Open Tool Kit for Mission Critical Systems

Douglas Wells
The Open Group
d.wells@opengroup.org
http://www.opengroup.org/ar

Portions of this material are based upon work supported by the Naval Surface Warfare Center Dahlgren Division under contract No. N00178-03-C-3001. (COR: Larry M. Gunter)

The Open Group

- "The Open Group is …
  - An international vendor- and technology-neutral consortium committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower time, cost and risk associated with integrating new technology across the enterprise."
- "Our Mission is to drive creation of Boundaryless Information Flow by:
  - working with customers on requirements, policies, and best practices;
  - working to develop consensus and interoperability, integrated specifications, and open source technologies;
  - offering operational services to consortia; and
  - certifying approved products and processes."

The Open Group: Research Group

- Charter: sustain corporate competence in evolving technologies
- Role: transfer research technology to commercial world
  - Work with researchers to develop new technologies
    - DARPA/AFRL
  - Work with early adopters on trial-use applications
    - MITRE, Honeywell Space Systems, NSWC
  - Work with vendors to productize results
    - OS: HP/Convex, Intel Paragon OSF/1®, MkLinux, Mac OS X™, IBM Workplace OS
    - Comm: DASCOM Secure DCE Firewall, Novell

The Internet QoS Challenge

- The Internet provides the opportunity to conduct business at vastly increased scales using a shared-cost infrastructure
- However, to take advantage of this opportunity, companies are “increasingly dependent on large-scale distributed systems that operate in unbounded network environments”(IEEE *Internet Computing* 11/99)

QoS Opportunity

- As value of transactions on networks grow, companies will seek guarantees of dependability, performance, and efficiency for distributed applications and networks
- To provide adequate levels of service to customers, companies need the same level of assured operation as they transition from the mainframe “Glass House”
  - End-to-end performance
  - Availability and Fault Resilience
  - Adaptivity to changing load and network conditions
- Thus, companies are requiring and negotiating Service Level Agreements (SLAs) based on various QoS metrics, eventually to include application-specific qualities of service (AQoS)

Military Weapons Systems Challenge

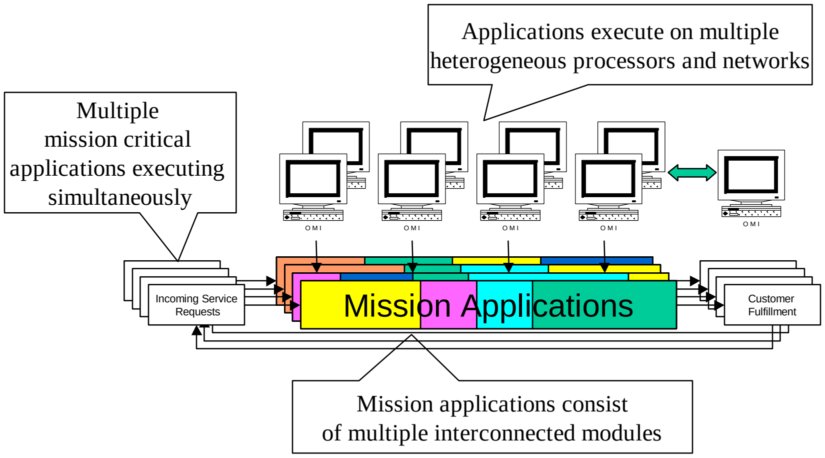
- Reduce lifecycle costs of building and maintaining mission critical computer-based systems, particularly shipboard and mobile
- Increase capabilities by leveraging advances in commercial market to address requirements of real-time, fault-tolerant tactical systems
- Assure supportability and ability to meet evolving mission goals by reducing system complexity
- Support network-centric model (Boundaryless Information Flow)

Reduction to Practice

- Open systems based on open standards
  - UNIX®, CORBA®, Web Services, etc.
- Real-time and/or fault-tolerant COTS components when available
- Use of specialized components to supplement COTS products to mitigate deficiencies
- Instrumentation and manageability interfaces
- Use of component framework for real-time, fault-tolerant, mission critical systems
  - Provide coherent architecture
  - Allow selection of components based on requirements
  - Extends, doesn't replace, commercial architectures
- Leveraging of COTS QoS-management capabilities

