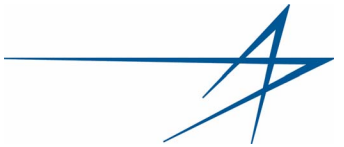


ARTAMIS™: Open Source and Extensibility in an Embedded Mission System

Alan Hohn

Lockheed Martin MST

1801 New York 17C

Owego, NY 13827

Alan.M.Hohn@lmco.com

Contents

- Fixed Wing Airborne Surveillance
- Service Oriented Architecture
- Java Enterprise
- Data Management
- Real-Time Integration
- Web Interfaces
- External Application Integration

80 Years of Maritime Surveillance

P-3 Orion

758 built

A B C Updates & Variants

S-3 Viking

187 built

A B Updates & Variants

C-130 Hercules

>2,400* built

Maritime Surveillance, AEW&C Variants

* C-130 still in production

P-2 Neptune

>1,000 built

Ventura

>3,000 built

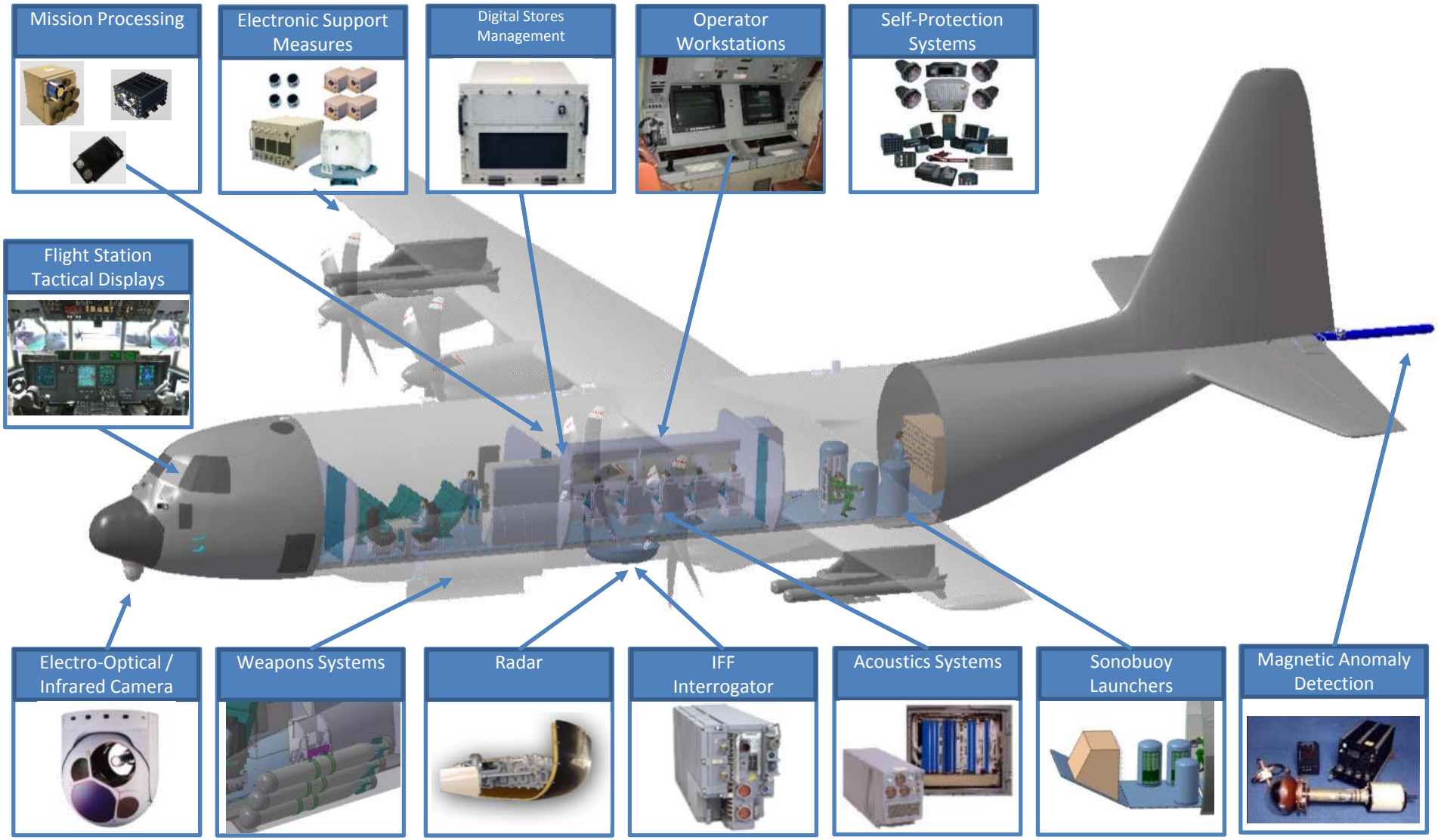
Hudson

>2,900 built

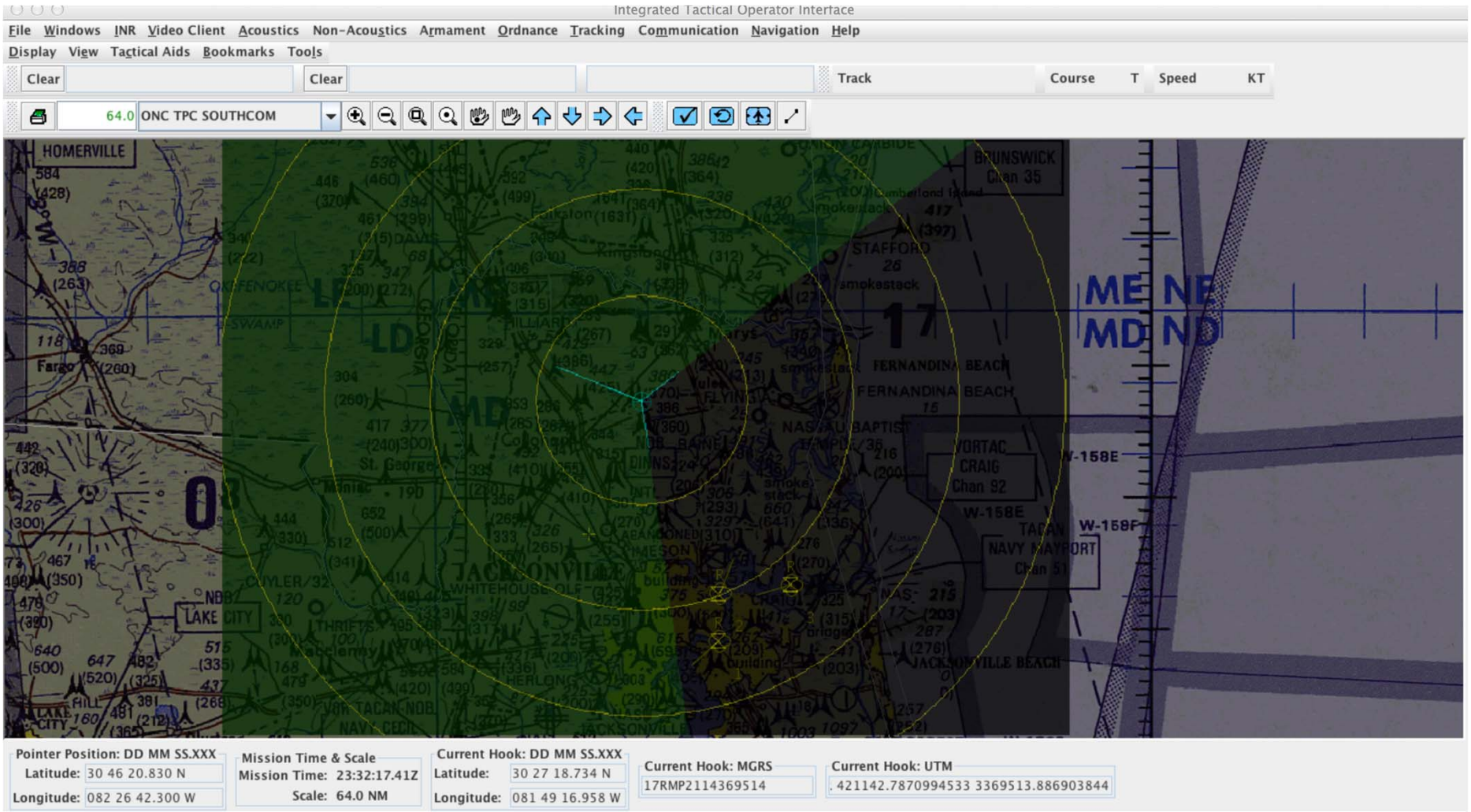
1930 1940 1950 1960 1970 1980 1990 2000 2010

