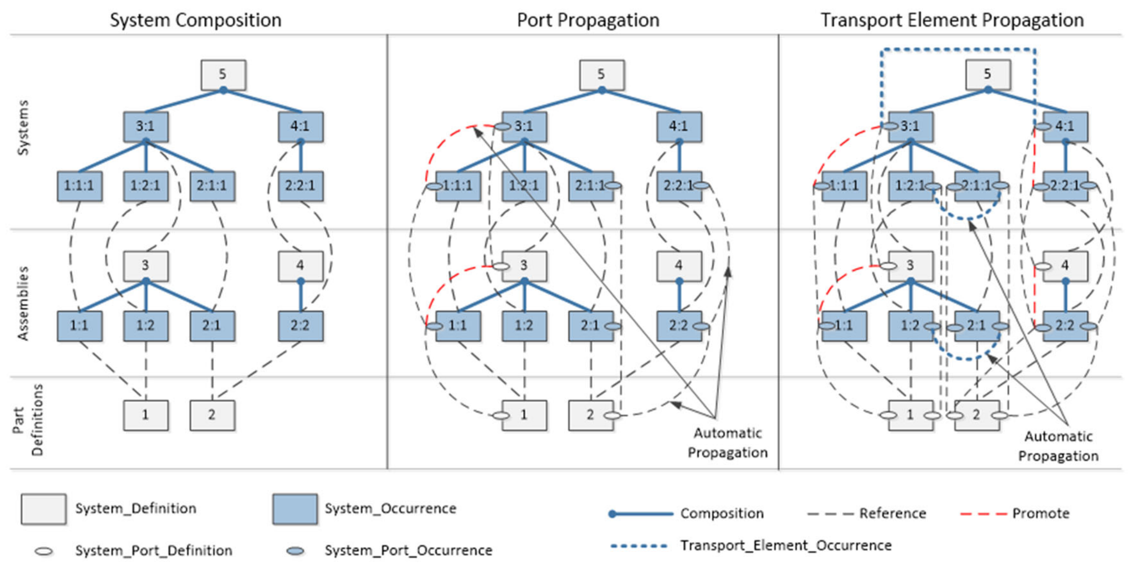


Figure 132. Occurrence modeling

Figure 132 demonstrates the concept of definition and occurrence modeling. (To simplify terminology, "occurrence" will be used interchangeably with "usage.") To understand these concepts, it is best to start at the bottom with the definition. This corresponds with how the models are actually built. One starts with definitions – and then creates occurrences.

Figure 132 shows there are two part definitions: 1 and 2. The definition of 1 used to create occurrences 1.1 and 1.2 (which could be left and right). The definition of 2 is using occurrences 2.1 and 2.2.

In this example, there are two assembly definitions: 3 and 4. The definition of assembly 3 is composed of the usages of 1.1, 1.2, and 2.1. The definition of assembly 4 is composed of the usage of 2.2.

The assemblies need to be put on the airplane. In this example, there is one occurrence of the 3 assembly definition: 3.1. There is one occurrence of the 4 assembly definition: 4.1.

The airplane definition 5, in this simplified example, is comprised of the occurrences of the 3.1 assembly definition and the 4.1 assembly definition.

This same approach can be used for other elements of the model, such as physical ports. To keep the example straightforward, an input physical port will be added to the 1 definition. An output physical port will be added to the 2 definition. When the occurrences of the 1 part definitions is created, the input physical ports will automatically be created on 1.1 and 1.2. When the occurrences on the 2 part are created, the output physical ports will automatically be created on 2.1 and 2.2.

When the occurrences of the 3 assembly definition are created, the physical ports for 1.1, 1.2 and 2.1 will automatically propagate. When the occurrence of the 4 assembly is created, the physical port of 2.2 will automatically propagate.

Figure 132 shows the promote link form occurrences to definitions. This same approach can be extended to transport elements.

This concept is very powerful and needed for the MBSE/MBSA of any large-scale project. If one changed an attribute on the input physical port, it would automatically propagate throughout the definition and occurrence models. This might not seem as powerful with this simple example, but given that some models will consist of millions of objects, this becomes an almost mandatory capability.

Modeling the transport elements establishes the transmission path. Once this has been done, the consistency (discrepancy checking) component of interface control can then be established, as depicted in Figure 133.

In this example, there are three bus usages, which are used to transmit and receive data. A publish link can be created between Parameter 1 – Usage 1 on LRU 1 –Usage 1 and Parameter 1 – Usage 1 on LRU 2 – Usage 1. A publish link can also be created between Parameter 1 – Usage 2 on LRU 1 –Usage 2 and Parameter 1 – Usage 1 on LRU 2 – Usage 1. With the transmission path established, it is possible to check for discrepancies (e.g., timing).

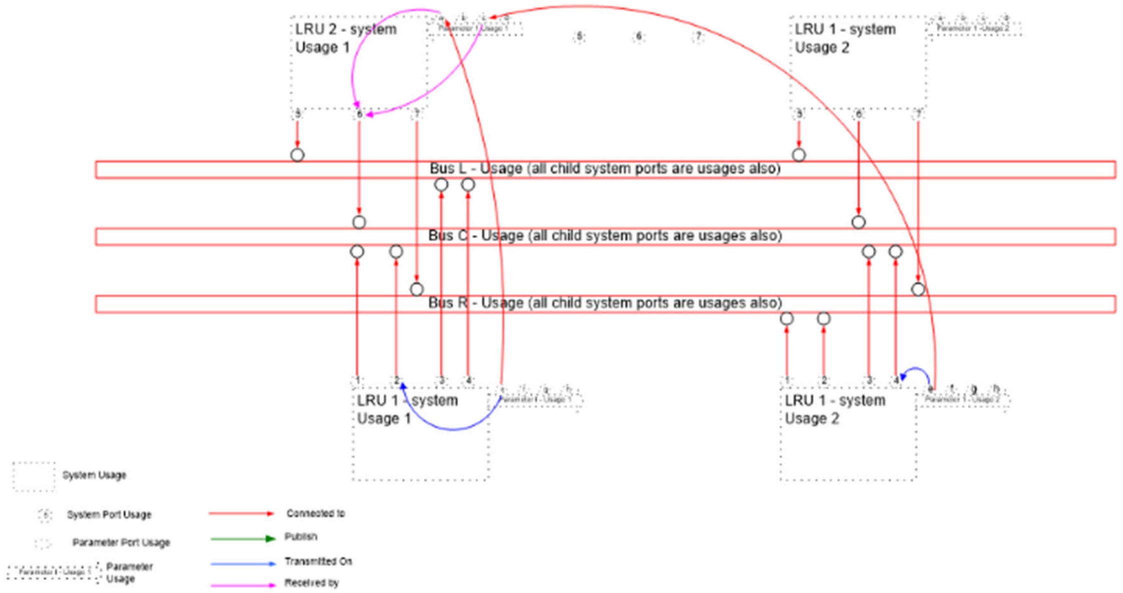
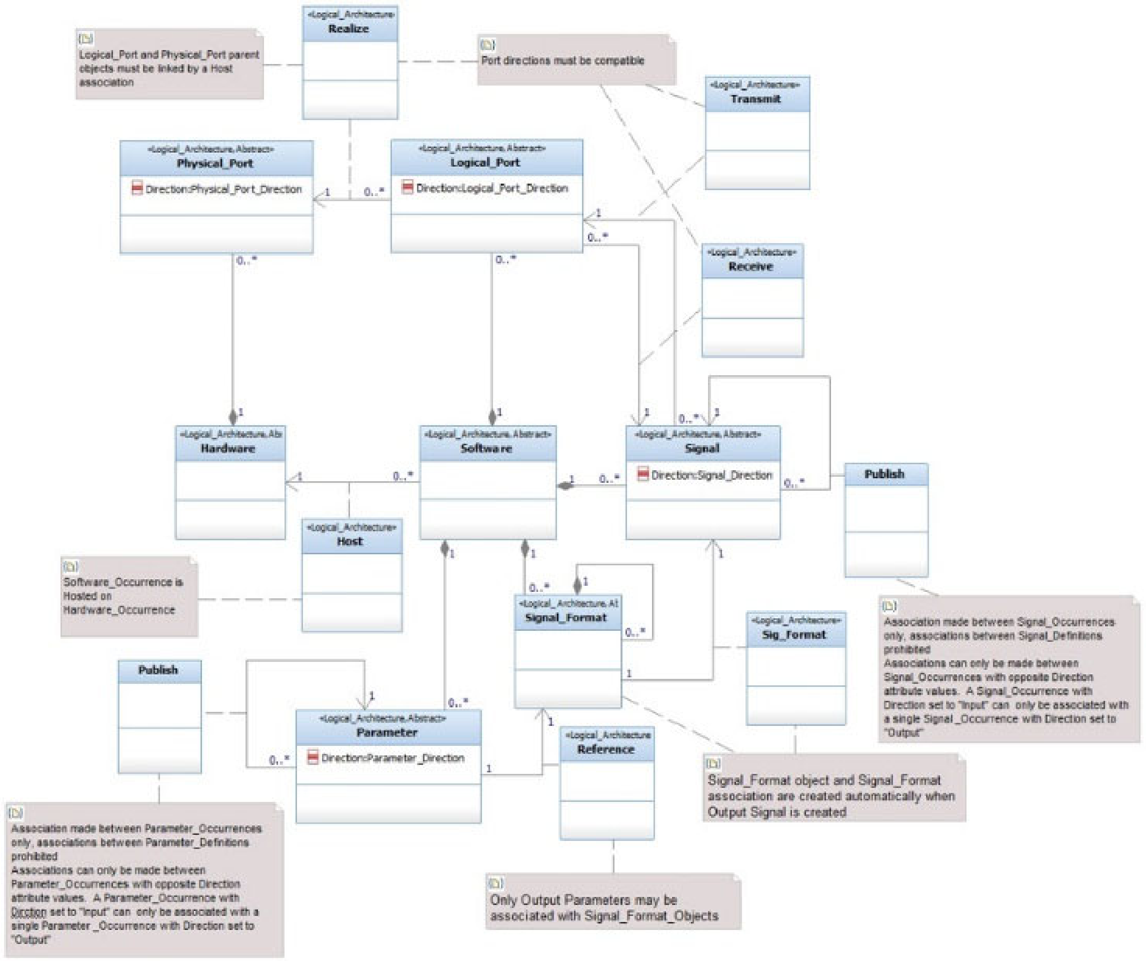


Figure 133. . Consistency (discrepancy checking)

Figure 134 shows a data model that can be used for occurrence modeling for signals, logical ports, physical ports, parameters, software, and hardware. This data model supports the creation of models for the transmission and receipt of signals, which contain parameters. The key elements include software, hardware with logical and physical ports, signals (comprised of parameters in the appropriate signal format), and transmit and receive links.

This data model will be used in subsequent examples to show how it supports the formalisms at the different abstraction levels, enabling the ability to check for model consistency.



Logical_Port and Physical_Port parent objects must be linked by a Host association
«Logical_Architecture» Realize
Port directions must be compatible
«Logical_Architecture» Transmit
«Logical_Architecture.Abstract» Physical_Port
Direction:Physical_Port_Direction
«Logical_Architecture.Abstract» Logical_Port
Direction:Logical_Port_Direction
«Logical_Architecture» Receive
«Logical_Architecture.Abs» Hardware
«Logical_Architecture.Abstract» Software
«Logical_Architecture.Abstract» Signal
Direction:Signal_Direction
Publish
«Logical_Architecture» Host
Software_Occurrence is Hosted on Hardware_Occurrence
«Logical_Architecture.At» Signal_Format
«Logical_Architecture» Sig_Format
Association made between Signal_Occurrences only, associations between Signal_Definitions prohibited
Associations can only be made between Signal_Occurrences with opposite Direction attribute values. A Signal_Occurrence with Direction set to "Input" can only be associated with a single Signal _Occurrence with Direction set to "Output"
Publish
«Logical_Architecture.Abstract» Parameter
Direction:Parameter_Direction
«Logical_Architecture» Reference
Signal_Format object and Signal_Format association are created automatically when Output Signal is created
Association made between Parameter_Occurrences only, associations between Parameter_Definitions prohibited
Associations can only be made between Parameter_Occurrences with opposite Direction attribute values. A Parameter_Occurrence with Dirction set to "Input" can only be associated with a single Parameter _Occurrence with Direction set to "Output"
Only Output Parameters may be associated with Signal_Format_Objects
1
0..*

Figure 134. Data model for occurrences

The focus of this section is to highlight how the definition/usage/occurrence data model can help with ensuring modeling consistency across different abstraction levels. For simplicity, Figure 135 will start with some of the key modeling building blocks already completed. The system definition and port definitions are modeled. (System objects communicate through system ports.) In this example, an ARINC 429 physical port is modeled on Part Definition 1. An ARINC 429 and an Ethernet physical port are modeled on Part Definition 2.

LRU Definition 1 is created, with the occurrences of Part 1 Definition and Part 2 Definition. When the occurrence are created, the objects and relationships propagate. This capability helps ensure model consistency between the definition and the assembly levels. For example, the Ethernet Port Part Definition on Part 2 is propagated when the Part 2 occurrence is created.

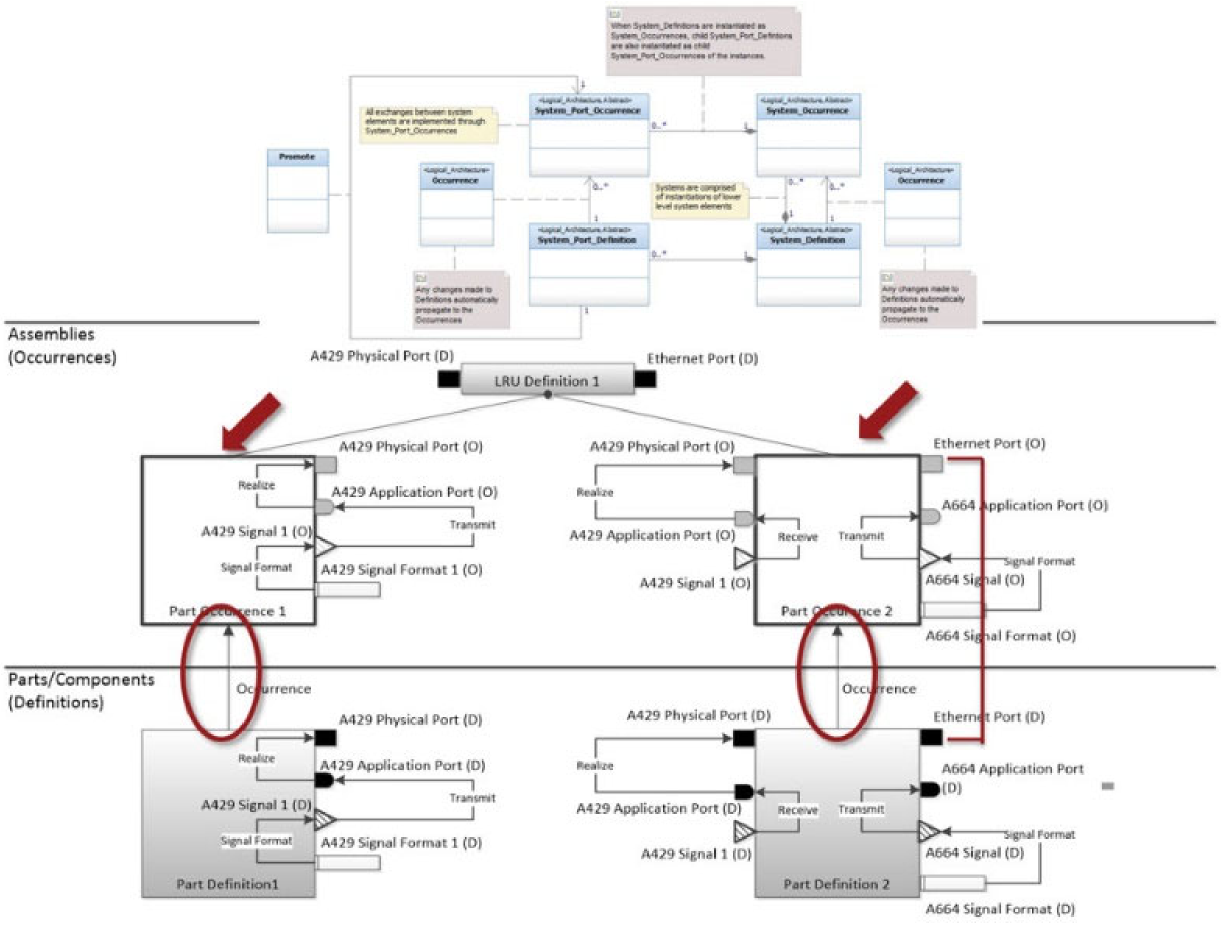


Figure 135. Definition/occurrence propagation

Figure 136 shows the installation of assemblies on an airplane, which is the top-level product definition. For simplicity, the airplane consists of two installed assemblies. The airplane product definition is created, with the occurrences of Assembly 1 and 2. When the occurrences are created, the objects and relationships propagate. This is a fundamental capability to help ensure model consistency between the assembly levels and the airplane-level definition.

