The SAE Architecture Analysis & Design Language (AADL)

www.aadl.info

Bruce Lewis
Army AMCOM SED
Redstone Arsenal, AL
bruce.a.lewis@us.army.mil
256-876-3224

Joyce L Tokar
Pyrrhus Software
Phoenix, AZ
tokar@pyrrhusoft.com
480-951-1019
Tutorial Objectives

• *Provide* an overview of the SAE AADL Standard.

• *Introduce* architecture-based development concepts.

• *Provide* a summary of AADL capabilities.

• *Demonstrate* the benefits of AADL in real-time systems design.

• *Provide* an overview of the AADL development environment.
Tutorial Outline

- Background & Introduction
- Introduction to AADL
- AADL in Use
- AADL Development Environment
- Benefits of AADL
The SAE AADL Standard

• Sponsored by
  ▶ Society of Automotive Engineers (SAE)
    • Avionics Systems Division (ASD)
      o Embedded Systems (AS2)
        » Avionics Architecture Description Language Subcommittee (AS2C)
        » Bruce Lewis -- Chairman
        » bruce.a.lewis@us.army.mil

• Status
  ▶ Requirements document SAE ARD 5296
    • Balloted and approved in 2000.
  ▶ Standard document SAE AS 5506
    • Balloted and approved 2004.

• Contact
  ▶ http://www.aadl.info email: info@aadl.info
SAE AS-2C ADL Subcommittee

- **Key Players:**
  - Bruce Lewis (AMCOM): Chair, technology user
  - Steve Vestal (Honeywell): MetaH originator, co-author
  - Peter Feiler (SEI): Secretary, main author, editor, technology user
  - Ed Colbert (USC): AADL & UML Mapping
  - Joyce Tokar (Pyrrhus Software): Programming Language Annex, co-editor

- **Members:**
  - Boeing, Rockwell, Honeywell, Lockheed Martin, Raytheon, Smith Industries, Airbus, Axlog, Dassault, EADS, Canadair, High Integrity Systems
  - NAVAir, Open Systems JTF, British MOD, US Army
  - European Space Agency

- **Coordination with:**
  - NATO, ESA, COTRE, OMG-UML
Architecture & Analysis Design Language
AADL

• Specification of
  - Real-time
  - Embedded
  - Fault-tolerant
  - Securely partitioned
  - Modal & dynamically configurable

• Software task and communication architectures

• Bound to
  - Distributed multiple processor hardware architectures

• Fields of application
  - Avionics, Aerospace, Automotive, Autonomous systems, …
Fields of Application

- Automotives
- Avionics
- Robotics

- Fixed sampling and processing rates
- Variable sampling and processing rates
- Stochastic event and processing rates

- Known operational modes and configurations
- Large-scale system integration
MetaH - A Precursor to AADL

1991 DARPA DSSA program begins
1992 Partitioned PFP target (Tartan MAR/i960MC)
1994 Multi-processor target (VME i960MC)
1995 Slack stealing scheduler
1998 Portable Ada 95 and POSIX middleware configurations
1999 Hybrid automata verification of core middleware modules

Numerous evaluation and demonstration projects, e.g.

- Missile G&C reference architecture, demos, others (AMCOM SED)
- Hybrid automata formal verification (AFOSR, Honeywell)
- Missile defense (Boeing)
- Fighter guidance SW fault tolerance (DARPA, CMU, Lockheed-Martin)
- Incremental Upgrade of Legacy Systems (AFRL, Boeing, Honeywell)
- Comanche study (AMCOM, Comanche PO, Boeing, Honeywell)
- Tactical Mobile Robotics (DARPA, Honeywell, Georgia Tech)
- Advanced Intercept Technology CWE (BMDO, MaxTech)
- Adaptive Computer Systems (DARPA, Honeywell)
- Avionics System Performance Management (AFRL, Honeywell)
- Ada Software Integrated Development/Verification (AFRL, Honeywell)
- FMS reference architecture (Honeywell)
- JSF vehicle control (Honeywell)
- IFMU reengineering (Honeywell)
MetaH Case Study at AMCOM

• Missile Application reengineered
  - Missile on-board software and 6DOF environment simulation executing on dual i80960MC, Tartan Ada, VME Boards
  - Built to Generic Missile Reference Architecture
  - Specified in MetaH, 12 to 16 concurrent processes
  - MetaH reduced total re-engineering cost 40% on first project it was used on. Missile prime estimated savings at 66%.