Lockheed Martin Has Long History In Airborne ASW and Maritime Surveillance

Airborne Maritime Surveillance
C-130 SeaHerc Example



ARTAMIS™ Operator Display



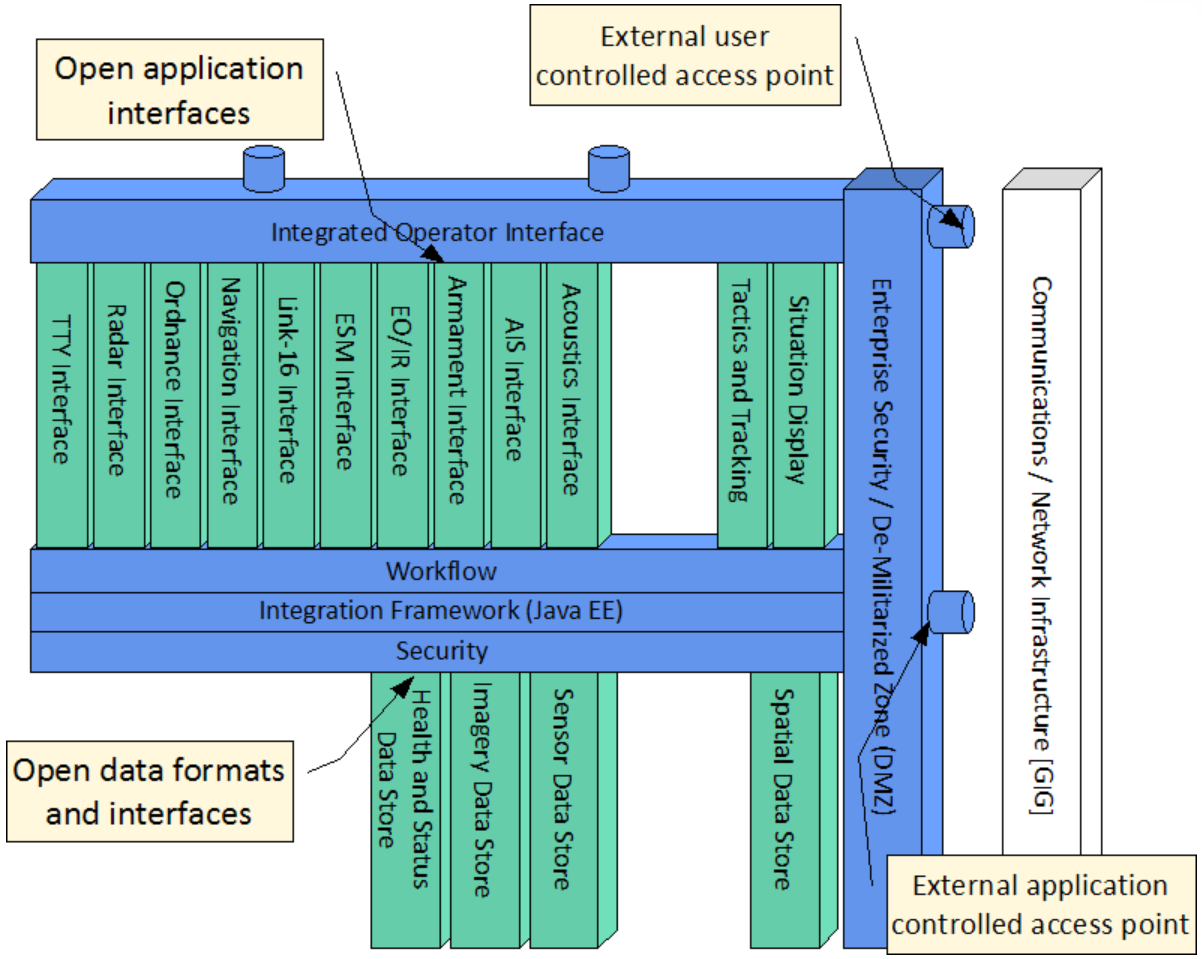
ARTAMIS™ History

- Initial architecture definition in 2005
- Created as the successor to an Ada-based mission system used in U.S. Navy P-3s starting in 1994
- Full-scale development under international P-3 contract started in 2009
- Delivered to aircraft in 2012
- ARTAMIS™ product name created in 2013

ARTAMIS™ Approach

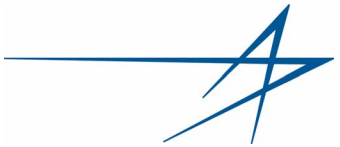
- Enterprise Architecture
  - Software/hardware loosely coupled with communications over high speed network backbone
  - N-tier architecture (presentation, application, data, I/O)
- Distributed System
  - Stand-alone workstations with local software execution
  - Support scalability of processing, type and number of workstations
- Re-architecture of tested, deployed mission system software
  - ASW/ASuW mission capabilities translated from Ada to Java and re-hosted in Web service environment
  - Data management modules migrated to COTS database
- Increased use of commercial off-the-shelf software and hardware
  - Java EE framework isolates applications from processing hardware
  - Services organized around subsystems and sensors