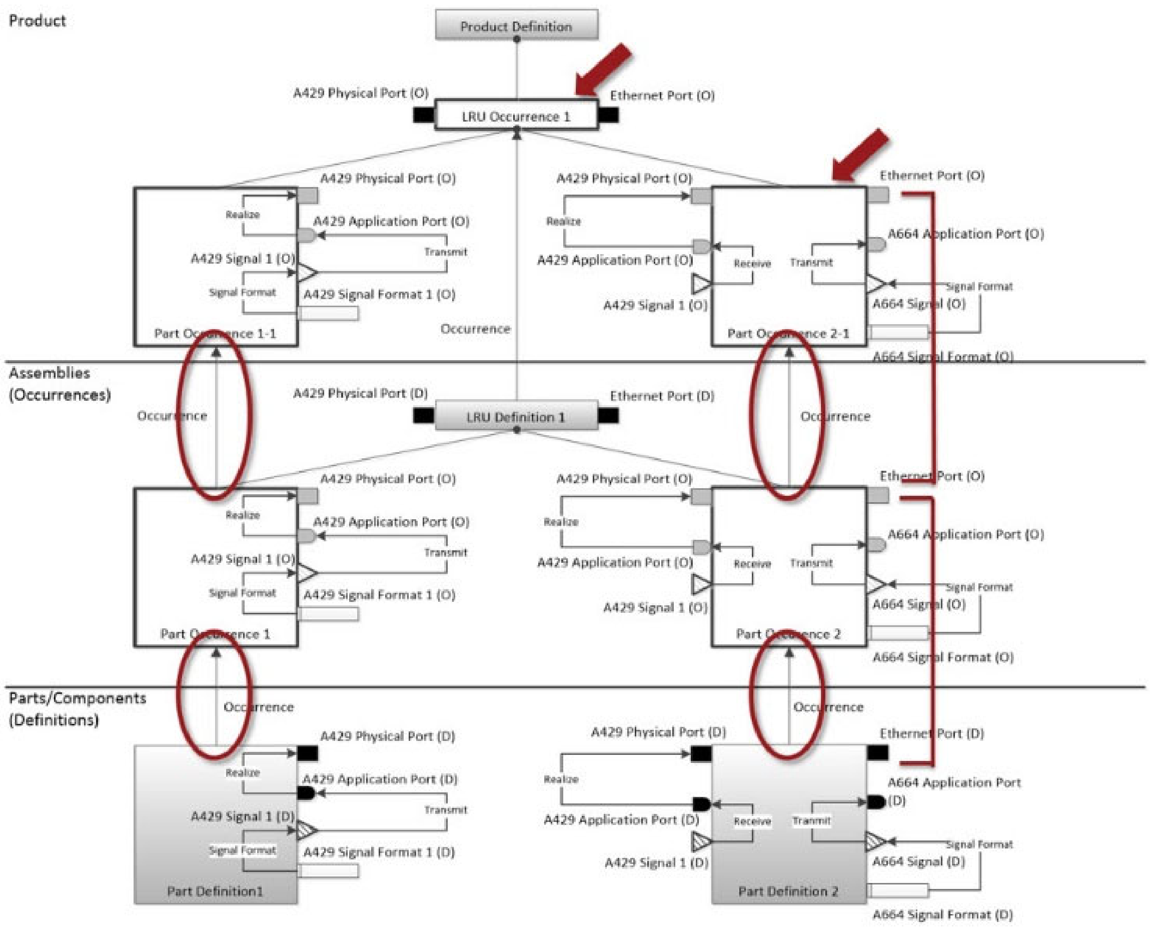


Figure 136. Assembly installation on the airplane

It should be noted that the definition/occurrence modeling can be done to an infinite number of abstraction levels, which is why it is a powerful in enabling model consistency.

Figure 137 shows how the models can be kept consistent when changes are made. In this example, an input physical port and input application port are added to the part definition.

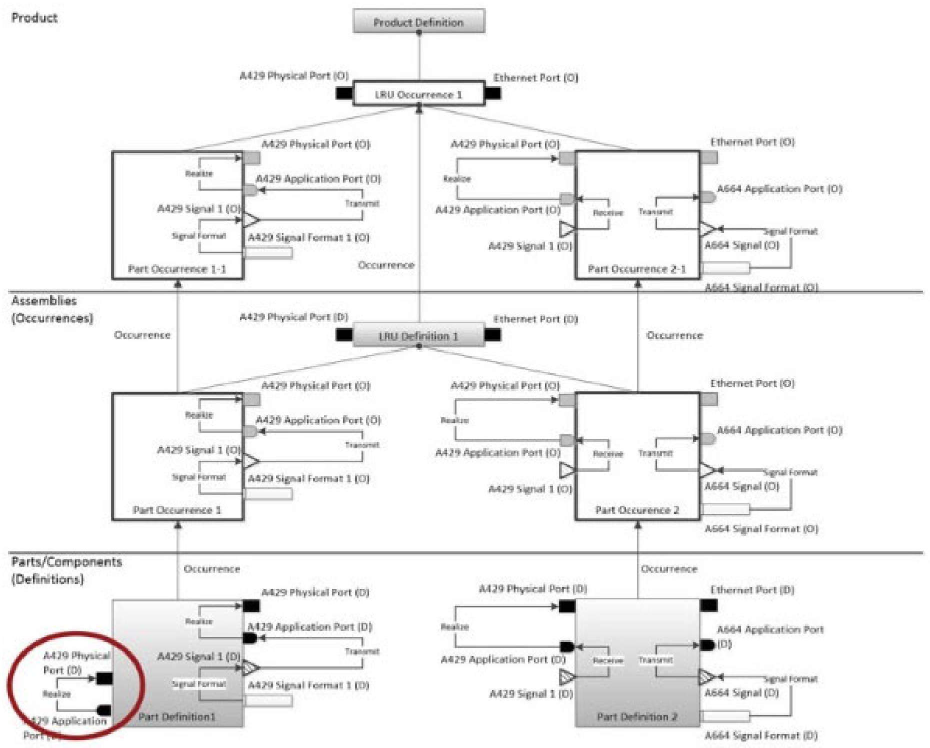


Figure 137. Additional system definition

Figure 138 shows the propagation of the additional definition at all occurrence levels. This helps keep the models consistent at the different abstraction levels when changes are made.

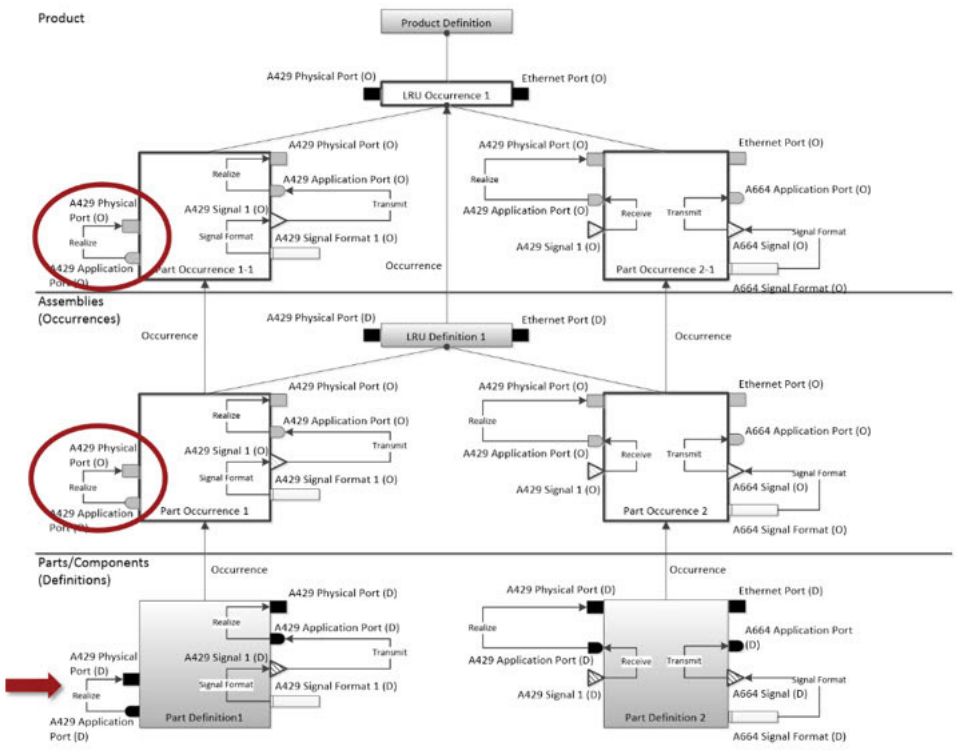


Figure 138. Propagation of definition level changes to occurrences

This is a powerful functionality, particularly when managing interfaces. This helps ensure the consistency of the interface models at different abstraction levels.

As shown in Figure 139, interfaces should never be created at the definition level. Interfaces should be created at the occurrence level. Intuitively, this should make sense. This was described in further detail in section 6.1, which discussed how it is important to know if an LRU is subscribing the left, center, right outputs (or some other combination).

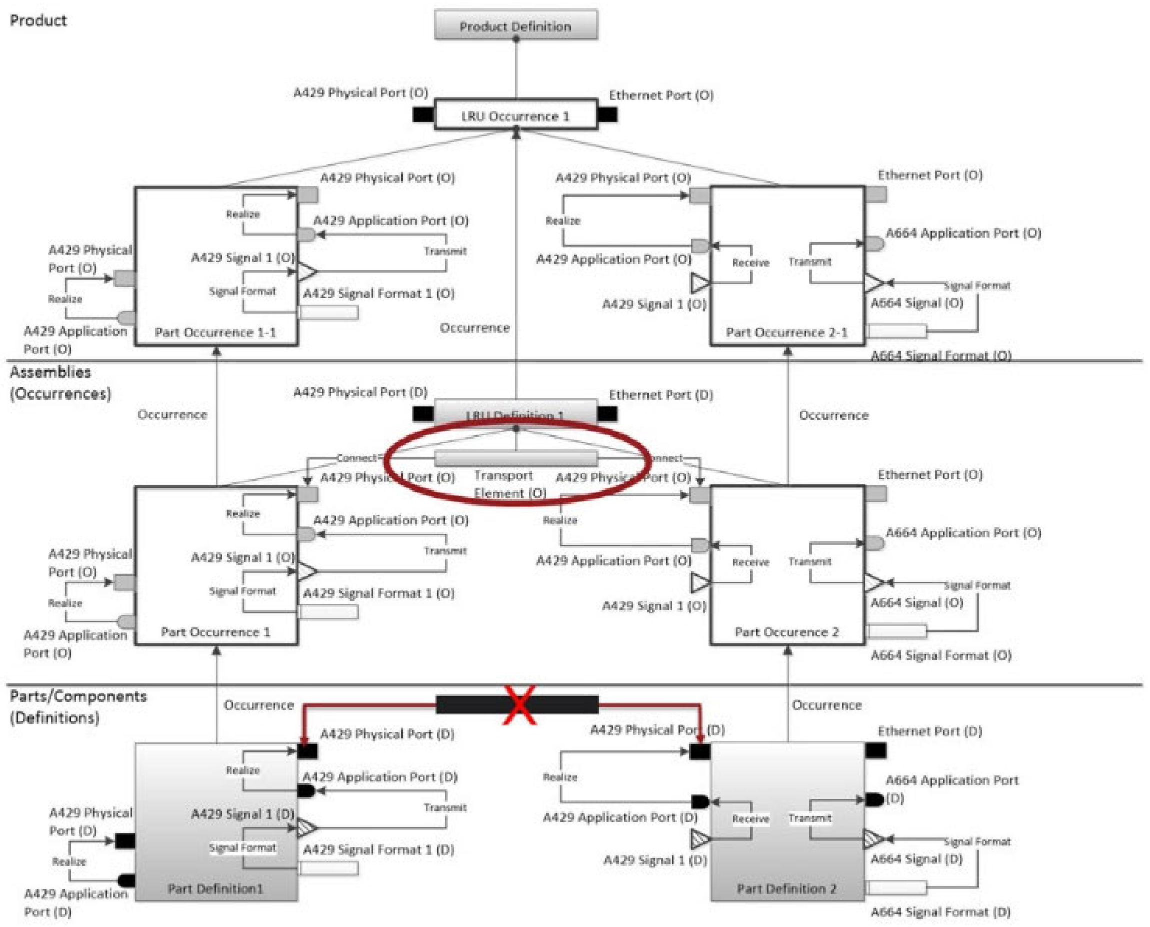


Figure 139. Interfaces at occurrence level

Figure 140 shows how the interfaces made at the occurrence level would propagate to higher occurrence levels.

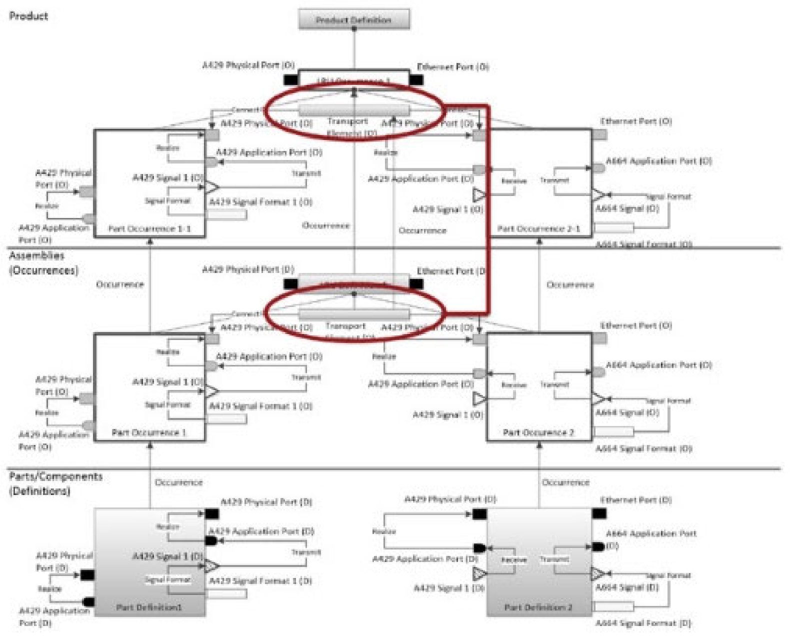


Figure 140. Interface fault propagation

Figure 141 shows the creation of a physical port definition to a part definition, which has occurrences on multiple products.

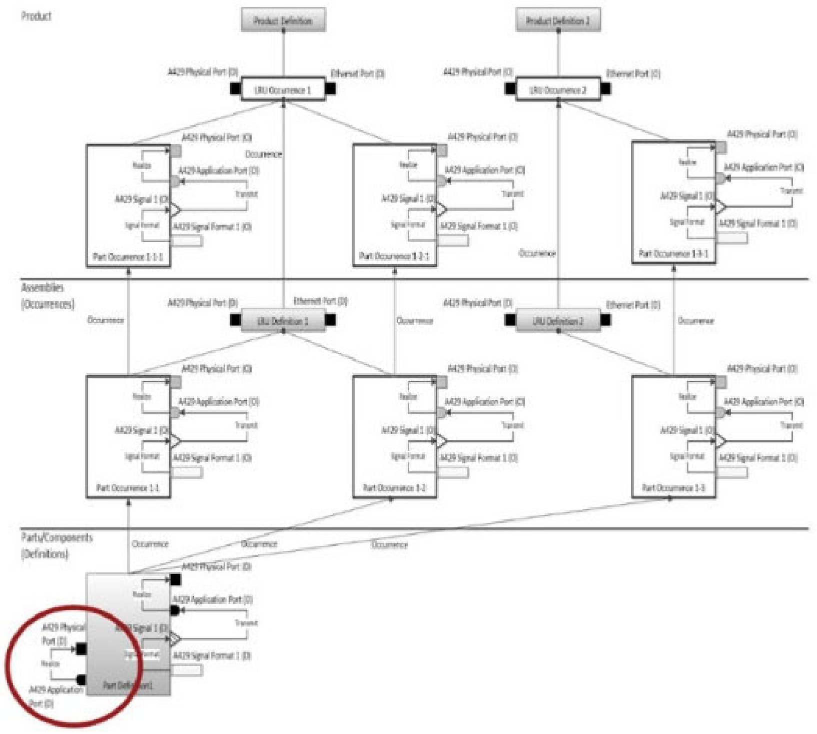


Figure 141. Definition creation for reuse across products

Figure 142 shows how the definition can propagate via occurrences both within and across products. This ability is particularly helpful if the same component is being used on different products (as often happens in the aviation industry).