The Managed QoS Environment

Slide courtesy of Al Sandlin, SPAWAR, U.S. Navy

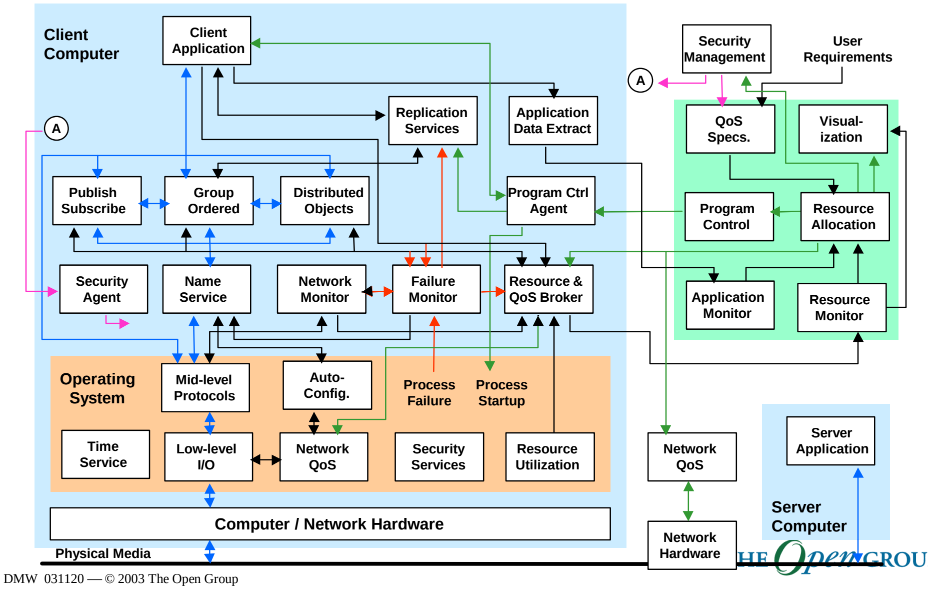


Applications execute on multiple heterogeneous processors and networks

Multiple mission critical applications executing simultaneously

Mission applications consist of multiple interconnected modules

HiPer-D QoS REFERENCE ARCHITECTURE



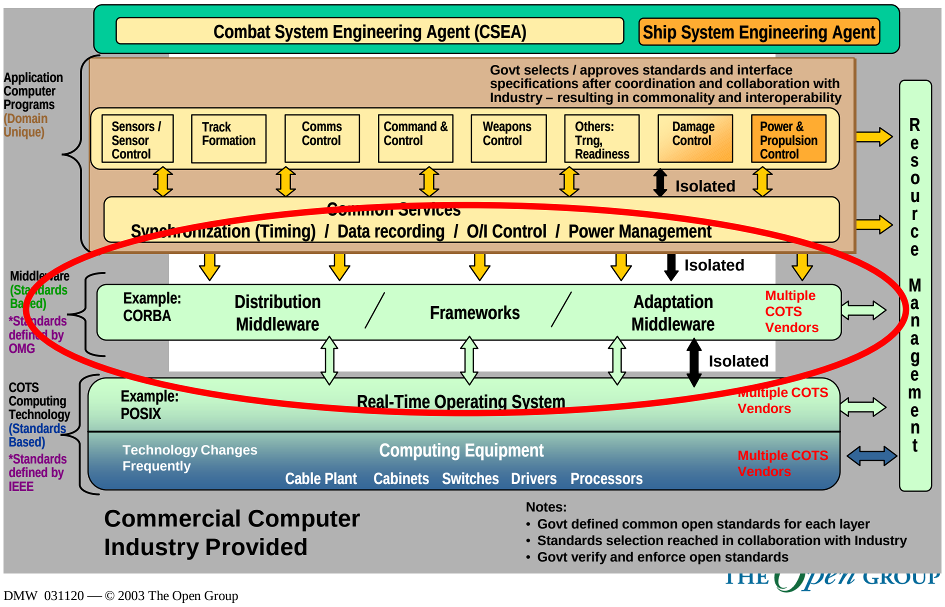
The U.S. Navy Problem

- Building hard real-time, fault-tolerant computer applications for shipboard weapons systems is difficult and expensive
  - Traditional solution has been for contractor to use purpose-built software and hardware
  - Recent systems reduced costs via use of COTS hardware
  - Continued use of traditional software methods has resulted in “COTS refresh” lifecycle surprise — expensive upgrades

A Potential Solution

- Use of COTS software could reduce costs by increasing portability and reducing complexity, but is difficult to use
    - Heterogeneity: multiple computer platforms (HW/SW), lifecycle upgrades (HW/SW), conflicts for shared resources
    - Multiple, independently developed software systems: JMCIS, ATWCS, etc., etc.
    - Scarcity of real-time and/or fault-tolerant COTS software
    - Limited availability of software developers trained in military-specific technologies