Service Oriented Architecture



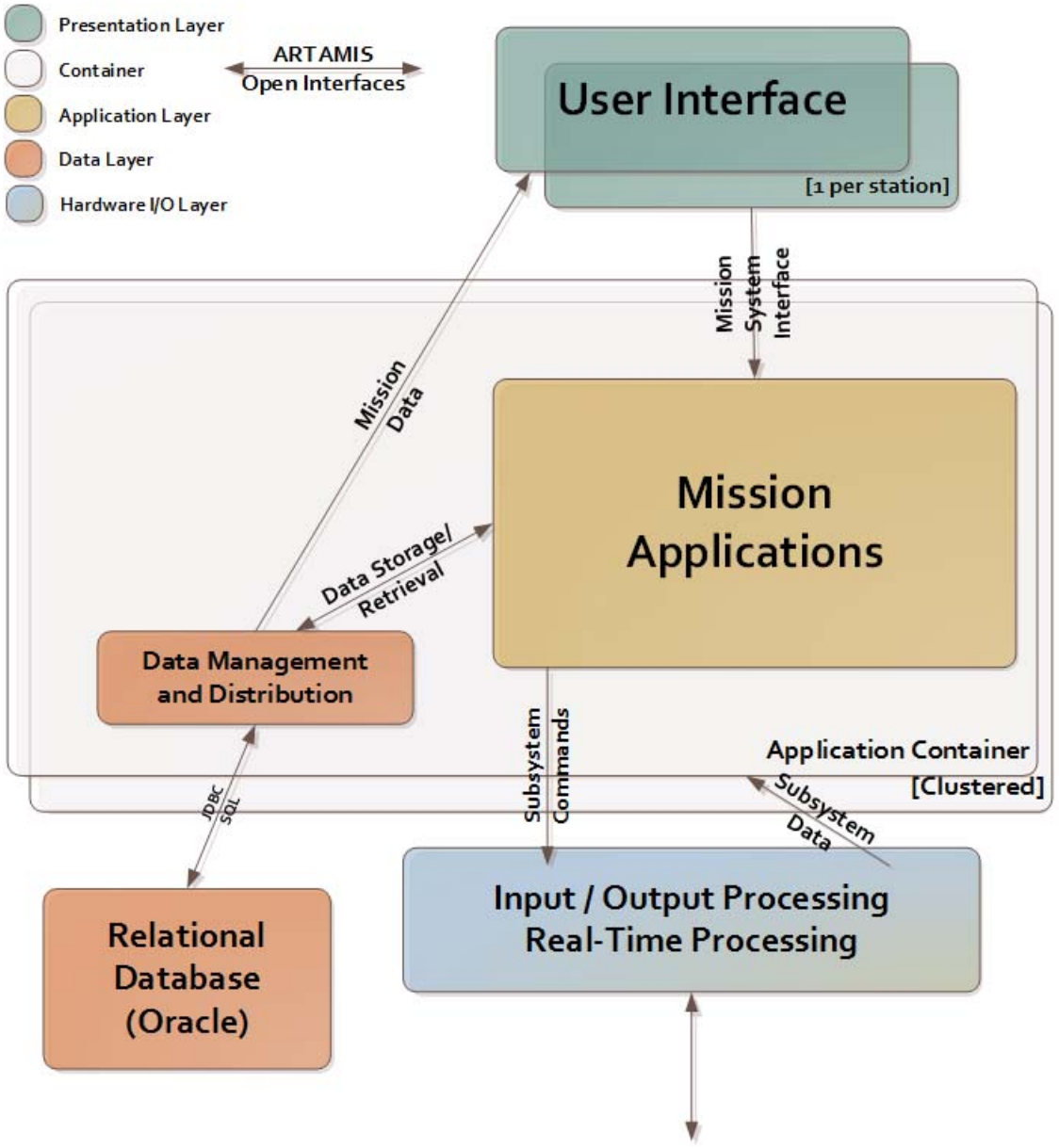
Large-grained components and clean interfaces are the hallmarks of a SOA

ARTAMIS™ Service Development

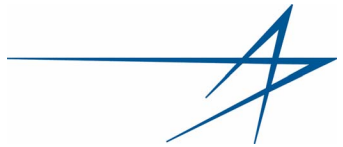


- ARTAMIS™ services are organized around the sensors and subsystems available
  - Services are composed at build time to support different platforms with different installed equipment
- Service development started with a “business” interface defining the service contract
  - This is a Plain Old Java Interface (POJI) that just specifies methods, parameters, return types and exceptions
  - This one interface is leveraged for Java remote invocation as well as REST web services

ARTAMIS™ Enterprise Architecture

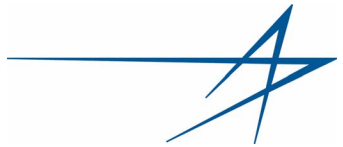


Java Enterprise Edition



- Large programs need to solve many of the same problems, independent of the *business logic* of the program itself
  - Examples include remote invocation, load balancing, failover, database connectivity, messaging, lookup
  - Building these *cross cutting concerns* directly into the business logic would make it harder to reuse them and harder to reuse the business logic
- Java EE is a set of technologies that provides these cross-cutting concerns, ideally with minimal impact on business logic code
  - This is done by packaging bits of business logic code together and handing them over to a framework that runs them
  - The business logic code just focuses on business logic
  - Java EE enforces rules on the use of threads, constraints the object's life cycle, and complicates the use of statics, shared resources, and so forth, enabling multiple applications to work side-by-side in the same container
- ARTAMIS™ uses JBoss Enterprise Application Platform (EAP), a supported version of the Wildfly open source Java EE application server