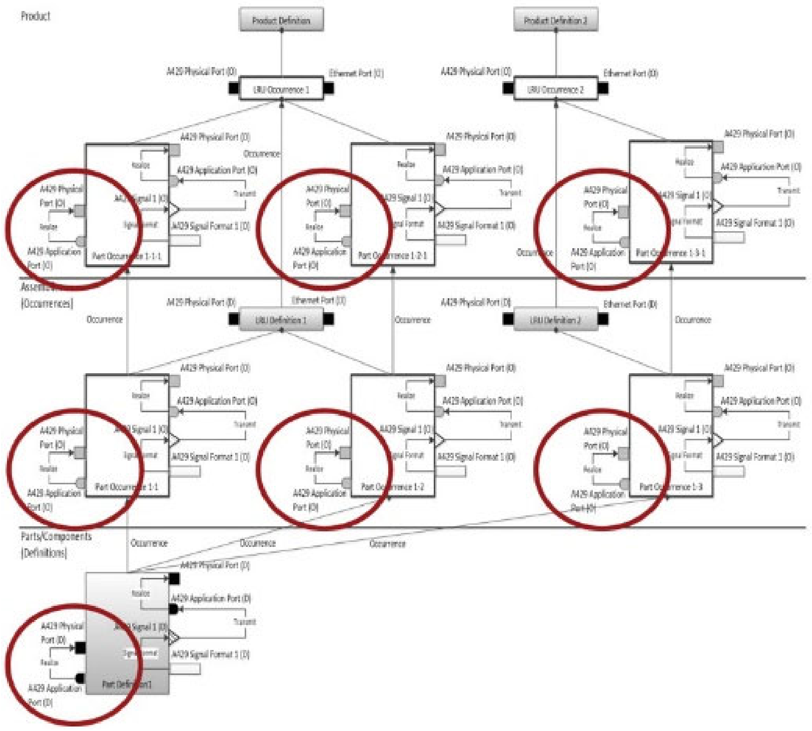


Figure 142. Propagation within and across products

One of the biggest benefits of MBSE and MBSA is having a "single source of truth," which precludes the need to synchronize data across different sources. This is where MBSE and MBSA can achieve efficiencies from the existing, acceptable processes.

One of the biggest challenges in MBSE and MBSA is to avoid having to recreate models created at different abstraction levels. Another challenge is the ability to keep the models synchronized. The preceding examples demonstrated how the definition/usage/occurrence modeling could address these challenges, particularly for highly integrated architectures. This capability can be foundational for being able to assess the completeness and correctness of the models, particularly from both a vertical and horizontal integration perspective.

9 Recommendations from MBSE/MBSA implementation

This section describes the criteria, mitigation approaches, recommendations, training material, and position papers, particularly related to training and existing guidance/standards. Table 17 describes the approach to address these research objectives.

Table 17. Current guidance and research approach

Research Objectives	**Detailed Work Description**
Develop criteria, mitigation approaches, recommendations, training material, or position papers	Identify potential additional training material. Identify proposed modifications to existing guidance/standards.

9.1 Training

If there are gaps in understanding or implementing the underlying principles of SE or safety, moving to MBSE and MBSA will not provide a panacea. Foundational knowledge, particularly related to SE and safety and modeling, are prerequisites to effective MBSE/MBSA training and implementation. This foundational training is applicable from both a development perspective (SE practitioner and safety practitioner) and from the certification engineer perspective. The certification engineer is considered to be an FAA certification engineer or delegate.

The training progression, as shown in Figure 143, consists of four fundamental building blocks:

- Foundational SE and safety training, which addresses the required SE and safety principles.
- Foundational modeling training, which addresses understanding the data models, engineering databases, creating model queries and modeling standards.
- MBSE and MBSA training, which addresses how to implement the SE and safety principles in a model-based environment.
- MBSE architect training, which addresses how to create an MBSE/MBSA integrated architecture and tool suite specifications.

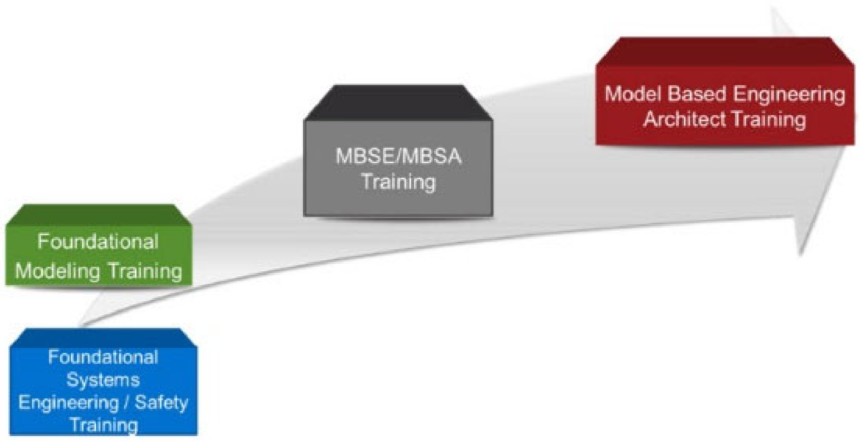

Model Based Engineering Architect Training
MBSE/MBSA Training
Foundational Modeling Training
Foundational Systems Engineering / Safety Training

Figure 143. Training progression

Table 18 identifies the recommended audiences for the respective foundational training blocks.

Table 18. Recommended training audiences

Training	**SE/Safety Practitioner**	**Modeling Practitioner**	**MBSE/Model-Based Safety Practitioner**	**Certification Engineer**	**MBSE Architect**
Foundational SE/Safety Training	X		X	X	
Foundational Modeling Training		X			X
MBSE/MBSA Training	X		X	X	
MBE Architecture Training					X

The SE/safety practitioner conducts the SE and safety activities. Examples of SE activities include requirements authoring, requirements validation, and implementation verification. Examples of safety activities include conducting FHAs, PRAs, and systems safety assessments.

The modeling practitioner creates the data models, which is usually done in UML. The importance of data models was discussed in section 8.

The MBSE/model-based safety practitioner uses MBSE and MBSA to conduct the required SE and safety activities.

The certification engineer is responsible for ensuring that the required certification activities have been completed, regardless of whether it is done with the more traditional document-focused approach or a model-based approach.

The MBE architect is responsible for integrating the overall MBE data models. As an example, tasks can include integrating the supply chain and manufacturing production.

Note: There should be a limited number of MBE architects. As noted in Table 18, the training material is only recommended for one job role. It is included for completeness.

These roles are not meant to be mutually exclusive. It is possible that the same individual creates both the data models and conducts the model-based SE or safety activities.

These recommendations are about recommended training content. This section/paper provides no recommendation for who should create or conduct the training.

9.1.1 Foundational systems engineering and safety training

It is important to have a good understanding of SE and safety before trying to implement MBSE and MBSA. It is important to have the required foundational knowledge before transitioning to MBSE and MBSA. Figure 144 highlights the importance of having a strong foundation in SE and safety; this is critical to successfully implementing MBSE and MBSA.

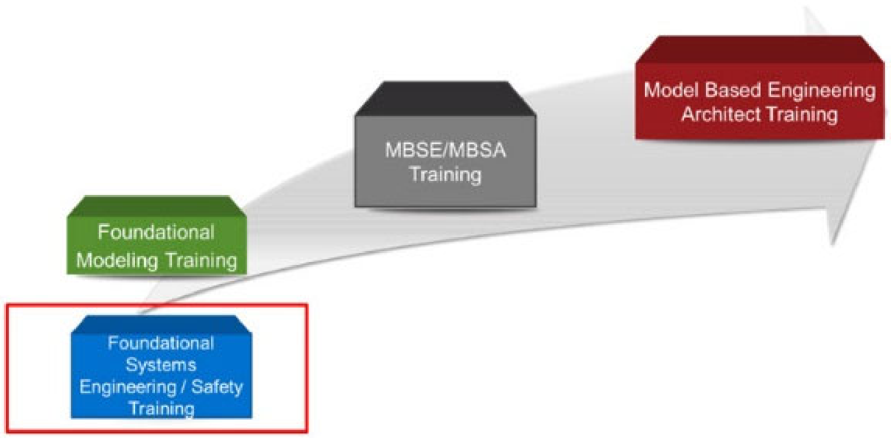

Model Based Engineering Architect Training
MBSE/MBSA Training
Foundational Modeling Training
Foundational Systems Engineering / Safety Training

Figure 144. Foundational systems engineering and safety training

The SE engine, as shown in Figure 145, provides a good framework for the foundational SE and safety principles. It includes activities and processes that are conducted to ensure that the user needs are met. It consists of the following:

- Requirements architecture depicts a hierarchy of requirements elements and their relationship to each other, as well as their relationships to other architectural elements.

- Functional architecture identifies what has to be done. It is the hierarchical arrangement of functions, their internal and external (external to the aggregation itself) functional interfaces and external physical interfaces, their respective requirements, and their design constraints. Functional architecture is allocated to the logical architecture, showing that logical architecture fulfills the intent of the functional architecture.
- Logical architecture identifies how it is done. Logical architecture includes definition of logical elements and the definition of the external and internal system interfaces. Logical system architecting establishes a "type of" depiction of the design solution, which allows for analysis of alternatives prior to entering physical architecture.
- Physical architecture identifies how it is implemented. The physical architecture associates identified physical components (which have properties, such as size, volume, density, part specifications) to their logical representation in the logical architecture.
- Validation ensures that requirements are complete and correct.
- Verification ensures the implemented design meets the requirements.
- Systems analysis and control are activities used to validate the systems architecture and to maintain control (via configuration management and program management practices).

As shown in Figure 145, safety is embedded in each of the different elements of the SE.

For example, safety-related requirements, which are derived from safety analyses, are part of the requirements architecture. The functional architecture contains functions used for the safety airplane-level and system-level FHAs. The logical architecture contains the systems architecture that will be used in the PSSAs and failure mode effects analyses (FMEAs). The physical architecture contains the spatial layout of the design, which is needed to support PRAs and ZSAs. Verification and validation include safety activities. For example, PASAs and PSSAs are conducted to ensure the safety requirements are complete and correct (validation). Verification is conducted to ensure that the implementation is meeting the safety requirements.

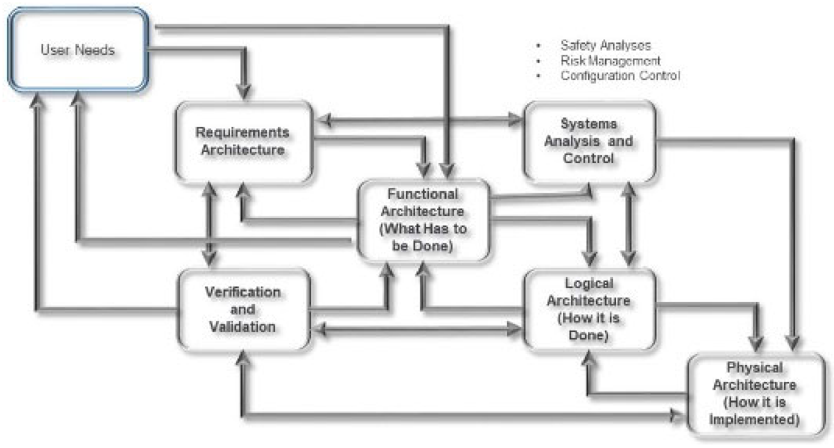

User Needs
Safety Analyses
Risk Management
Configuration Control
Requirements Architecture
Systems Analysis and Control
Functional Architecture (What Has to be Done)
Verification and Validation
Logical Architecture (How it is Done)
Physical Architecture (How it is Implemented)

Figure 145. Training progression

Figure 146 contains an example of a training curriculum that would help to ensure there is a strong foundational understanding of SE and safety principles. This training should be independent of any tool or modeling discussions. The recommended foundational SE and safety training should include:

- Introduction to SE and safety principles training.
- Requirements authoring and management training.
- Functional Analysis training.
- Functional Integration training.
- Interface management training.
- Safety training.
- Configuration management training.

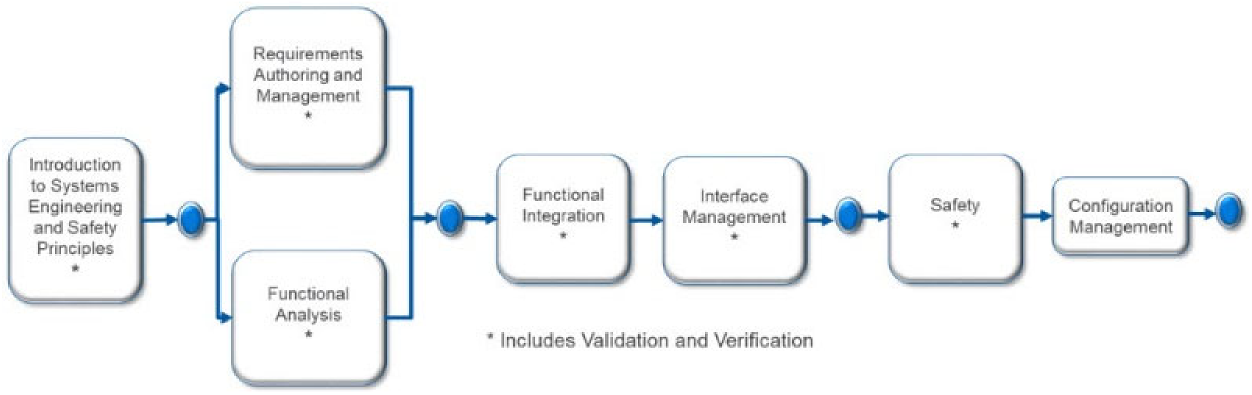

Introduction to Systems Engineering and Safety Principles *
Requirements Authoring and Management *
Functional Analysis *
Functional Integration *
Interface Management *
Safety *
Configuration Management
* Includes Validation and Verification

Figure 146. Sample training curriculum

As noted in Figure 146, validation and verification are included in multiple parts of the training curriculum. For example, requirements authoring and management training should include requirements validation (completeness and correctness). Safety training should include how to validate and verify that the systems architecture is safe in nominal and non-nominal conditions.

9.1.1.1 Introduction to systems engineering and safety principles training

Entry-level training should be available for individuals who are not familiar with SE and safety principles. This entry-level training should provide an overview of the entire SE process and address the following questions:

- What is "SE"?
- What are safety principles"
- Why use it?
- When is it used?
- How is SE used?

9.1.1.2 Requirements authoring and management training

Developing requirements and objectives and verification plans is a key element of the larger SE process. It is important to have a strong understanding of the steps to define, validate, verify, and manage requirements throughout the life of the product.

The requirements authoring and management training should address the following topics:

- Requirements management process.
- Requirements development.
- Requirements validation.
- Requirements verification.

The training should address how new and derived requirements are developed using inputs from a variety of sources and analytical approaches to define specific, valid, and verifiable product requirements. It should address how to decompose the requirements into lower level requirements. The training should emphasize that the requirements management process steps are performed iteratively and continually over the life of the product, as requirements are added or modified, with increasing levels of fidelity.

The training should address how to validate that requirements are correct and complete. Requirements correctness includes checking the implementation of standard guidelines for writing good requirements and ensuring that the requirements are technically correct. To ensure that the requirements are complete, the requirements architecture should be reviewed to identify missing, competing, conflicting, duplicate, testable, or obsolete requirements. This includes requirement traceability analysis, which reviews the parent-child traceability for all requirements developed as a result of top-down decomposition activities. This helps validate that applicable requirements can be easily and unambiguously tracked to their next higher levels and do not inappropriately skip levels, thus demonstrating completeness.