U.S. Navy Open Architecture (OA)



Combat System Engineering Agent (CSEA)

Ship System Engineering Agent

Application Computer Programs (Domain Unique)

Govt selects / approves standards and interface specifications after coordination and collaboration with Industry – resulting in commonality and interoperability

Sensors / Sensor Control

Track Formation

Comms Control

Command & Control

Weapons Control

Others: Trng, Readiness

Damage Control

Power & Propulsion Control

Isolated

Common Services

Synchronization (Timing) / Data recording / O/I Control / Power Management

Isolated

Middleware (Standards Based)

*Standards defined by OMG

Example: CORBA

Distribution Middleware / Frameworks / Adaptation Middleware

Multiple COTS Vendors

Isolated

COTS Computing Technology (Standards Based)

*Standards defined by IEEE

Example: POSIX

Real-Time Operating System

Multiple COTS Vendors

Technology Changes Frequently

Computing Equipment

Cable Plant Cabinets Switches Drivers Processors

Multiple COTS Vendors

Resource Management

Commercial Computer Industry Provided

Notes:

- Govt defined common open standards for each layer
- Standards selection reached in collaboration with Industry
- Govt verify and enforce open standards

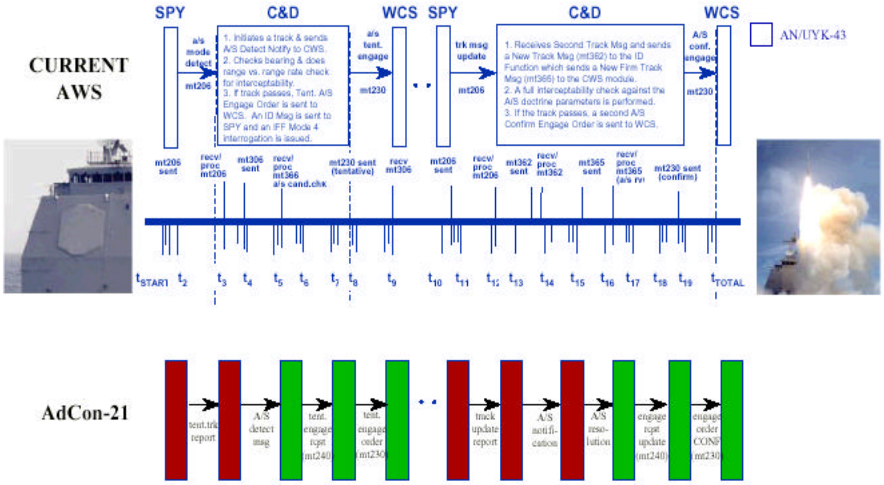
Risks

- Introduction of COTS hardware
  - Reduced cost of initial deployment
  - But, resulted in high upgrade costs
- Can we repeat this benefit for software—without the negative side effect?
- Early indications are positive
  - HiPer-D
  - Boeing’s Boldstroke
  - MITRE/The Open Group JavaOne Ballsorter Demo
- Outstanding issues:
  - What is the effect of product changes due to the evolving marketplace?
  - Can the initial successes be replicated in large scale projects?

HiPer-D: An Example Context

- High-Performance Distributed (HiPer-D) Project (at NSWC in Dahlgen, VA) is applying COTS-based, distributed computing techniques to prototype shipboard weapons systems
- One major objective is capacity scalability (“Load Invariant Computing”)
- AAW (ship self-defense) is “hard” real-time
  - Mandated timing requirements
  - Mandated failure recovery requirements
  - “Auto-Special Doctrine” execution path