Java EE Shared Resources



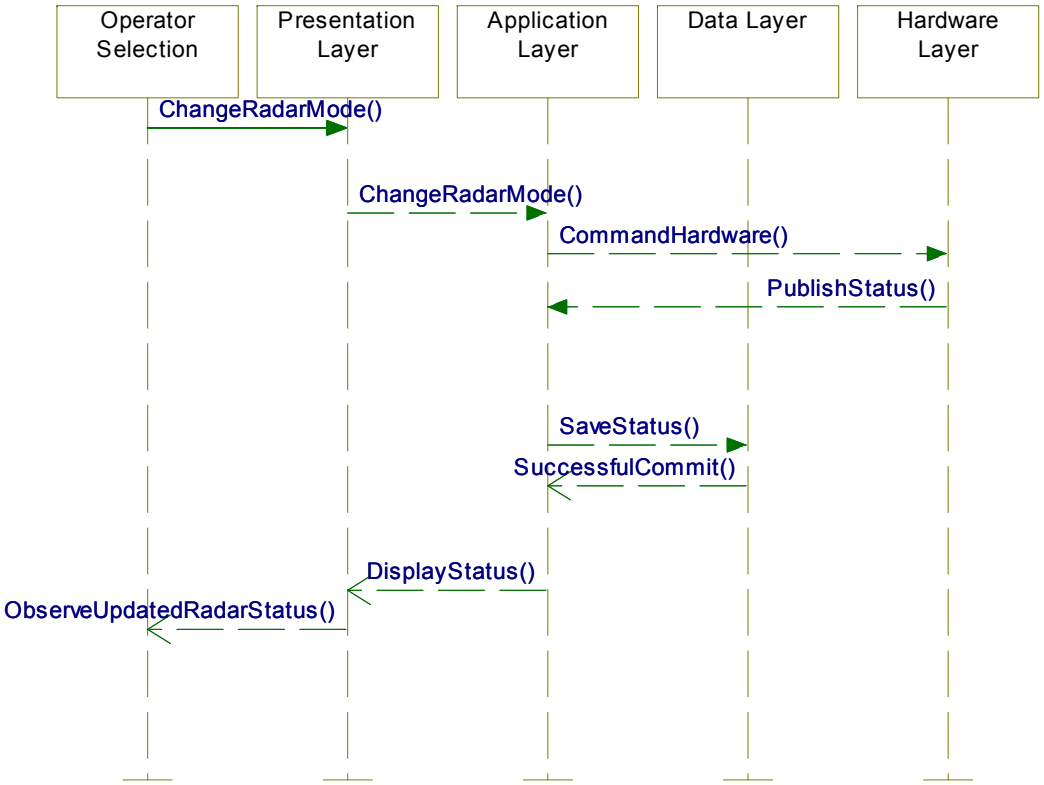
- The Java EE application server simplifies access to commonly-used resources
  - Remote invocation of Java EE applications
  - Timer service
  - A directory enabling components to find each other
  - Database connections
  - Publish-subscribe messaging
- The Java EE application server handles failover, connection pooling, transaction management, and other facilities required to write robust, high-performing applications

Data Distribution Architecture

- Data follows a ‘synchronous-down, asynchronous-up’ distribution architecture
- Operator commands are synchronous through the application layer to the hardware interface layer
  - Permits immediate reporting of error conditions
  - Simplifies threading model and ordering of operations between multiple stations
- Sensor data is distributed asynchronously through the application layer to the presentation layer
  - Allows greater flexibility in adding new operator displays, workstations
  - Enables publish-subscribe messaging between applications and from applications to operator, supporting open architecture

Example Sequence Diagram
Change Radar Mode

- Delegate Pattern - send request to application server
- Service invoked and processing starts
- Command forwarded to the hardware for formatting message to subsystem
- Command executed on subsystem
- Status published to the mission system and processed