The training should address how to verify that the design correctly implemented the validated requirements. It should address different verification methods, such as analysis, inspection, demonstration, and test.

9.1.1.3 Functional analysis training

The functional analysis training should include:

- Identifying what the system has to do.
- Identifying functions the system needs to perform (the "what").
- Creating a functional architecture.

The training should address the required tasks to be performed during functional architecting. A key element of a functional analysis is the creation of a functional architecture that describes the functions, and then allocates those functions to a design. This includes describing the requirements for each function, organizing the functions logically, detailing the functions by levels, and allocating the functions to the logical or physical architecture.

Foundational elements of functional analysis include:

- Functional decomposition.
- Functional data flow diagrams.
- Functional flow block diagrams, which identify the temporal relationships between functions.

Figure 147 shows an example of a functional decomposition, which identifies the parent-child hierarchy of functions. The way in which the functions are conceived will be the form that the design and final product take.

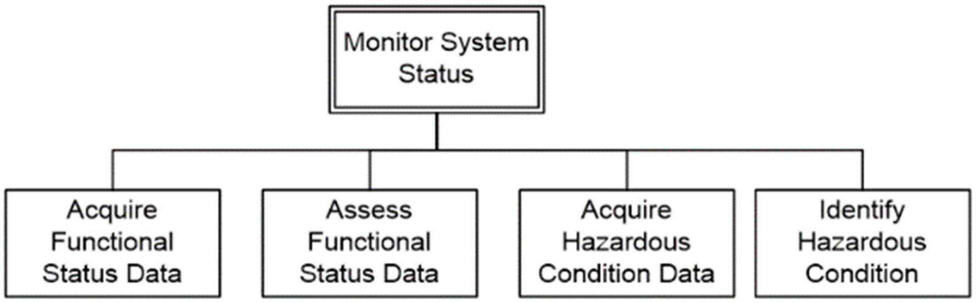

Monitor System Status
Acquire Functional Status Data
Assess Functional Status Data
Acquire Hazardous Condition Data
Identify Hazardous Condition

Figure 147. Functional decomposition

Figure 148 shows an example of a functional data flow diagram, which identifies the input and output data flows (i.e., interfaces) between functions.

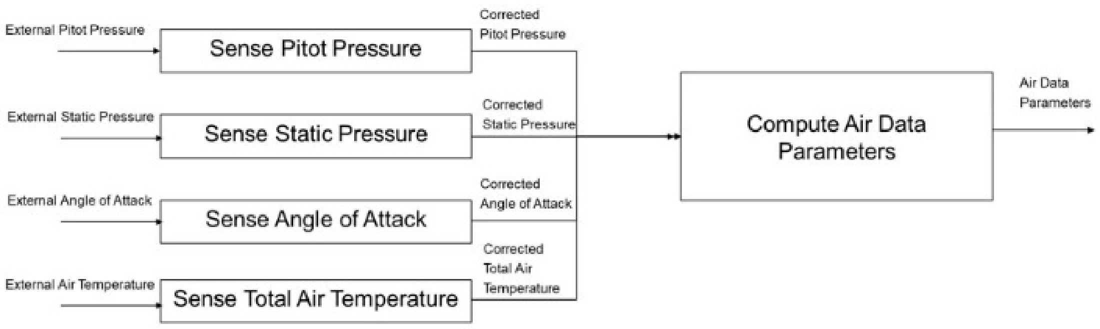


Figure 148. Functional data flow diagram

Figure 149 shows an example of a functional data flow diagram, which identifies the temporal relationships between functions. It is a simplified diagram, showing the relationships between the Environmental Control System (ECS) with the galley.

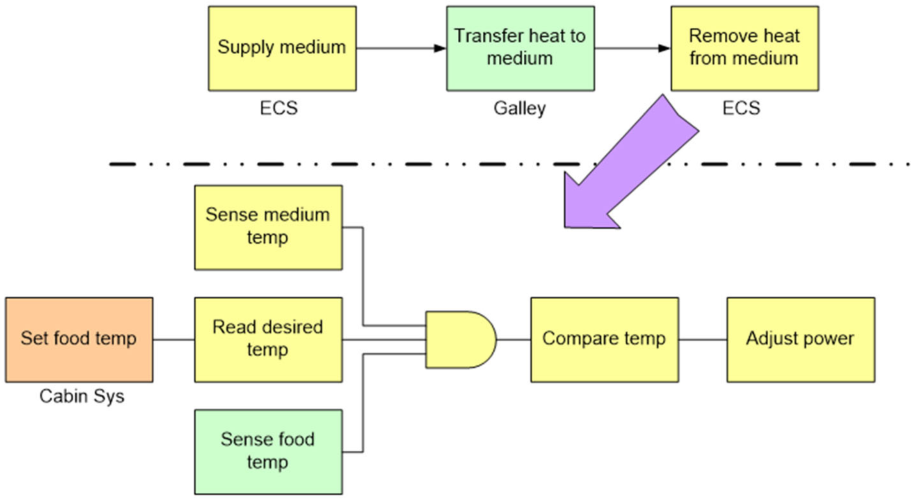


Figure 149. Functional flow block diagram

Functions can be considered to be a subset of the requirements architecture. The functions are captured in the "verb-noun" format and used to create data flow diagrams and functional flow block diagrams, as shown in the examples above. Theoretically, on a completely new program (e.g., new architecture, new mission), the functions should be created first. The functions would then be expanded into the more expansive requirements format. Additional performance and constraints would be generated.

Figure 150 shows the interrelationships between the requirements architecture and the functional architecture.

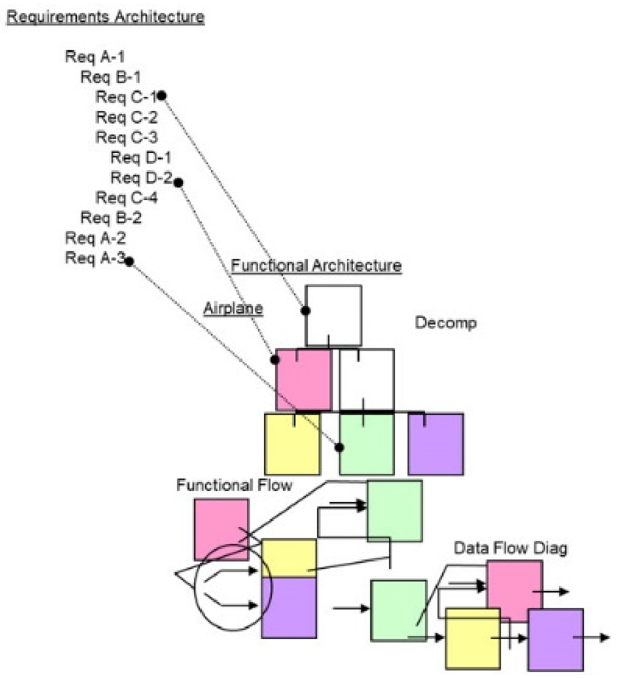


Figure 150. Requirements and functional architecture

9.1.1.4 Functional integration training

Functional integration identifies interfaces and validates functional integration correctness. An integral part of the functional integration activities is the validation of the logical architecture, which is implementing the functions. The functional integration training should include:

- Understanding the differences between federated and integrated distributed system architectures.
- Understanding the challenges associated with increasing integration.
- Understanding the purpose and composition of a systems architecture.
- Understanding what a systems architecture is.

- Identifying users and their needs.
- Developing functional architectures.
- Identifying performance measures
- Validating requirements architectures by analyzing requirements allocations to functions and systems architecture elements.
- Developing logical architectures.
- Integrating systems.
- Analyzing integrated architectures for potential failure modes, effects, and criticality.

As shown in Figure 151, functional integration includes creating the logical architecture and integrating it with the requirements architecture and functional architecture.

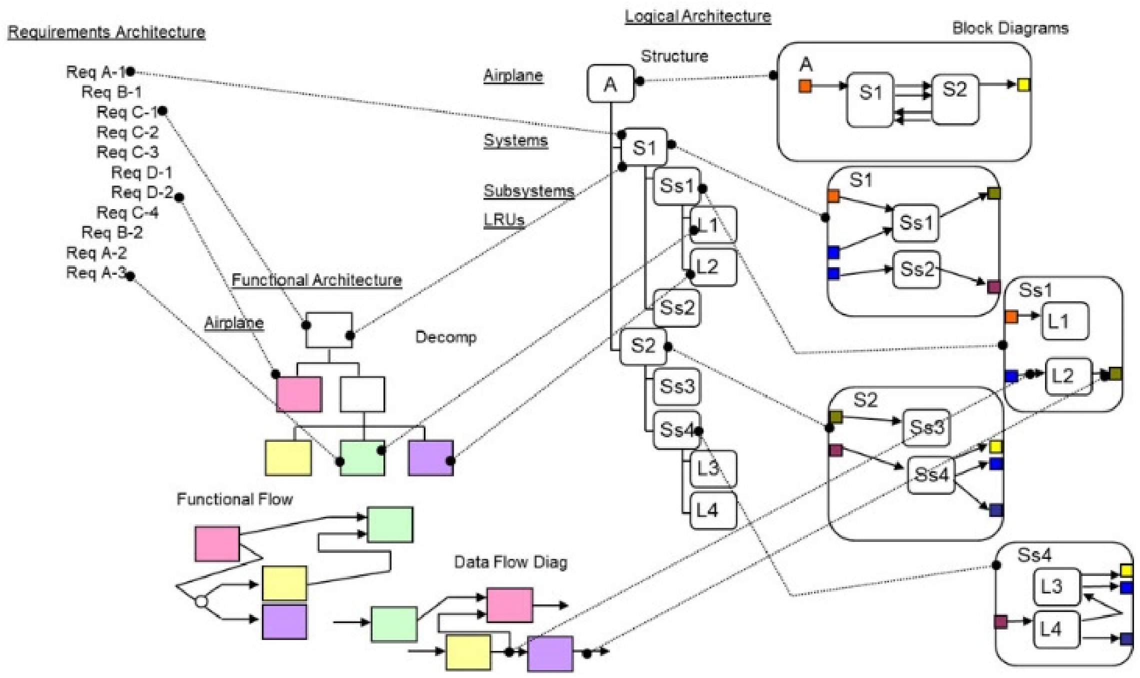


Figure 151. Functional integration

Figure 152 shows some of the increased integration challenges, as federated systems move to integrated distributed system architectures. Increased integration helps to improve performance and reduce weight by reducing the number of required LRUs or physical components and wiring, and improving modularity for updates. Functional integration training should clarify the required activities that need to be accomplished for the different types of system architectures.

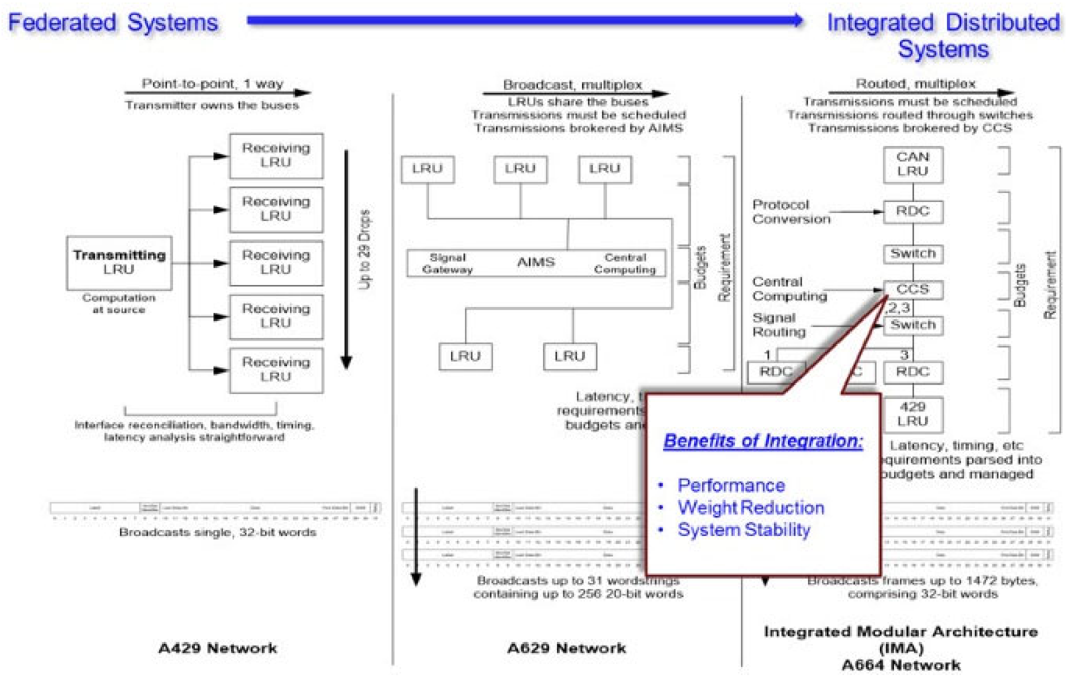

Federated Systems
Integrated Distributed Systems
Point-to-point, 1 way
Transmitter owns the buses
Transmitting LRU
Computation at source
Receiving LRU
Up to 29 Drops
Interface reconciliation, bandwidth, timing, latency analysis straightforward
Broadcasts single, 32-bit words
A429 Network
Broadcast, multiplex
LRUs share the buses
Transmissions must be scheduled
Transmissions brokered by AIMS
LRU
Signal Gateway
AIMS
Central Computing
Budgets
Requirement
Broadcasts up to 31 wordstrings containing up to 256 20-bit words
A629 Network
Routed, multiplex
Transmissions must be scheduled
Transmissions routed through switches
Transmissions brokered by CCS
CAN LRU
Protocol Conversion
RDC
Switch
Central Computing
CCS
Signal Routing
429 LRU
Latency, timing, etc requirements parsed into budgets and managed
Broadcasts frames up to 1472 bytes, comprising 32-bit words
Integrated Modular Architecture (IMA) A664 Network
Benefits of Integration:
Performance
Weight Reduction
System Stability

Figure 152. Increased integration

The transition from federated to integrated, distributed systems increases the required integration activities to ensure that the system is going to work as expected, in both nominal and failure conditions.

For example, as shown in Figure 152, an ARINC429 architecture is federated, with most LRUs executing functions locally. There are point-to-point, unidirectional buses; bi-directional communications require two buses. Signals are transmitted from publisher to all subscribers.

In an ARINC629 architecture, some centralized computing exists. There are multiplex buses, two-way communications, and multiple transmitters per individual bus. Signals are broadcast to every LRU connected to the bus. Bus schedules are required to avoid signal clashes.

In an ARINC664 architecture (which is common for IMA architectures), computing is centralized. Most LRUs gather "raw" data, which is then processed by centralized computing functions. There are multiplex buses, two-way communications, and multiple transmitters per individual bus. ARINC664 messages are not broadcast over the entire network. Message traffic is directed to subscribers through routers (switches).

The transition from federated to integrated, distributed architectures can increase the number of interfaces, which increases the importance of interface management.

9.1.1.5 Interface management training

To ensure that systems will work properly in both nominal and failure cases, it is important to be able to develop, negotiate, and manage logical interfaces. This includes designating and defining the logical interfaces (internal and external) and documenting interface descriptions into an integrated data package. Interface management training should cover topics such as:

- Establishing useful boundaries around a system.
- Defining interfaces that cross the system boundaries in terms of connections, conduits, and media.
- Describing how interfaces are managed and reconciled across system boundaries.
- Describing how interface standards facilitate the interface management task.
- Describing how systems evolution is making interface management more complicated and difficult.
- Describing how logical architectures facilitate interface management and reconciliation.
- Describing in detail the process for electronic interface management.
- Discussing the various safety implications of interface management.

9.1.1.6 Safety training

The foundational safety training material should address the following questions:

- What does it mean for the system to be safe?
- How do you ensure good architectural and functional decisions?
- How do you ensure that appropriate mitigations/redundancies are in place?
- What are the objectives of the different types of safety analyses?
- How do you know when analyses are complete?
- How do you account for interactions between systems?
- How do you ensure safety will be maintained for subsequent changes?

Figure 153 identifies different types of safety analyses that should be covered in foundational safety training.