Why DD-21 Needs Assured Response: SPY Radar Auto-Special Time-Line

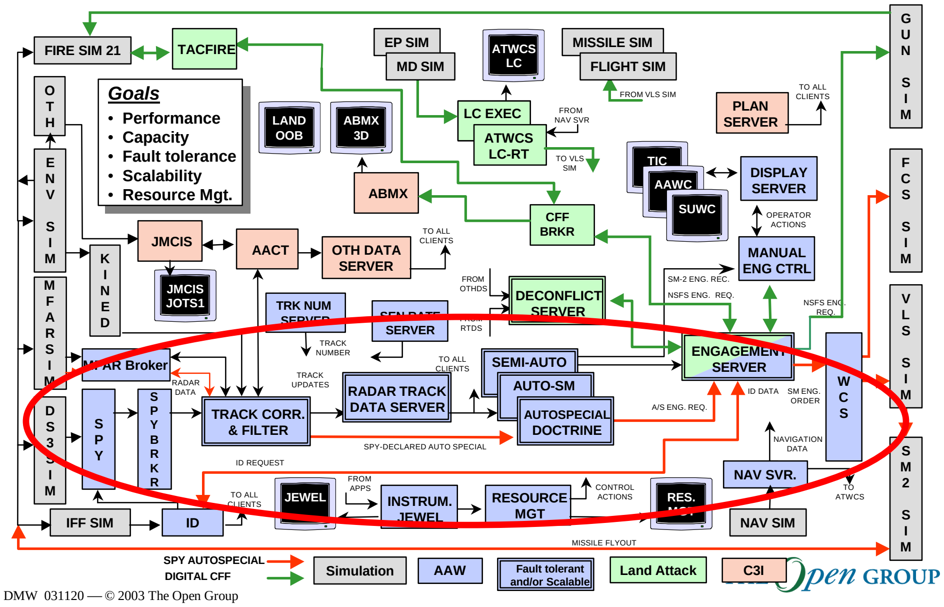


1. Initiates a track & sends A/S Detect Notify to CWS.
2. Checks bearing & does range vs. range rate check for interceptability.
3. If track passes, Tent. A/S Engage Order is sent to WCS. An ID Msg is sent to SPY and an IFF Mode 4 interrogation is issued.

1. Receives Second Track Msg and sends a New Track Msg (mt362) to the ID Function which sends a New Firm Track Msg (mt366) to the CWS module.
2. A full interceptability check against the A/S doctrine parameters is performed.
3. If the track passes, a second A/S Confirm Engage Order is sent to WCS.

DEMO 99 FUNCTIONAL BLOCK DIAGRAM

(From NSWC HiPer-D Project)



Goals

- Performance
- Capacity
- Fault tolerance
- Scalability
- Resource Mgt.

Software Architecture—Paths

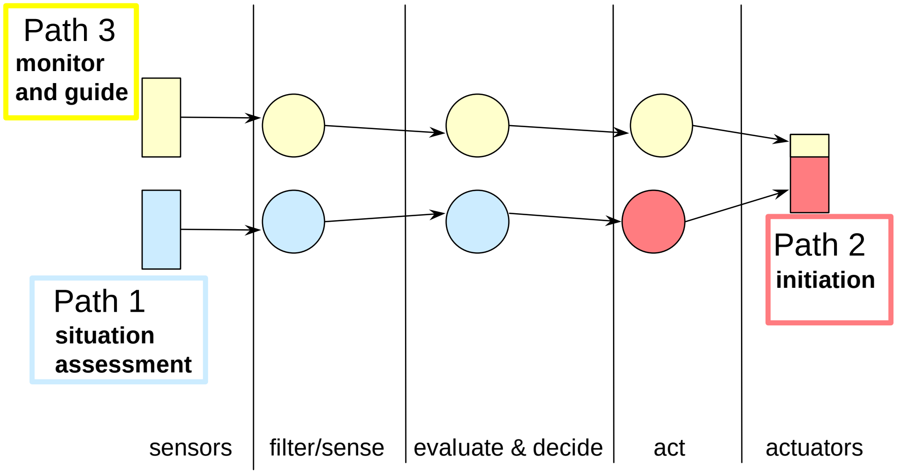


Diagram courtesy of Lonnie Welch, Ohio U.

Path 3
monitor and guide

Path 1
situation assessment

Path 2
initiation

sensors | filter/sense | evaluate & decide | act | actuators

Open Tool Kit¾ Overview

- Issue: Use of COTS software components to gain capability and reduce cost on shipboard weapons systems requires real-time, fault-tolerant, standards-based integration tools
- Goal: Tool kit to reduce cost, development time, and number of defects throughout tactical systems lifecycle by use of COTS languages and fault management components
- Required Capabilities:
  - Reusable real-time, fault-tolerant software components
  - Ability to leverage COTS software for use in real-time, fault-tolerance applications
  - Facilities to manage QoS at system, middleware, and application levels

Open Tool Kit¾ Enabling Technologies

- Software framework and tool-kit for real-time, fault-tolerant distributed systems
  - Leveraging open systems and standards
    - UNIX/POSIX/Linux®, CORBA, CIM, AIC, ARM
  - Leveraging COTS software capabilities, and marketplace training and knowledge
- Real-time Java™ language and environment.
- Real-time group communication
- **Goal: The open tool kit provides a context within which multiple vendors can insert components that will interoperate within the infrastructure of mission critical systems.**

- Acronyms:
  - CIM: Common Information Model
  - AIC: Application Instrumentation and Control
  - ARM: Application Response Measurement