ARTAMIS™ Real-Time Software

- ARTAMIS™ real-time requirements relate to interaction with sensors and subsystems
  - Allows encapsulation of real-time processing solely in the I/O services software component
  - No real-time requirements in application or presentation tiers
- Real-time requirement to send launch command to Digital Stores Management System (DSMS) upon determination of fly-to-point (FTP) capture
  - Includes ARINC 575 and MIL-STD-1553B interfaces
  - Includes aircraft position smoothing, capture determination, and armament and ordnance interfaces
  - Selected real-time Java and real-time Linux for commonality with other components and re-use of algorithms

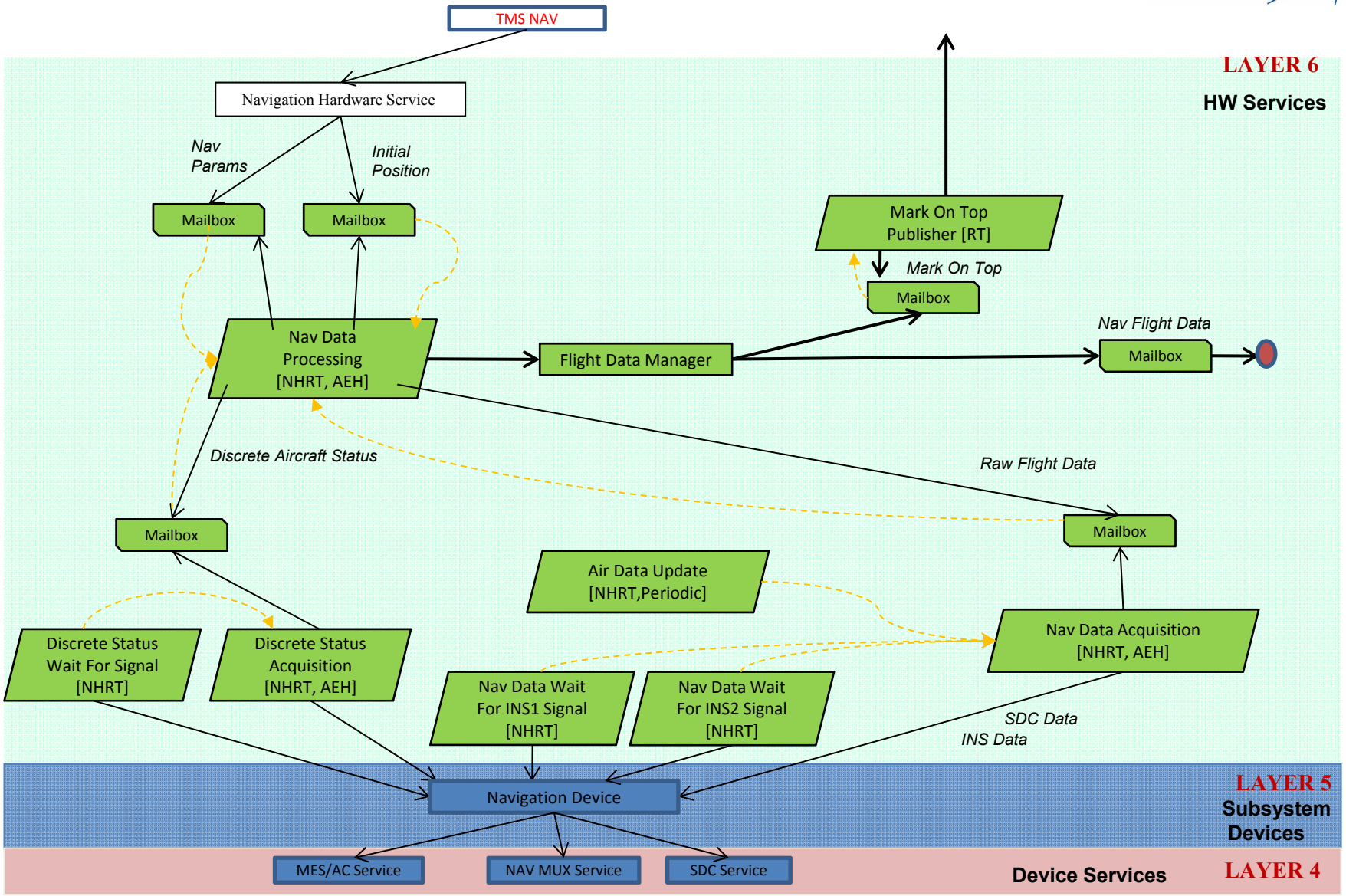
Real-Time Java

- The real-time specification for Java (RTSJ) provides for deterministic behavior from Java programs
  - Requires a compatible Java Virtual Machine
  - Must run on a compatible real-time operating system
  - Requires the use of explicit real-time threads
- The most significant barrier to real-time Java is preemption by the garbage collector
  - There have been experiments with deterministic garbage collection, but only to a “high degree of confidence”
  - To truly guarantee hard real-time, the RTSJ allows the use of real-time threads that cannot be preempted by the garbage collector (and can preempt it)
  - These threads must manage memory differently from normal Java programs

Real-Time Java Memory Management

- Normal Java programs create objects at will
  - Object allocation occurs in one or more heap spaces
  - These objects are garbage collected by the Java virtual machine in a background process once they are no longer “reachable” from live code
  - If heap spaces are exhausted, threads creating objects may block while garbage collection is performed
- The RTSJ provides for a NoHeapRealtimeThread (NHRT) that may preempt the garbage collector
  - The NHRT may not allocate or reference objects in heap space
  - All allocation is done in Immortal Memory (never freed) or Scoped Memory (freed in its entirety when out of scope)
- NHRTs can coexist with normal Java threads in a single program and typically communicate using shared queues

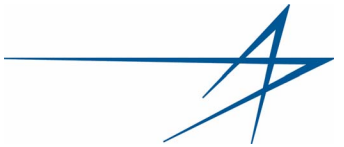