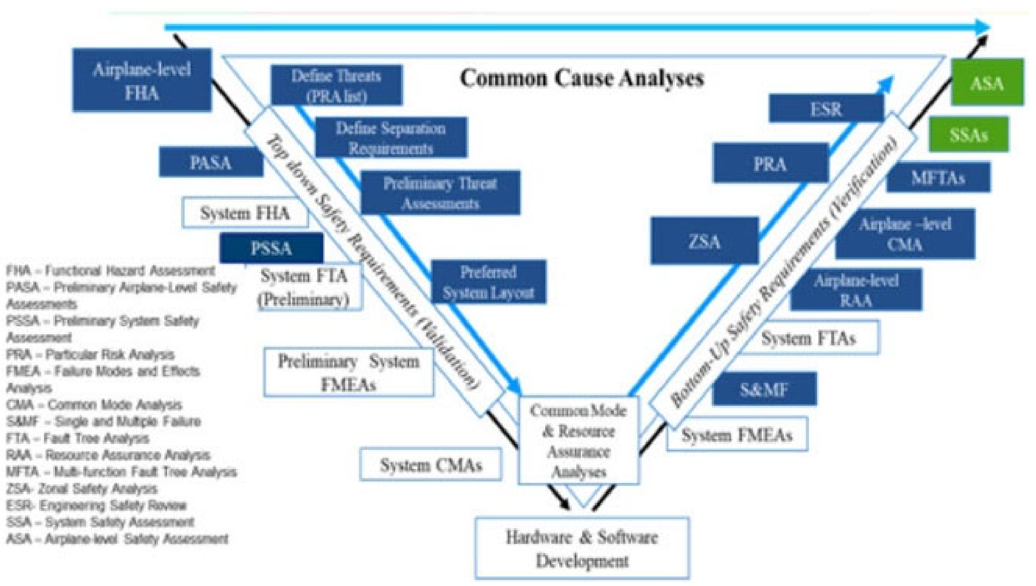

Common Cause Analyses
Airplane-level FHA
Define Threats (PRA list)
Define Separation Requirements
Preliminary Threat Assessments
Preferred System Layout
PASA
System FHA
PSSA
System FTA (Preliminary)
Preliminary System FMEAs
System CMAs
Top down Safety Requirements (Validation)
Common Mode & Resource Assurance Analyses
Hardware & Software Development
Bottom-Up Safety Requirements (Verification)
ESR
PRA
ZSA
ASA
SSAs
MFTAs
Airplane -level CMA
Airplane-level RAA
System FTAs
S&MF
System FMEAs
FHA – Functional Hazard Assessment
PASA – Preliminary Airplane-Level Safety Assessments
PSSA – Preliminary System Safety Assessment
PRA – Particular Risk Analysis
FMEA – Failure Modes and Effects Analysis
CMA – Common Mode Analysis
S&MF – Single and Multiple Failure
FTA – Fault Tree Analysis
RAA – Resource Assurance Analysis
MFTA – Multi-function Fault Tree Analysis
ZSA- Zonal Safety Analysis
ESR- Engineering Safety Review
SSA – System Safety Assessment
ASA – Airplane-level Safety Assessment

Figure 153. Safety analyses

There should be a good understanding of each of the different types of safety analyses and assessments and their interrelationships, as follows:

- FHAs are comprehensive qualitative evaluations of airplane or systems functions, the failure conditions associated with each function, the airplane effects, and hazard classification of those effects. FHAs are performed at the airplane level and at the systems level.
- PASA is used to evaluate the proposed airplane architecture to develop safety requirements:
  - Ensure no single system fault can result in catastrophic outcome.
  - Ensure the robustness of design (e.g., allocation of the resource functions).
  - Evaluate iterative nature of the design throughout development maturity.
  - Identify safety requirements.
- PSSA is performed early during airplane development to examine proposed system architecture by evaluating the failure conditions and associated safety objectives.
- PRA is used to assess external events/threats to ensure continued safe flight and landing is maintained during and after the event.
- ZSA and ESRs are performed to ensure that equipment and transport element installation designs meet the safety separation requirements.

    - ZSA is performed on mature digital models, when the design, including transport elements, is representative of the delivery airplane configuration.
    - Engineering Safety Review (ESR) is performed, for example, on the as-built product, on a first article of new major installations, first airplane of a new type design.
- FMEA is a bottom-up systematic method of identifying the failure modes of a system item, function, or piece-part and determining the effects on the system and airplane. It is used to assess any potential single failure of the system components or software that could result in hazardous or catastrophic conditions.
- System Fault Tree Analysis (FTA) is a graphical quantitative evaluation of failure conditions that contribute to the top event.
- System Common Mode Analysis (CMA) is used to evaluate potential common mode failures in the systems that tree up to a top-level event. CMA is intended to verify:
    - Adequate independence between the AND-ed events.
    - Failures and errors affecting multiple systems are evaluated.
- Resource Assurance Analysis (RAA) is used to verify that the DAL of items supporting an airplane level FHA entry satisfy the hazard classification for each airplane level failure condition.
- Single and Multiple Failure (S&MF) Analysis provides a structured methodology to analyze failures of key integration components and functions to determine if airplane, flight crew, and occupant impacts are as expected and acceptable.

9.1.1.7 Configuration management training

Configuration management training should address the five functions of configuration management, as shown in Figure 154:

- Configuration management planning and implementation management provides the umbrella plan and aligns the configuration management related deliverables.
- Configuration identification represents the processes by which product configuration items are labeled and documented. This includes, for example, product structure design, part numbering, and management identifies documents, and approves and controls changes and variances to baselines, requirements, products, and documentation.

- Configuration status accounting records and reports information needed to manage the configuration of a product effectively throughout its life cycle, from design to delivery through disposal.
- Configuration verification and audit establishes the traceability to verify that the product meets its requirements and has been accurately represented in all configuration documentation (i.e., as built = as planned = as designed = as contracted/specified).
- Serialization rules, part marking requirements. Proper configuration identification is essential to provide technical and contractual control, verification, approval, and change control for product configurations and their interfaces.
- Configuration change.

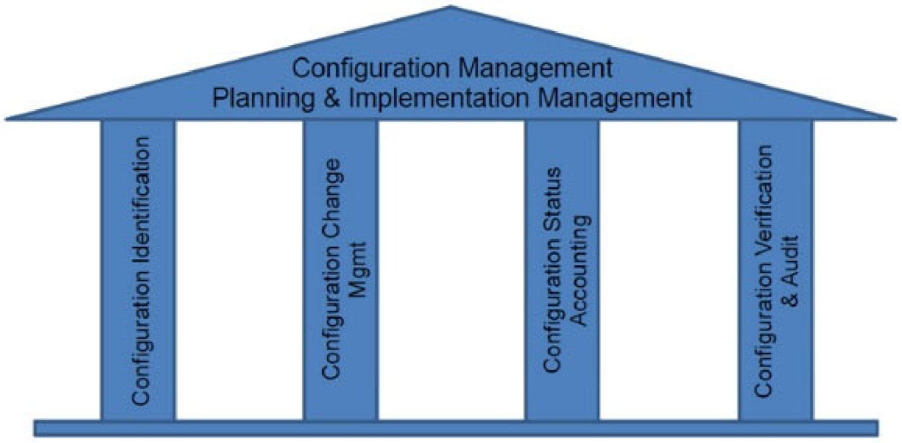

Configuration Management
Planning & Implementation Management
Configuration Identification
Configuration Change Mgmt
Configuration Status Accounting
Configuration Verification & Audit

Figure 154. Functions of configuration management

Configuration management should address both baseline capture and version control. In an MBSE/MBSA environment, a baseline is a snapshot of the entire data set. It is a "frozen" version of the data. Individual data objects can have versions. The training should emphasize the importance of ensuring that links between different data are treated as their own element; the links should also have versions and should be included in baselines.

9.1.2 Foundational modeling training

With a solid foundation in SE and safety principles, the next training emphasis should be on foundational modeling training, as shown in Figure 155.

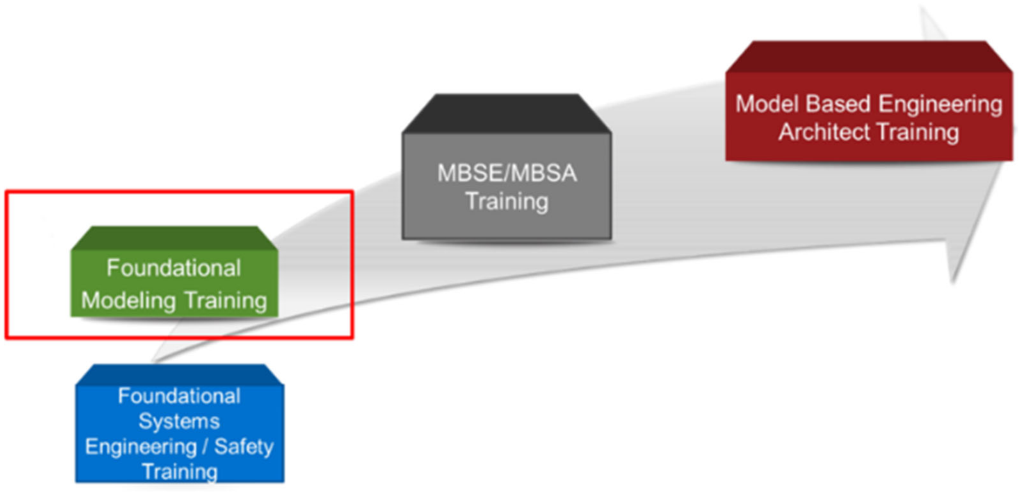

Model Based Engineering Architect Training
MBSE/MBSA Training
Foundational Modeling Training
Foundational Systems Engineering / Safety Training

Figure 155. Functions of configuration management

Figure 156 shows examples of foundational modeling training topics, consisting of:

- Introduction to data modeling.
- Introduction to engineering databases.
- Composing model queries.
- Modeling standards.

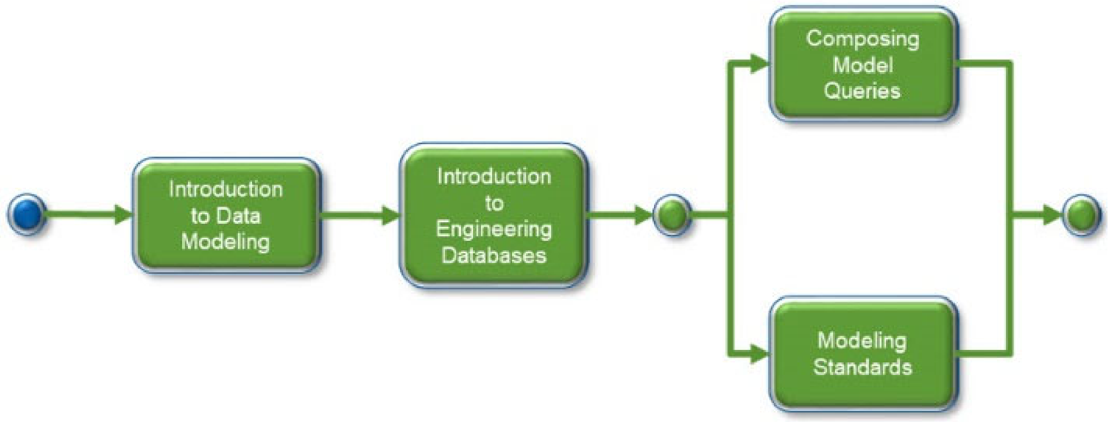

Introduction to Data Modeling
Introduction to Engineering Databases
Composing Model Queries
Modeling Standards

Figure 156. Foundation modeling training topics

9.1.2.1 Introduction to data modeling

Understanding data modeling will greatly increase the likelihood of successful MBSE and MBSA implementation. If MBSE and MBSA are implemented without an integrated, coherent data model structure, it will be very difficult to achieve the desired MBSE and MBSA benefits. The data model is foundational to being able to integrate the different models used for the SE and safety activities.

An introduction to data modeling should address some basic topics, such as:

- What is a model?
- What are data?
- What is a data model?
- Why use model data?
- What is data model composition?
- What are objects?
- What are attributes?

For example, there should be an understanding on considerations to make on whether to model data as an object or an attribute of an object. An example is the location of equipment (e.g., forward equipment bay). Location could be modeled as hierarchical objects; it could also be modeled as attributes of objects (such as a component location attribute). It is important to understand the tradeoffs between the different approaches.

The training objectives should enable a person to be able to:

- Define and describe a model.
- Define and describe data.
- Define and describe a data model.
- Describe the elements comprising a data model.
- Effectively define objects representing data.
- Effectively define attributes describing objects.
- Effectively define relationships between model objects.
- Create an integrated data model.

UML is an effective language that can be used to create the data models, document the objects, attributes, and relationships, as shown in Figure 157. The importance of data models was discussed in Section 4.6.

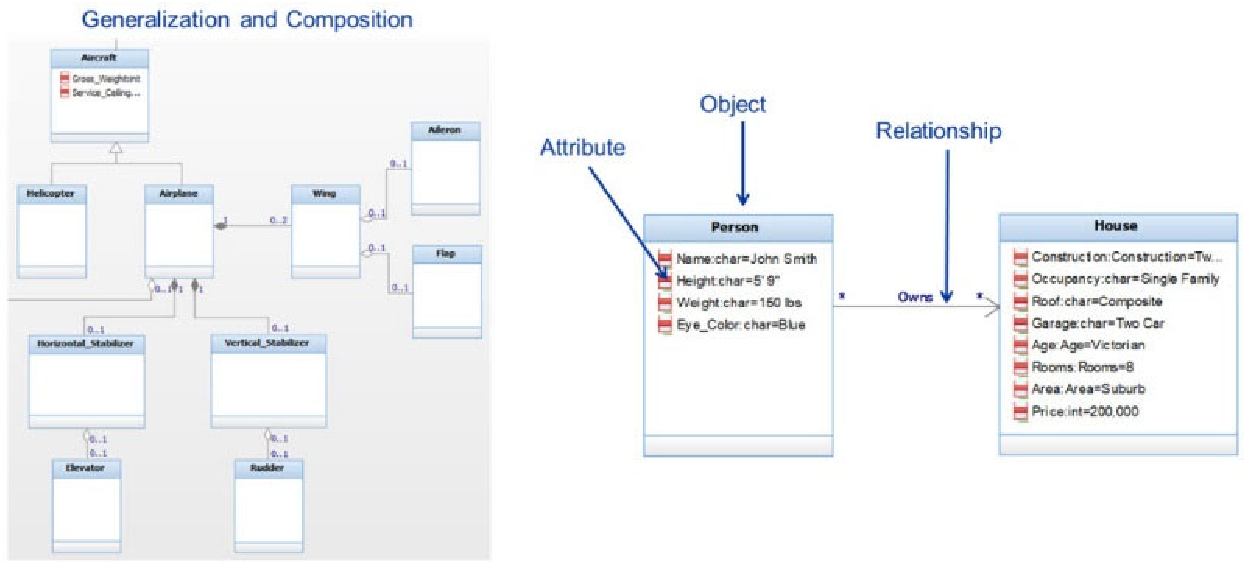


Figure 157. UML 2.0 class diagrams

9.1.2.2 Introduction to engineering databases

Implementing MBSE and MBSA will result in data rich models that can only be effectively managed in databases. Databases permit multi-user input and immediate synchronization, improving efficiency and productivity. Usage of a database is also integral to providing the required configuration management. Engineering databases are also integral to creating the "single source of truth."

Figure 158 shows the transition from traditional systems engineering in a document-based environment to one with model-based generation of artifacts. As discussed in in section 7.1, MBSE and MBSA allow documents to be generated from a single source of truth. This precludes the need to synchronize the same data in multiple documents.

Training should identify how to:

- Create a data model capturing the content of a document.
- Configure a database to represent a data model.
- Populate a database with a model comprising the entire contents of a document.
- Perform basic queries on a database model.