Java™

- Heterogeneity: “ Write once, run anywhere”
- Widely used
  - Availability of trained software developers
  - Availability of software packages
  - Active development of additional capabilities under Java Community Process
- Shares many Ada attributes
  - Object-orientation, type safety
  - Eliminates accidental buffer overflows
  - Multithreaded programming model
- Problem: Java’s inherent use of traditional garbage collection incompatible with hard real-time systems

Real-Time Java™

- Extends Java capabilities to real-time and embedded systems
  - Mechanism to avoid garbage collector interference
  - Direct access to hardware devices
  - Increased resolution in clocks/timers
- Specifications based on NIST-sponsored requirements process, including commercial and military participants
- Characteristics:
  “ Write once carefully, run anywhere conditionally”
- Two (Competing?) Standards
  - Real-Time Specification for Java (RTSJ) — Sun
  - J Core — J Consortium®

Critical Issues

- Can we use real-time Java to develop complex, distributed, real-time mission critical applications?
- Performance⇒
  - incremental/native compilation
- Resource Management⇒
  - At least as good as legacy systems
  - JVM provides useful failure domain boundary
- Garbage Collection⇒
  - Scoped Memory
  - “Real-time” (incremental) garbage collectors
- Function⇒
  - RTSJ + distribution
  - RTSJ + J2EE

How do we design systems to meet deadlines that involve multiple nodes?

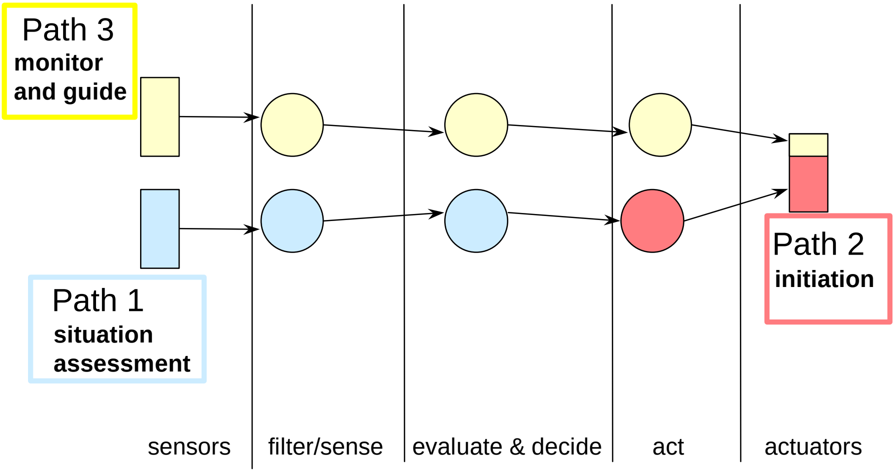
- DRTSJ (JSR-50)⇒Someday
- Need to run under control of scheduler
- Avoid/defer garbage collection
  - No Heap Real-Time Threads
  - Scoped Memory
- Need to use “standard” packages
  - java.net/java.nio
  - Can we use these standard packages in scoped memory environments?
  - Do they use “long-term” (static) variables?

Group Communications

- Mechanism for communicating among groups of hosts in a computer network
  - Includes recovery mechanism for host failure
  - Leverages reliable multicast mechanism (HW/SW)
  - Host failure detection is based on time-out
- Group members share state knowledge in order to
  - Partition load for scalability
  - Provide redundancy for fault tolerance
  - Or both
- Problem: “COTS” group communication packages are optimized for throughput, not for guaranteed response time

Software Architecture—Paths [redux]

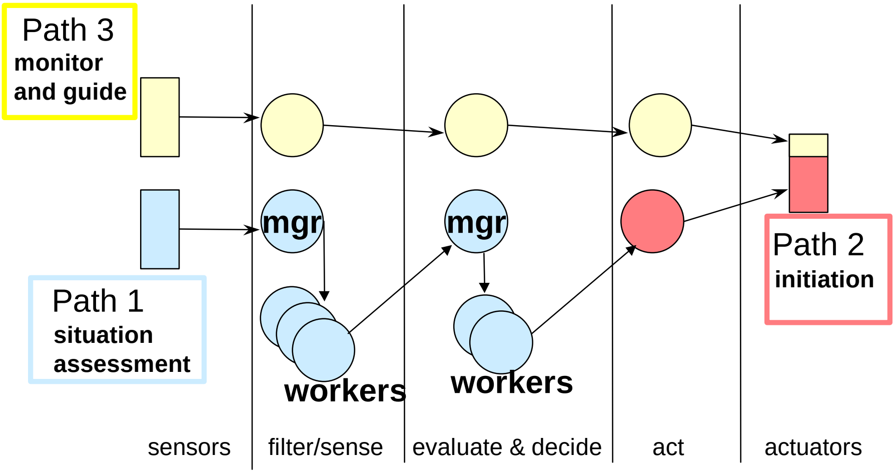
Diagram courtesy of Lonnie Welch, Ohio U.