Example Real-time Processing Acquire Navigation Data



NHRT = No-Heap Realtime Thread

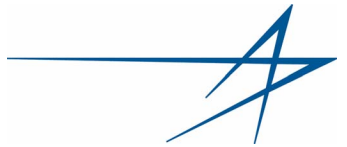
AEH = Asynchronous Event Handler

ARTAMIS™ Real-time Lessons Learned



- The RTSJ is a viable path for deterministic processing in Java
  - < 1ms latencies for NHRT processing, even on a fully loaded system
  - Common algorithm implementations were successfully shared between real-time and non-real-time code
- Real-time Java code is not very much like regular Java code
  - Regular Java code can allocate very many short-lived objects because the garbage collector is optimized for this kind of use
  - Real-time Java code must be parsimonious with object allocation or have very short-lived memory scopes
  - Code that was not written for real-time Java will probably have to be analyzed and modified before it be efficient
- Close monitoring of immortal memory usage is essential under a variety of realistic use cases to avoid unexpected "leaking" of heap objects into immortal memory

Web Interfaces



- JBoss EAP includes Resteasy, a Java library for building HTTP REST web services
- REST-enabling the service interfaces is done by annotating the methods in the business interface
- The open source Jackson library automatically translates requests and responses to and from JavaScript Object Notation (JSON)
- The open source Cometd library provides a publish-subscribe messaging capability using HTTP long polling or WebSockets

ARTAMIS™ Partner Program Application Customization

- Third-party applications can supply data (tracks, symbology) to the user's situation display
- Third-party applications can add pull-down and right-click menu items, including changes to existing mission system objects
- Applications are notified of operator input by publish-subscribe messaging

Summary

- ARTAMIS™ brings enterprise technology and open source software into the airborne surveillance domain
- Open source software is used to greatly simplify the development effort involved in integrating new applications and capabilities into the system
- Real-time and low-level interfaces are encapsulated to provide deterministic, reliable performance
- Third-party integration is explicitly provided using the ARTAMIS™ Partner Program