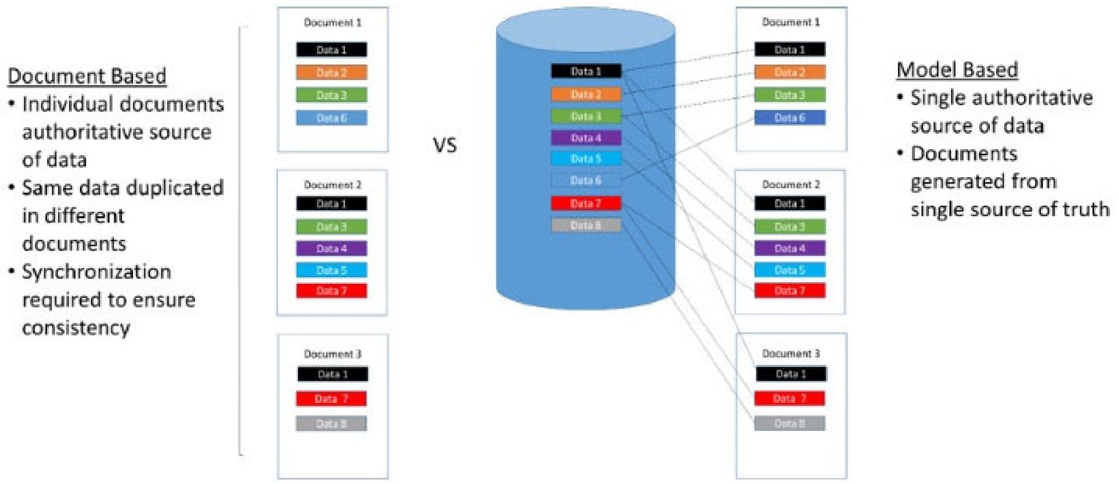
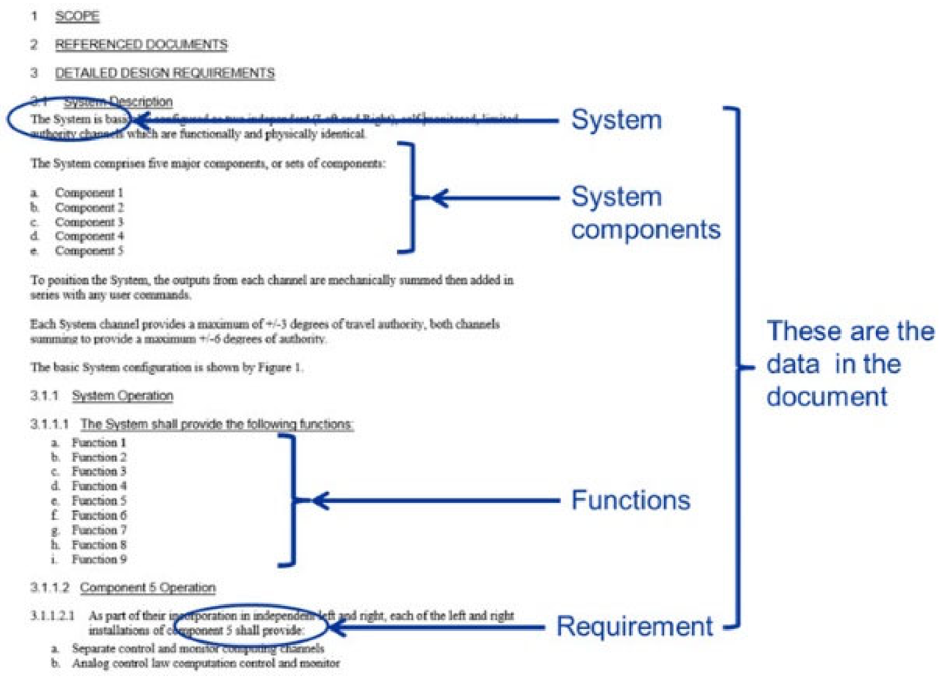

Document Based
• Individual documents authoritative source of data
• Same data duplicated in different documents
• Synchronization required to ensure consistency
Document 1
Data 1
Data 2
Data 3
Data 6
Document 2
Data 1
Data 3
Data 4
Data 5
Data 7
Document 3
Data 1
Data 7
Data 8
VS
Data 1
Data 2
Data 3
Data 4
Data 5
Data 6
Data 7
Data 8
Model Based
• Single authoritative source of data
• Documents generated from single source of truth

Figure 158. MBSE/MBSA model generated documents

It is beneficial for the training to emphasize that documents are not data in and of themselves. Figure 159 provides an example, showing how documents are repositories of data.



1 SCOPE

2 REFERENCED DOCUMENTS

3 DETAILED DESIGN REQUIREMENTS

3.1 System Description

The System is bas[illegible] configured as two independent (Left and Right), self-monitored, limited authority channels which are functionally and physically identical.

The System comprises five major components, or sets of components:

a. Component 1
b. Component 2
c. Component 3
d. Component 4
e. Component 5

To position the System, the outputs from each channel are mechanically summed then added in series with any user commands.

Each System channel provides a maximum of +/-3 degrees of travel authority, both channels summing to provide a maximum +/-6 degrees of authority.

The basic System configuration is shown by Figure 1.

3.1.1 System Operation

3.1.1.1 The System shall provide the following functions:

a. Function 1
b. Function 2
c. Function 3
d. Function 4
e. Function 5
f. Function 6
g. Function 7
h. Function 8
i. Function 9

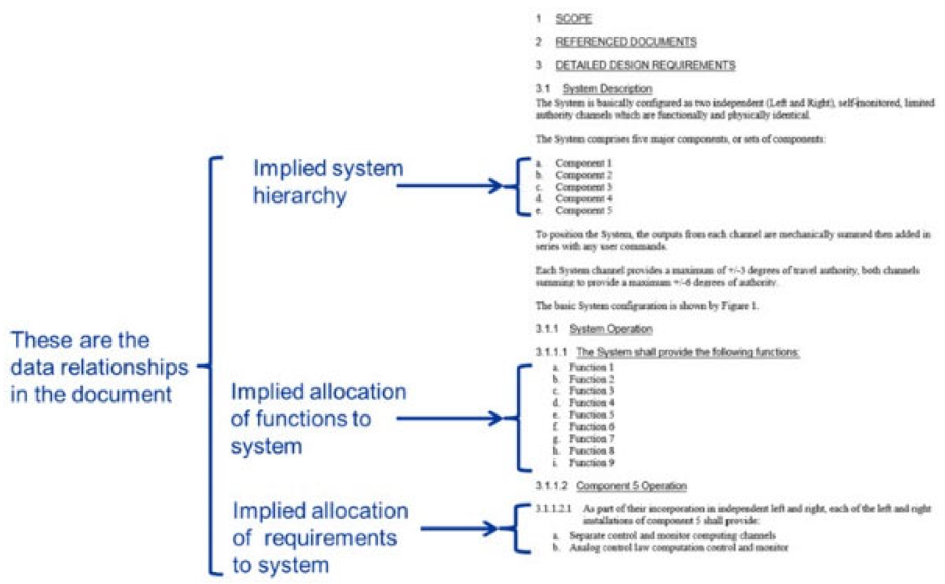
3.1.1.2 Component 5 Operation

3.1.1.2.1 As part of their incorporation in independent left and right, each of the left and right installations of component 5 shall provide:

a. Separate control and monitor computing channels
b. Analog control law computation control and monitor

Figure 159. Documents as repositories of data

As shown in Figure 160, a document has implied relationships that are embedded in its document structure. This includes both implied hierarchies and allocation relationships. There are several benefits to explicitly capturing these relationships and hierarchies in a model. Explicit relationships and hierarchies can be queried, analyzed, and validated. Rules can be established to help validate the completeness and correctness of the document's content. It is difficult (more time consuming) to do this with implicit relationships.


These are the data relationships in the document
Implied system hierarchy
Implied allocation of functions to system
Implied allocation of requirements to system

1 SCOPE

2 REFERENCED DOCUMENTS

3 DETAILED DESIGN REQUIREMENTS

3.1 System Description

The System is basically configured as two independent (Left and Right), self-monitored, limited authority channels which are functionally and physically identical.

The System comprises five major components, or sets of components:

a. Component 1
b. Component 2
c. Component 3
d. Component 4
e. Component 5

To position the System, the outputs from each channel are mechanically summed then added in series with any user commands.

Each System channel provides a maximum of +/-3 degrees of travel authority, both channels summing to provide a maximum +/-6 degrees of authority.

The basic System configuration is shown by Figure 1.

3.1.1 System Operation

3.1.1.1 The System shall provide the following functions:

a. Function 1
b. Function 2
c. Function 3
d. Function 4
e. Function 5
f. Function 6
g. Function 7
h. Function 8
i. Function 9

3.1.1.2 Component 5 Operation

3.1.1.2.1 As part of their incorporation in independent left and right, each of the left and right installations of component 5 shall provide:

a. Separate control and monitor computing channels
b. Analog control law computation control and monitor

Figure 160. Implied relationships in documents made explicit in models

Figure 161 provides an example of one way to parse the contents of a document into a data model. The training will need to recognize that this transition from a document-centric perspective to a model-based perspective can require a cognitive shift in the users. A data model infrastructure needs to be established in order to successfully implement MBSE and MBSA. It is important to also recognize that it can sometimes require a culture shift.

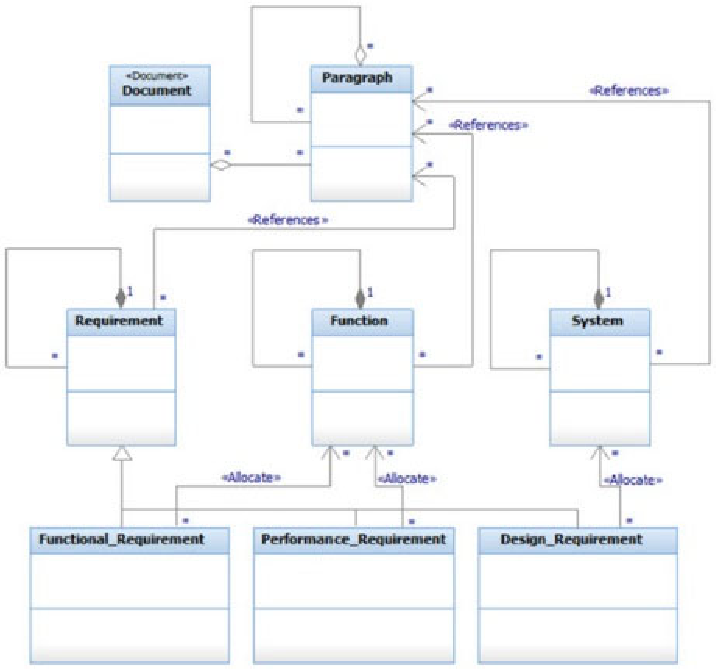


Figure 161. Data model for model-based documents

9.1.2.3 Composing model queries

One of the key benefits of model-based engineering is the ability to query and analyze the data. The ability to create model queries is important, because it enables the identification of missing, incorrect, or incompatible data. Examples of errors are unallocated functions, discrepancies between required network timing definitions, and bandwidth overutilization. Without query capabilities, MBSE and MBSA cannot be achieved; models can be captured, but they cannot be adequately analyzed.

The training material should enable the user to:

- Define a database query, including the SELECT statement and SELECT statement clauses.
- Identify the common elements of a database query and describe their proper usage.
- Apply database queries to system architecture models and construct logical queries of these models by appropriately applying the query elements.

Building a useful QUERY entails structuring the SELECT statement elements, so they are logically correct to return the desired results. The training should include fundamental and common elements of a SELECT statement, such as:

- FROM
- WHERE
- AND
- OR
- FOR EACH
- ADD
- REMOVE
- ORDER BY

It is important to be familiar with the underlying data model. This will greatly improve the queries and increase the understanding of what questions can be asked of the model. In turn, this improves the analytical capabilities. Figure 162 provides an example of the underlying data model that would support a query to know the functions and requirements that the system hardware must fulfill.

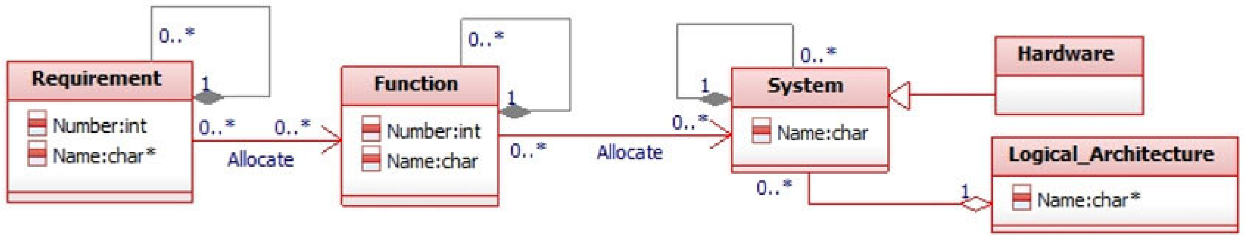


Figure 162. Data model queries

The modeling practitioner should work closely with the SE and safety practitioner to ensure that the data model and queries support their required activities.

9.1.2.4 Modeling standards

Section 8.3 highlights the importance of modeling standards in being able to integrate the different models. Modeling standards (see example in Figure 163) help ensure that the models are created in a consistent manner. The standard data model is then constrained by embedded rules that help achieve data integrity.

The training material should enable the user to complete activities, such as:

- Compose the requirements architecture data model.
- Compose the functional architecture data model.
- Compose the logical architecture data model.
- Create an integrated architecture model, comprising requirements and functional architecture elements that are allocated to elements of the logical architecture.
- Comply with systems architecture modeling standards.

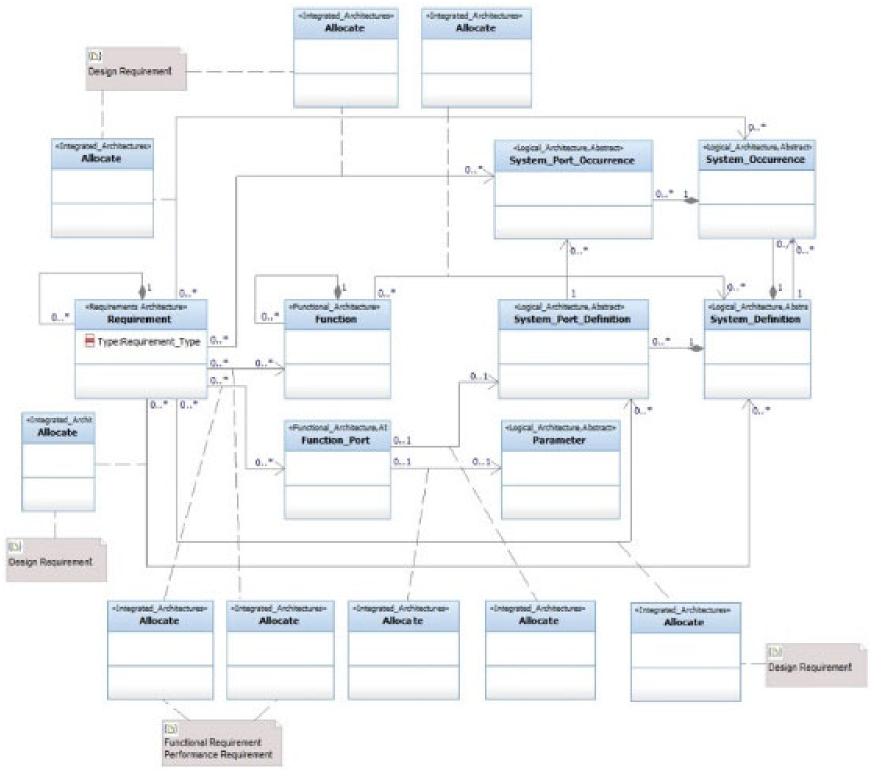

«Integrated_Architectures» Allocate
Design Requirement
«Logical_Architecture, Abstract» System_Port_Occurrence
«Logical_Architecture, Abstract» System_Occurrence
«Requirements Architecture» Requirement
Type:Requirement_Type
«Functional_Architecture» Function
«Logical_Architecture, Abstract» System_Port_Definition
«Logical_Architecture, Abstract» System_Definition
«Functional_Architecture» Function_Port
«Logical_Architecture, Abstract» Parameter
Functional Requirement
Performance Requirement

Figure 163. Model standards example

9.1.3 MBSE/MBSA training

As shown in Figure 164, with the foundational SE and safety and foundational modeling training in place (as needed for the target audiences), effective MBSE/MBSA training can be conducted.