THE Open GROUP

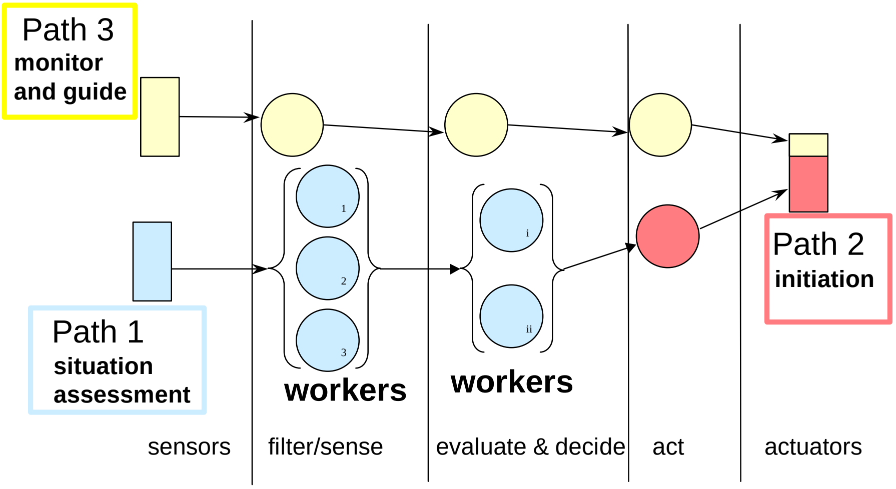
Scalability

Diagram courtesy of Lonnie Welch, Ohio U.



THE Open GROUP

Global State Awareness within Group



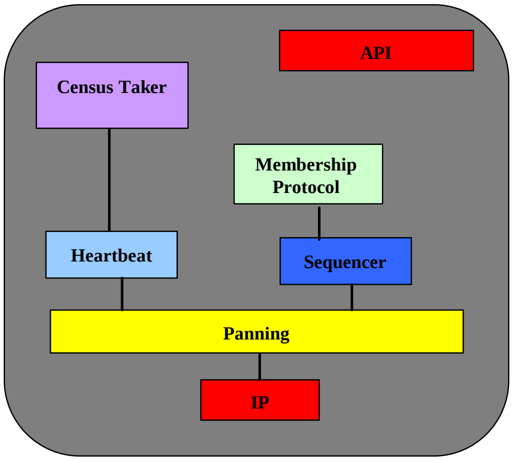
Achieving Real-Time Group Communications

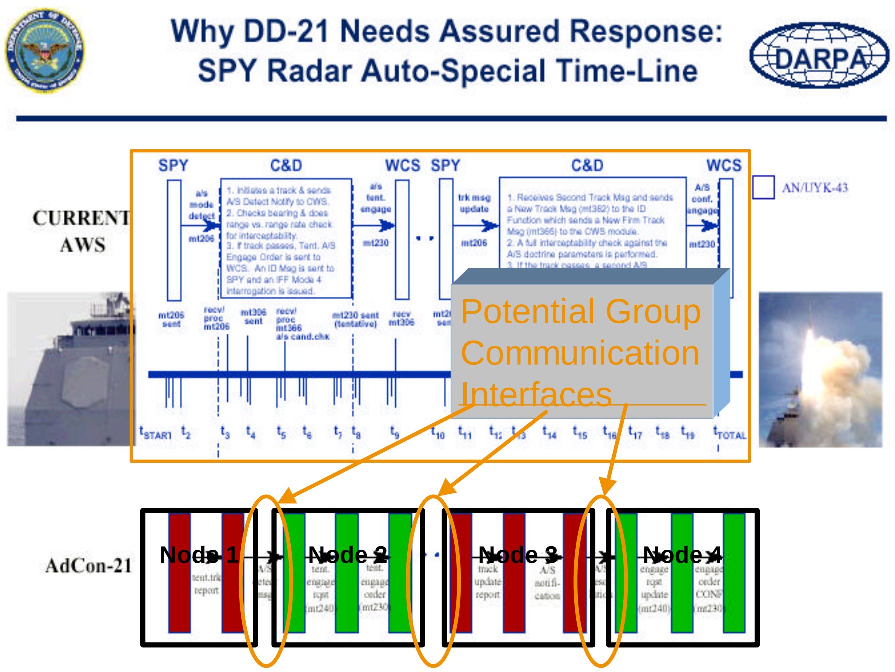
- Message time-outs must be an order of magnitude faster than overall system time constraint
  - End-to-end 1 second deadline might require 0.1 second time-out at each stage
- Group communication time-out periods are often of same order of magnitude as scheduling jitter
- False positives (tardy nodes declared dead), while handled correctly, are expensive
  - Node is forced *down*, then allowed to rejoin
  - Requires reacquisition of application state
- Current "COTS" components (Isis, Ensemble) not designed using real-time techniques and can not achieve these targets