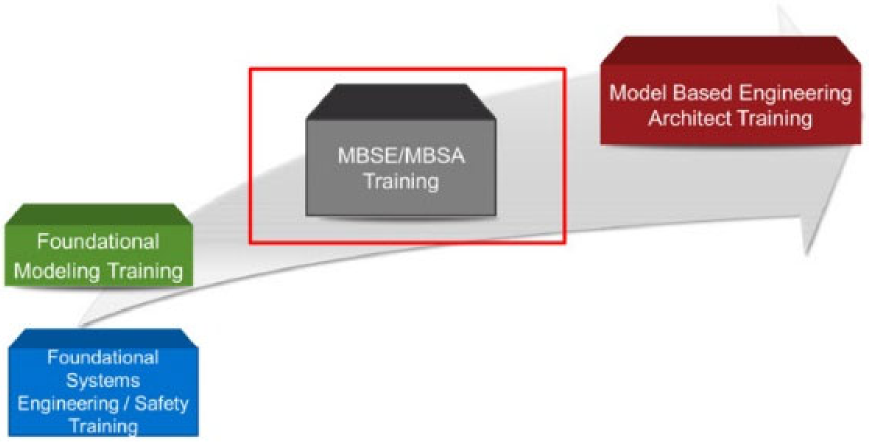

Foundational Systems Engineering / Safety Training
Foundational Modeling Training
MBSE/MBSA Training
Model Based Engineering Architect Training

Figure 164. MBSE/MBSA training

Figure 165 shows an example of recommended MBSE/MBSA training topics, consisting of:

- Introduction to MBSE.
- MBSE tool suite training.
- Introduction to MBSA.
- MBSA tool suite training.

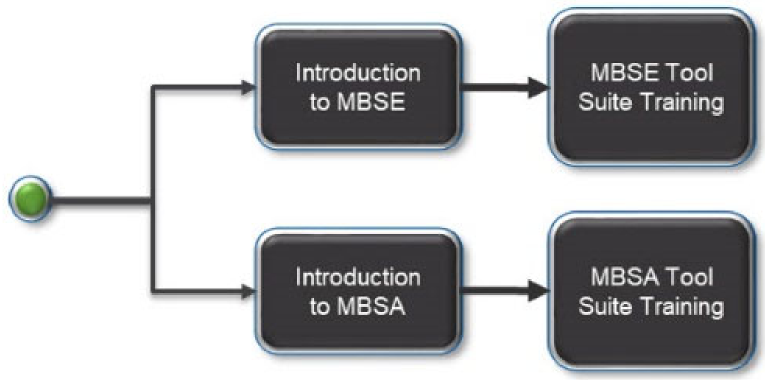


Figure 165. MBSE/MBSA training topics

9.1.3.1 Introduction to MBSE training

The concepts covered in Section 3.2 should be included in an introductory MBSE course. As shown in Figure 166, as part of MBSE, hardware and software are captured, managed, and integrated in an integrated database. The training material should address how MBSE is used to integrate SoS, system, subsystem, component and software requirements with the functional, logical, and physical architecture plus behavior of the system (RFLP+B).

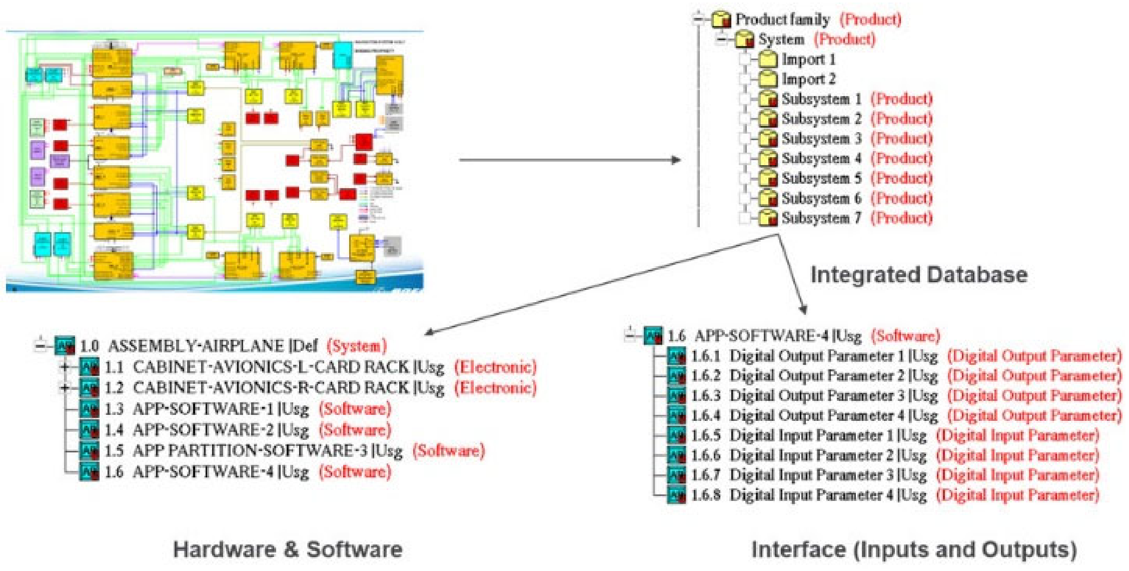

Product family (Product)
System (Product)
Import 1
Import 2
Subsystem 1 (Product)
Subsystem 2 (Product)
Subsystem 3 (Product)
Subsystem 4 (Product)
Subsystem 5 (Product)
Subsystem 6 (Product)
Subsystem 7 (Product)
Integrated Database
1.0 ASSEMBLY-AIRPLANE |Def (System)
1.1 CABINET-AVIONICS-L-CARD RACK |Usg (Electronic)
1.2 CABINET-AVIONICS-R-CARD RACK |Usg (Electronic)
1.3 APP-SOFTWARE-1 |Usg (Software)
1.4 APP-SOFTWARE-2 |Usg (Software)
1.5 APP PARTITION-SOFTWARE-3 |Usg (Software)
1.6 APP-SOFTWARE-4 |Usg (Software)
1.6 APP-SOFTWARE-4 |Usg (Software)
1.6.1 Digital Output Parameter 1 |Usg (Digital Output Parameter)
1.6.2 Digital Output Parameter 2 |Usg (Digital Output Parameter)
1.6.3 Digital Output Parameter 3 |Usg (Digital Output Parameter)
1.6.4 Digital Output Parameter 4 |Usg (Digital Output Parameter)
1.6.5 Digital Input Parameter 1 |Usg (Digital Input Parameter)
1.6.6 Digital Input Parameter 2 |Usg (Digital Input Parameter)
1.6.7 Digital Input Parameter 3 |Usg (Digital Input Parameter)
1.6.8 Digital Input Parameter 4 |Usg (Digital Input Parameter)
Hardware & Software
Interface (Inputs and Outputs)

Figure 166. MBSE introductory material

MBSE introductory material should address creating models both at the systems architecture level and at the system design level.

The training should cover examples of system architecture and system level artifacts that can be created in MBSE, such as:

- Use case diagrams.
- Block definition diagrams.
- Activity diagrams.
- Requirements architecture.
- Functional architecture.
- Logical architecture.
- Analysis models.
- Simulation.

9.1.3.2 MBSE tool suite training

There are several commercially available tools that enable MBSE. This part of the training would be geared toward the tool-specific application of MBSE. Even if the underlying data model is the same (which is not a given), there can be differences in the tool implementation to achieve the same result. Tool-specific training will not be effective without a good understanding of the recommended prerequisites, which provide the "why" of MBSE. The tool-specific training explains "how" the "why" is mechanized for a particular tool suite. Figure 167 is an example of a tool-specific training, showing how a requirement (and its children requirements) could be created.

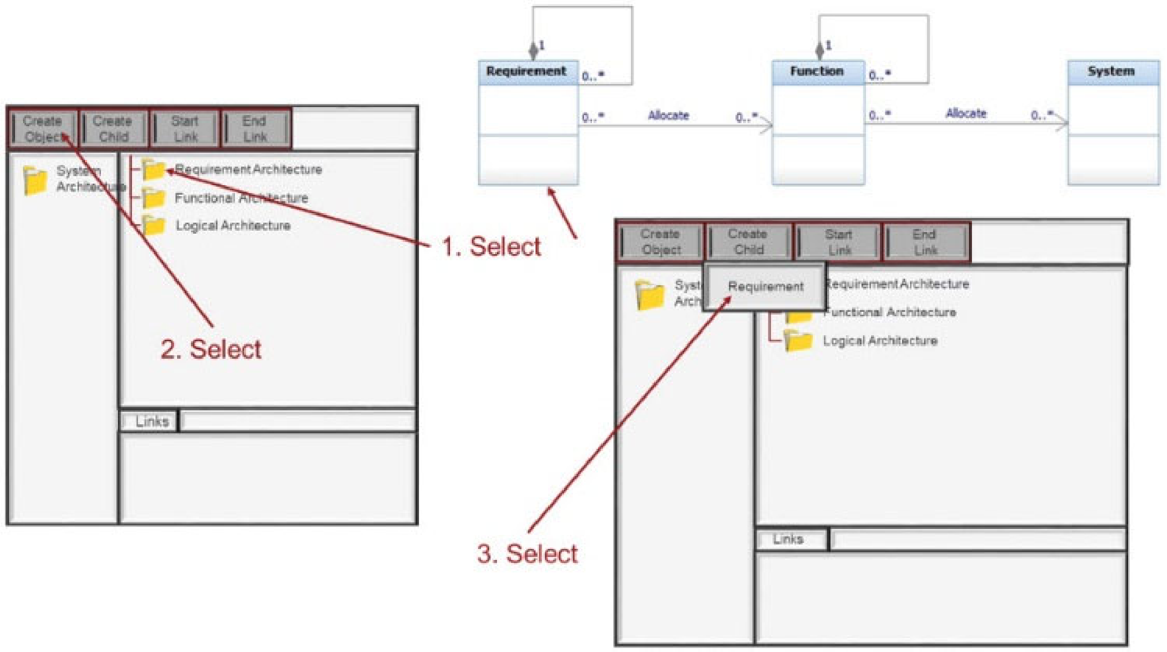


Figure 167. Tool-specific application

9.1.3.3 *Introduction to MBSA training*

The concepts covered in section 3.3 should be included in an introductory MBSA course. As shown in Figure 168, the training should identify how to model key MBSA building blocks, such as functions, FHAs, and systems architectures (with associated safety dependencies). It should emphasize both the top-down analyses (such as cutset generation) and the bottom-up analyses (such as fault propagation).

Important concepts to include in the training material include how to model the following:

- Safety dependencies.
- Redundant sources.
- Loss of function.
- Malfunction (including erroneous).
- Integration of fault propagation and cutset generation.

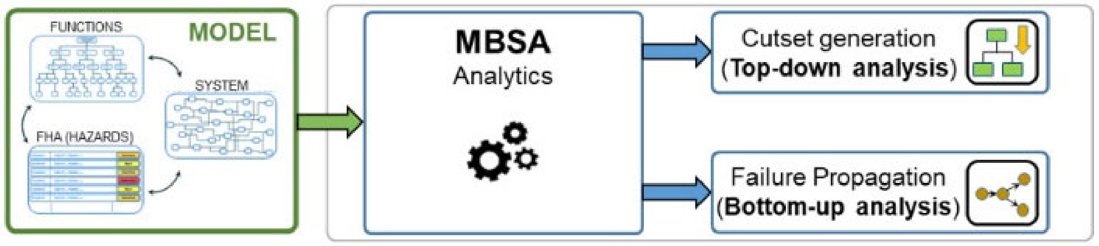


Figure 168. MBSA introductory training material

Interrelationships exist between the different safety analyses. Figure 169 provides an example of the integration between top-down FHAs and bottom-up FMEAs, integrated with the results of the fault trees. The integration of these three analyses, as needed, validate that the top-down and bottom-up analyses are complete, correct, and consistent. In addition, it helps validate that the airplane level hazard categorization is acceptable, based on the failure probability. The MBSA training should address how to ensure that the top-down and bottom-up safety analyses are consistent.

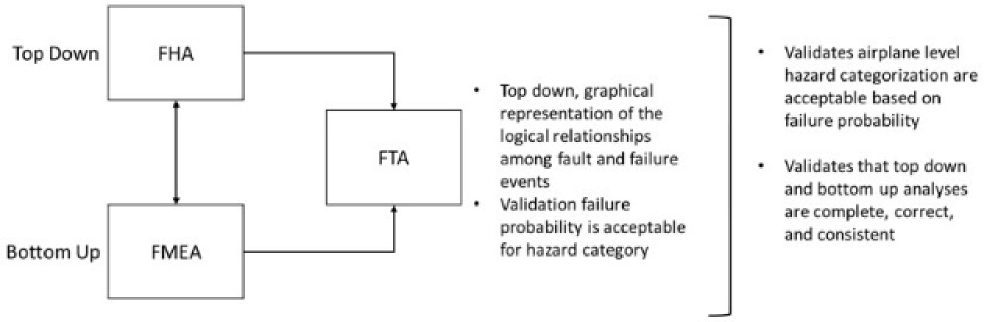

Top Down
FHA
FTA
Bottom Up
FMEA
• Top down, graphical representation of the logical relationships among fault and failure events
• Validation failure probability is acceptable for hazard category
• Validates airplane level hazard categorization are acceptable based on failure probability
• Validates that top down and bottom up analyses are complete, correct, and consistent

Figure 169. Integration of top-down and bottom-up safety analyses

The efficiency of ensuring the consistency between different safety analyses is improved by having a "single source of truth" MBSA model. For example, if a system FMEA hazard classification is more severe than its system FHA, then either the system level FHA needs to be updated or a design/procedure change is needed.

9.1.3.4 MBSA tool suite training

There are several commercially available tools that enable MBSA. In addition, some companies have developed their own proprietary tools. Similar to MBSE, this part of the training would be geared toward the tool-specific application of MBSA. The tool-specific training explains "how" the "why" is mechanized for a particular MBSA tool suite.

9.1.4 MBSE architect training

As shown in Figure 170, the final recommended training material is related to MBSE architect training. It is recommended for architects who are responsible for developing the infrastructure for large-scale systems integration.

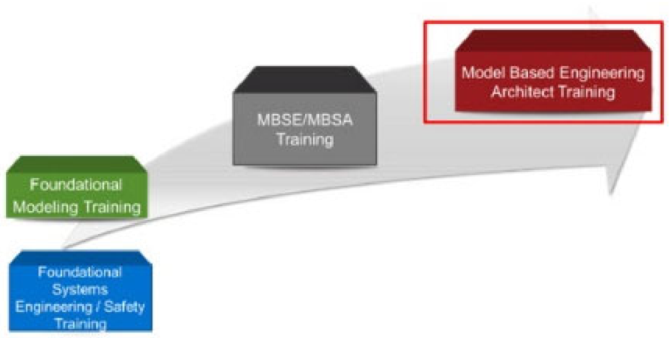

Foundational Systems Engineering / Safety Training
Foundational Modeling Training
MBSE/MBSA Training
Model Based Engineering Architect Training

Figure 170. MBSE architect training

As shown in Figure 171, the MBSE architect is responsible for the integration of the different MBSE activities and the integration of MBSE with MBE.

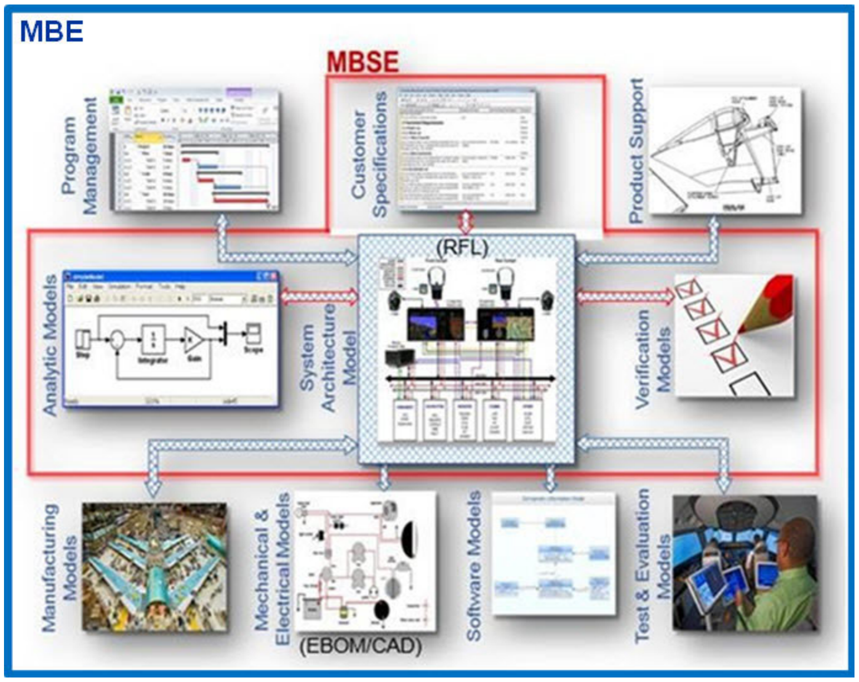


Figure 171. MBSE integration with MBE (source: Boeing)

With MBSE architecture training, one should be able to:

- Describe and create an MBSE architecture specification.
- Develop effective work processes describing necessary tool suite functionality.

- Develop an effective domain model (data model), defining the data, attributes, and relationships necessary to support work processes.
- Develop effective business requirements constraining the domain (data model) within the MBSE tool suite.

9.2 Proposed modifications to existing guidance/standards

The existing guidance/standards describe the objectives that must be satisfied, regardless of whether traditional systems engineering/safety or MBSE/MBSA is being used. Table 19 identifies key industry guidelines/standards and the primary applicable system level, although other guidelines/standards may be used.

Table 19. Key existing industry guidelines

Industry Guideline	**Purpose**	**Primary Applicable System Level**
ARP 4761, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment	Provides guidelines and methods for performing the safety assessment for civil aircraft, including (but not limited to) safety analyses such as FHA, PSSA, and SSA.	Airplane system/subsystem
ARP 4754A, Guidelines for Development of Civil Aircraft and Systems	Provides guidelines on the development assurance process. This includes validation of requirements and verification of the design implementation for certification and product assurance. The development planning elements consist of: ▪ Development. ▪ Safety program. ▪ Requirements management. ▪ Validation. ▪ Implementation verification. ▪ Configuration management. ▪ Process assurance. ▪ Certification. ▪ Software integration process. ▪ Software configuration management. ▪ Software quality assurance process ▪ Certification liaison	Airplane system/subsystem

Industry Guideline	**Purpose**	**Primary Applicable System Level**
DO-254, Design Assurance Guidance for Airborne Electronic Hardware	Provides design assurance guidance for the development of airborne electronic hardware. Key processes include: ▪ Hardware safety assessment. ▪ Requirements capture process. ▪ Validation. ▪ Verification. ▪ Configuration management. ▪ Process assurance. ▪ Certification liaison.	Airborne Electronic Hardware (AEH)
DO-297, Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations	Provides guidance for IMA modules, applications, and systems. The integral processes consist of: ▪ Safety assessment. ▪ System development assurance. ▪ Validation. ▪ Verification. ▪ Configuration management. ▪ Quality assurance. ▪ Certification liaison.	Software AEH
DO-178, Software Considerations in Airborne System and Equipment Certification	Provides guidance for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements. The guidance includes: ▪ Objectives for software life cycle processes. ▪ Activities that provide a means for satisfying those objectives. ▪ Descriptions of the evidence in the form of software life cycle data that indicate that the objectives have been satisfied. ▪ Variations in the objectives, independence, software life cycle data, and control categories by software level.	Software

Industry Guideline	Purpose	Primary Applicable System Level
DO-331, Model-Based Development and Verification Supplement to DO-178C and DO-278A	Contains modification and additions to DO-178C objectives, activities, explanatory text, and software life cycle data that should be addressed when model-based development and verification are used as part of the software life cycle.	Software

Section 4 discussed the interrelationships between the different industry guidance/standards. In general, the interrelationships are well established and clear in the industry guidance documentation. These interrelationships remain the same, regardless of whether the development is occurring via the more traditional document-based approach or via MBSE/MBSA.

Because there is a tendency for systems architecture to become more integrated, there are benefits to emphasizing the need for both vertical and horizontal integration. Figure 172 shows the vertical and horizontal integration at the different levels (airplane, system, and item). This is not meant to imply that the existing guidance is inadequate; it is a question of emphasis.

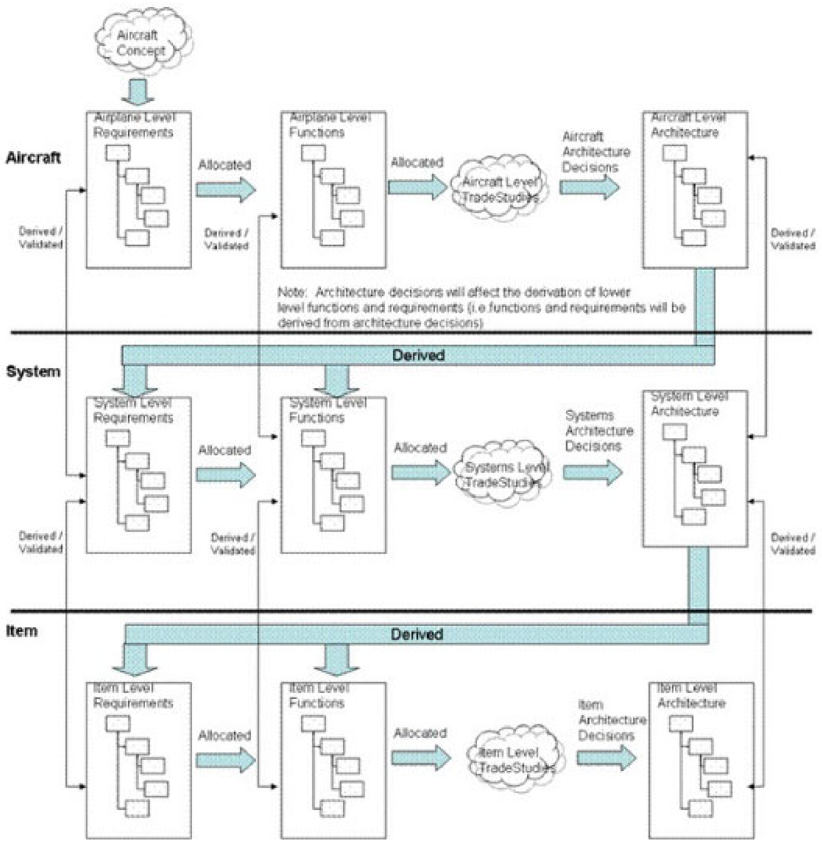

Aircraft Concept
Aircraft
Airplane Level Requirements
Allocated
Airplane Level Functions
Allocated
Aircraft Level TradeStudies
Aircraft Architecture Decisions
Aircraft Level Architecture
Derived / Validated
Derived / Validated
Derived / Validated
Note: Architecture decisions will affect the derivation of lower level functions and requirements (i.e.functions and requirements will be derived from architecture decisions)
Derived
System
System Level Requirements
Allocated
System Level Functions
Allocated
Systems Level TradeStudies
Systems Architecture Decisions
System Level Architecture
Derived / Validated
Derived / Validated
Derived / Validated
Item
Derived
Item Level Requirements
Allocated
Item Level Functions
Allocated
Item Level TradeStudies
Item Architecture Decisions
Item Level Architecture

Figure 172. Horizontal and vertical integration

Requirements errors, omissions, or conflicts can be caused by different factors, such as:

- *Incomplete, incorrect, or missing requirements* – Missing requirements, by definition, will not be incorporated into the design. Incomplete or incorrect requirements may be incorporated into the design, but implementation verification conducted against the incomplete or incorrect requirements should reveal the deficiency, therefore highlighting the importance of validating requirements to be complete and correct.
- *Incorrect implementation of otherwise correct requirements* – In this case, the requirements were validated to be complete and correct. However, the design did not actually meet the requirement. This should be discovered during the implementation verification.
- *Incomplete, inadequate change impact analysis* – In this case, a change is made to an established verified baseline. If a change is not adequately addressed, it could adversely affect the operation of another aircraft system, function, or sub-function. This is sometimes referred to as "change on change." After a change is implemented in one system, it may have unanticipated, unexpected effects on other systems, resulting in the need to drive additional changes.
- *Incomplete, inadequate planning* – In this case, required activities are not conducted, because the planning was incomplete or insufficient.

Table 20 identifies potential improvements to key existing guidance/standards for highly integrated, distributed systems. Section 9.1 discusses how one needs to do sufficient traditional SE/safety before adequate MBSE/MBSA can be achieved. Having incomplete, incorrect, or missing requirements will have an adverse impact, regardless of whether one is using a traditional or model-based approach. This table identifies potential improvements to address root causes.

Table 20. Potential improvements to key existing guidance/standards

Root Cause	**Potential Improvement**
Incomplete, incorrect, or missing requirements	Improved understanding of the integration of new technologies, particularly with respect to timing (e.g., latency, jitter). Improved understanding of requirements in light of potential failure conditions and/or unexpected pilot actions; not identifying the system modes into requirements definition. Improved systems integration focus leading to prevention of requirements' conflict between systems/subsystems boundaries. Improved systems integration focus leading to cumulative effects of otherwise acceptable individual systems level cascading effects. Improved level of formality and process definition in system validation that ensures consistency and completeness. Improved validation of interfaces between systems, commensurate with the inevitable evolutionary nature of this complex problem. Improved validation of the assumptions about the environment.
Incomplete, inadequate change impact analysis	Improved consideration of integration aspects when developing a problem solution, particularly for new, novel, and/or complex systems and new environments. Improved process guidance to facilitate requirements validation for the modification of existing systems.
Incomplete, incorrect planning	Improved consideration of integration aspects when developing a problem solution, particularly for new, novel, and/or complex systems and new environments. Improved process guidance to facilitate requirements validation for the modification of existing systems.

The recommended improvements could help clarify existing guidance/standards. However, the major impediment to successful large-scale implementation of MBSE and MBSA is not related to improving existing industry guidance/standards. The biggest impediments are:

- Data standards.
- Tool capabilities.

9.2.1 Data standards

Data standards are the rules by which data is described and recorded. In order to share, exchange, and understand data, the format, as well as the meaning, must be standardized. Data standards are the foundation for data interchange/exchange/language consistency.

Data standards are needed in order to:

- Enable model architecture reuse by providing an agreed upon way to author the systems architecture independent of tool implementation.
- Improve integration of safety and reliability data with the system model by providing an agreed upon way to author architecture models.
- Improve integration of supplier models by providing an agreed upon way to author design and vertically integrate.
- Improve model quality by providing a standard format for analysis and simulation models.
- Improve requirements quality by providing a standard exchange format for requirements (such as importing/exporting requirements to other engineering domains or exchanging supplier data).
- Enable digital threads throughout design life cycle.

If different data standards are being used, it will be extremely difficult (if not impossible) to combine the data from multiple sources. This becomes increasingly important, as systems architectures move from federated systems to integrated, distributed systems.

Figure 173 shows an example of data interactions and the number of suppliers involved in a relatively simple function: turn on a hydraulic pump via a flight deck switch.

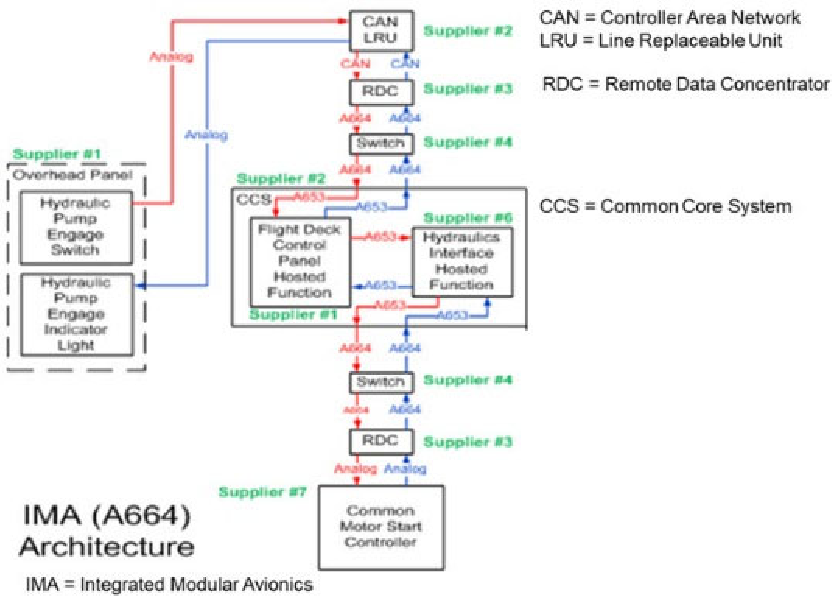

CAN = Controller Area Network
LRU = Line Replaceable Unit
RDC = Remote Data Concentrator
CCS = Common Core System
CAN LRU
Supplier #2
Analog
CAN
RDC
Supplier #3
A664
Switch
Supplier #4
Supplier #1
Overhead Panel
Hydraulic Pump Engage Switch
Hydraulic Pump Engage Indicator Light
CCS
A653
Flight Deck Control Panel Hosted Function
Supplier #6
Hydraulics Interface Hosted Function
Supplier #7
Common Motor Start Controller
IMA (A664) Architecture
IMA = Integrated Modular Avionics

Figure 173. Example of using different data models

This example operation involves equipment from seven different suppliers. Without a common data standard, the potential for different and incompatible data models from seven different suppliers exists. Given that this example has a relatively simple interface, it should be apparent how having different data models could make it significantly difficult to achieve large-scale systems integration.

MBSE/MBSA standards are relatively weak in interoperability. For example, several SysML tools do not interoperate with each other. In addition, the SysML standard organization (OMG) does not enforce or promote interoperability.

The creation and adoption of standard models will be helpful to the advancement of MBSE/MBSA. This should include standard MBSE/MBSA data models, exchange standards, schema, and accompanying composition/aggregation/construction rules.

9.2.2 Tools

This report will continue to remain tool agnostic. However, it is important to highlight (without going into a tool-by-tool comparison), some common weaknesses in MBSE/MBSA tools, including:

- Change and confirmation management.
- Scalability for large amounts of data.
- Supplier integration.

- Integrating supplier data that uses a different tool but is using the same standard.
- Sharing only a portion of the data with the supplier.

MBSE/MBSA tools tend to have been developed independently. Because tools tend to have different data models, the different data models need to be mapped to enable data sharing. Data transfer utilities need to be developed to enable data to be transferred between tools. Developing and managing these different data utilities can become cost prohibitive.

10 Recommendation

To help with MBSE/MBSA implementation across the industry, it would be helpful if the FAA participated with a standard organization to help emphasize the need for the creation of recognized MBSE/MBSA data standard.

11 References

1. SAE ARP 4754A/EUROCAE ED-79A, *"Guidelines for Development of Civil Aircraft and Systems,"* December 21, 2010, covering development assurance processes.
2. SAE ARP 4761, *"Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems,"* 1996, describing safety assessment processes.
3. DO-178B/C, *"Software Considerations in Airborne Systems and Equipment Certification,"* RTCA Inc., Washington, DC, 2001, covering software design assurance processes.
4. DO-254, *"Design Assurance Guidance for Airborne Electronic Hardware,"* RTCA Inc., Washington, DC, April 19, 2000, covering airborne electronic hardware design assurance processes.
5. DO-297, *"Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations,"* RTCA Inc., Washington, DC, November 8, 2005, covering integrated modular avionics.
6. DO-331, *"Model-Based Development and Verification Supplement to DO-178C and DO-278A,"* RTCA Inc., Washington, DC, December 13, 2011, covering use of model-based development and verification in the software lifecycle for software that is produced in accordance with DO-178C.

7. SAE ARP 4754/EUROCAE ED-79, *"Certification Considerations for Highly Integrated or Complex Aircraft Systems,"* 1996, likewise covering development assurance processes.

8. SAE AIR6110, *"Contiguous Aircraft/System Development Process Example,"* February 5, 2020, covering a contiguous example of the aircraft and systems development for a fictitious aircraft design.

9. SAE AIR6218, *"Constructing Development Assurance Plan for Integrated Systems,"* October 22, 2018, covering the crucial elements to be considered when constructing the development assurance plans as described in Chapter 2 of ARP 4754A.

10. Advisory Circular 20-174, *"Development of Civil Aircraft and Systems,"* Sept 30, 2011, recognizes ARP 4754A as an acceptable method for establishing a development assurance process.

11. Advisory Circular 20-170, *"Integrated Modular Avionics Development, Verification, Integration and Approval Using RTC/DO-297,"* October 28, 2010, covering guidance on how to obtain FAA airworthiness approval for the development, verification and integration of an integrated modular avionics system for installation into an aircraft or engine.

12. Advisory Circular 20-115D, *"Airborne Software Development Assurance Using EUROCAE ED-12 and RTCA DO-178,"* July 21, 2017, covering acceptable means of compliance for the software aspects for airborne systems and equipment.

13. Advisory Circular 20-152, *"Design Assurance Guidance for Airborne Electronic Hardware,"* July 5, 2005, covering acceptable means to gain FAA approval for showing the equipment design is appropriate for its intended function.