GIPC (Group InterProcess Communication)

- Portable group communication mechanism optimized for real-time predictability
- Special failure semantics and recovery actions based on real-time concepts
- Implemented within The Open Group's CORDS communication protocol framework
  - Numerous "microprotocols" ensure configurability
- Uses CORDS "paths" to manage resources (CPU, buffers, memory, bandwidth, network channels)
  - Resources can be reserved or prioritized
  - Paths managed via system and/or *ad hoc* schedulers
- "Ball sorter" demonstrated reliable node and application-level recovery in less than 400 ms. over both Ethernet and Myrinet

GIPC Group Membership Protocol Stack





Why DD-21 Needs Assured Response: SPY Radar Auto-Special Time-Line

CURRENT AWS

1. Initiates a track & sends A/S Detect Notify to CWS.
2. Checks bearing & does range vs. range rate check for interceptability.
3. If track passes, Tent. A/S Engage Order is sent to WCS. An ID Msg is sent to SPY and an IFF Mode 4 interrogation is issued.

1. Receives Second Track Msg and sends a New Track Msg (mt362) to the ID Function which sends a New Firm Track Msg (mt366) to the CWS module.
2. A full interceptability check against the A/S doctrine parameters is performed.
3. If the track passes, a second A/S

Potential Group Communication Interfaces

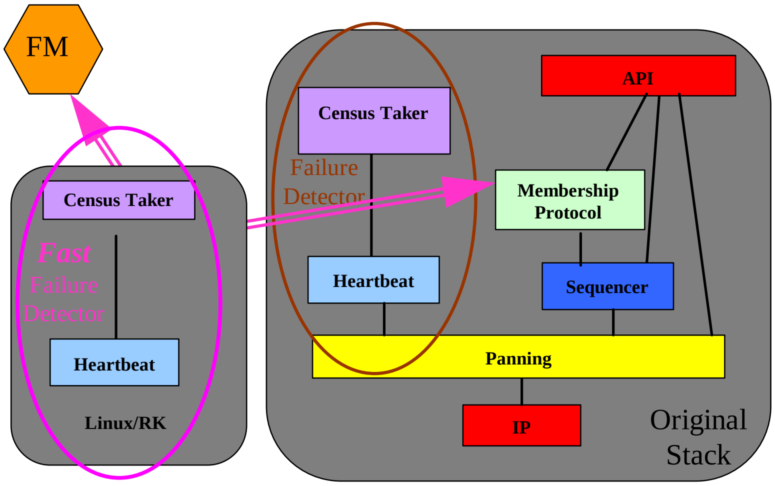
AdCon-21

SPY
C&D
WCS
SPY
C&D
WCS
AN/UYK-43
a/s mode detect
mt206
a/s tent. engage
mt230
trk msg update
mt206
A/S conf. engage
mt230
mt206 sent
recv/ proc mt206
mt306 sent
recv/ proc mt366 a/s cand.chk
mt230 sent (tentative)
recv mt306
t_{START} t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15} t_{16} t_{17} t_{18} t_{19} t_{TOTAL}
Node 1
Node 2
Node 3
Node 4
tent.trk report
tent. engage rqst (mt240)
tent. engage order (mt230)
track update report
A/S notifi-cation
engage rqst update (mt240)
engage order CONF (mt230)

Fast Failure Detector (FFD)

- General Goal of FFD:
  - Provide faster, more reliable detection of host node failure than other components on COTS systems
  - Enhance ability to use non-real-time components in real-time systems
- Method:
  - Isolate detection mechanism into separate component to take advantage of specific environment
  - Inject failure notification into original component
- Example Applications:
  - Provision special FFD process/thread with higher priority and/or reserved resources
  - Install FFD function in interrupt level device driver
  - Interface to OS-specific mechanisms, e.g., *waitpid*(3)

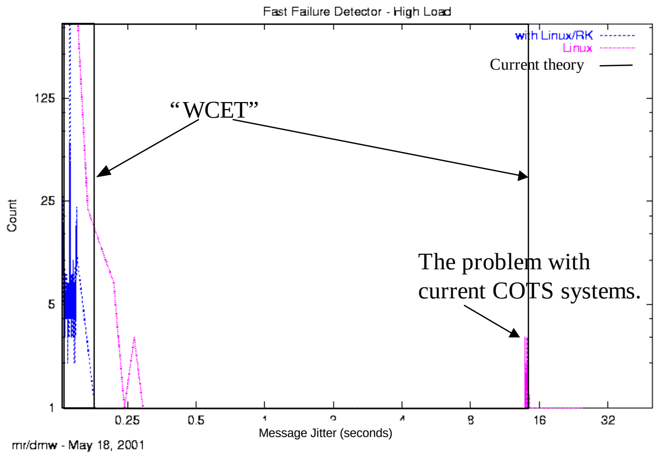
Application of FFD to Group Membership



A Demonstration of FFD Concept

- Installation into HiPer-D test-bed with Ensemble
  - Solaris, Linux/RK
  - Gigabit Ethernet
- Specific Goal of HiPer-D Integration Effort
  - Detect and report host failure within 250 msec
  - This will allow an application to recover from host failure within 1 second worst case, even with substantial state reacquisition
- Results
  - Observed reliable failure detection and notification times of 90-160 msec on both Solaris and Linux/RK
  - FFD resource usage within measurement noise (probably <1% of 100 MHz, 32MB PC and <5% of 10 Mbps Ethernet (not measurements from HiPer-D test-bed))

FFD Message Latency (Jitter) Characterization



“WCET”

The problem with current COTS systems.

mr/dmw - May 18, 2001

Project Strategy

- Drive development from critical technical issues
  - Primarily in RTSJ
  - Evaluate progress based on driving application
- Leverage components via experiments
  - Explores critical technical issues in each component
  - Proves utility of tool kit
  - Utilize TET to drive experiments when appropriate
- Ongoing evaluation of product goals and content based on interaction with potential customers

Existing Components

- CORDS/GIPC
- FFD
- RTSJ
  - TimeSys RTSJ Reference Implementation (RI)
  - TimeSys RTSJ Product (JTime)
- RT CORBA (projected)
  - OIS ORBexpress
  - WUStL/UCI/Vanderbilt TAO
- TET (Test Environment Toolkit)

Anticipated Extensions

- Resource Management
  - DMTF CIM/WBEM (OpenPegasus) — component mgmt.
  - ARM, AIC — application/performance monitor/control
- Enhanced Networking
  - SCTP (Stream Control Transport Protocol)
- ADL-Plus — behavioral testing
- Additional CORBA capabilities

Test/Operational Environment

- RTOS
  - Solaris
  - GPL TimeSys Linux
  - TimeSys Linux product
- Network Infrastructure
  - TCP/IP-based
  - Network characteristics controllable via socket QoS parameters

Framework Characteristics

- Framework: interoperability and compatibility
  - Additional components from other vendors can be incorporated into this framework
  - Interoperable with existing, deployed systems and components
- Independent and incremental insertion of framework and tool kit components
- Not necessary to use Java to use overall framework

Commercialization Plan

- Different than traditional product efforts
- Multiple, concurrent technology insertion paths
  - Productization (3-5 years)
    - Transfer of resulting technology to traditional vendors
    - Availability to multiple suppliers
  - Standardization (2-5 years)
    - Continuing adaptation towards emerging standards
    - Early influence on standards development
    - Interaction with The Open Group Forums
    - Interaction with other standards groups
  - Direct insertion via trial use (ongoing)
    - Continued HiPer-D participation
    - Collaboration with contractors

Benefits Summary

- Reduces deployment cost via increased reuse of COTS and contractor software components in heterogeneous systems
  - Increases reuse of "write once" real-time fault-tolerance software technology
  - Enhances non-real-time software for use in real-time systems
  - Compatible and interoperable with existing base
- Reduces development time by leveraging programmer productivity
  - Use of powerful real-time fault tolerance tools based on COTS languages and architectures

Benefits Summary (cont'd)

- Reduces maintenance costs
  - Reduces "COTS refresh" costs via portability of real-time, fault-tolerant applications
  - Reduces system complexity via coherent architecture based on standardized framework and tool-kit

Open Tool Kit for Mission Critical Systems

For more information:
http://www.opengroup.org/ar/
d.wells@opengroup.org