• Missile Application ported to a new execution environment
  - Multiple ports to single and dual processor implementations
  - New processors (Pentium and PowerPC), compilers, O/S
  - First time executable, flew correctly on each target environment
  - Ports took a few weeks rather than 10 months.
AMCOM Effort Saved Using MetaH

Total project savings 50%, re-target savings 90%
Architecture Description Languages

Research ADLs

- **MetaH**
  - Real-time, modal, system family
  - Analysis & generation
  - RMA based scheduling
- **Rapide, Wright, ..**
  - Behavioral validation
- **ADL Interchange**
  - ACME, xADL
  - ADML (MCC/Open Group, TOGAF)

Industrial Strength

- **HOOD/STOOD**
- **SDL**
- **UML 2.0, UML-RT**
Typical Software Development Process

Requirements Analysis
Design
Implementation
Integration

manual, paper intensive, error prone, resistant to change
Model-Based System Engineering

Model-Based & Architecture-Driven

Requirements Analysis

Explicit Architecture Engineering Models Use of AADL

Design, Analysis and Implementation

System Integration

Predictable System Rapid Integration Upgradeability

AADL Tutorial
Model-Based System Engineering

SoS Analysis
- Schedulability
- Performance
- Reliability
- Fault Tolerance
- Dynamic Configurability

System Construction
- Executive generation
- System Integration

Model the Architecture, Abstract & Precise

Performance-Critical Architecture Model
Application System & Execution Platform

Software Systems Engineer

Application Software

Execution Platform

GPS  DB  HTTPS  RTOS

Devices  Memory  Bus  Processor

Domain Specific Components & Subsystems

Automatic Target Recognition
Guidance & Control
Mechanized Sensor & Signal Processing
Ambulatory

AADL Tutorial
Tutorial Outline

• Background & Introduction
• Introduction to AADL
• AADL in Use
• AADL Development Environment
• Benefits of AADL
What is Software Architecture?

• Architecture is the fundamental organization of a system as embodied in
  its components,
  their relationships to each other and the environment,
  the principles governing its design and evolution.
• The software architecture of a program or computing system is
  the structure or structural arrangements of its composite software elements,
  the externally visible properties of those elements,
  the relationships among them.

Architecture is the foundation of good software system engineering
What is an Architecture Description Language (ADL)?

• The architecture of a system defines its high-level structure and exposes its gross organization as a collection of interacting components.
• An Architecture Description Language (ADL) focuses on the high-level structure of the overall application rather than on the implementation details of any specific component.
• ADLs and their accompanying toolsets support architecture-based development, formal modeling, and analysis of architectural specifications.
Common Foundation of ADLs

- **Components** represent the primary (computational) elements and data stores of a system.
- **AADL Components**:
  - **Software Components**:
    - Data, subprogram, thread, thread groups process.
  - **Execution Environment Components**:
    - Memory, bus, device, processor
  - **Composite Components**:
    - System
- **AADL Component Interfaces**:
  - **Shared Data**
  - **Ports**
    - Data ports, event ports, event data ports.
  - **Subprogram Parameters**
Common Foundation of ADLs

• **Connectors** represent interactions among components.

• **Connectors in AADL:**
  - Subprogram Call Sequence
  - Connections
    - Port connections, access connections
  - Flows
Common Foundation of ADLs

- **Systems** represent configurations of components and connectors.
- **Systems in AADL:**
  - Packages
  - Systems
Common Foundation of ADLs

- **Properties** represent semantic information about a system and its components that goes beyond structure.

- **AADL supplies:**
  - Predefined Properties
  - User-defined Property Sets
Common Foundation of ADLs

• Constraints represent claims about an architectural design that should remain true even as it evolves over time.

• Constraints in AADL:
  - Hybrid automata assertions
  - Property value constraints
  - Annexes
Common Foundation of ADLs

• **Styles** represent families of related systems.
• **Styles in AADL:**
  - Multiple implementations
  - Refinement
  - Packages
  - Property sets
  - Annexes
Common Foundation of ADLs

- Components
- Connectors
- Systems
- Properties
- Constraints
- Styles
Focus Of SAE AADL

• Component View
  - Model of system composition & hierarchy.
  - Well-defined component interfaces.

• Concurrency & Interaction View
  - Time ordering of data, messages, and events.
  - Dynamic operational behavior.
  - Explicit interaction paths & protocols.

• Execution view
  - Execution platform as resources.
  - Binding of application software.
  - Specification & analysis of runtime properties
    - timeliness, throughput, reliability, graceful degradation, …
The AADL in a Nutshell

**Performance-Critical**
- Application
  - Thread, Process, System
- Execution Platform
  - Execution engine, Memory, Bus, Device

**Layering & Composition**
- Components
  - Specifications
  - Variant implementations
- Ports
- Connections
- Domain data objects
- Behaviors

**System Architecture**
- **Extensible/Scalable**
  - Multi-processor/multi-process, easily add/change and see effects. User defined domain specific functions.
- **Flexible**
  - System spec used to change implementation. Interface with any standard or application
- **Generic**
  - Modular, scalable, system “block diagram” with semantics
- **Object-Oriented**
  - Clearly defined object, messaging, properties, decomposition
- **Open**
  - Gov usage rights. Industry AADL standard
- **Usable and Available**
  - Approach/formalism is SIMPLE/UNIFORM, PRACTICAL, and EASY TO USE, LEARN, AND INTERFACE WITH OTHER APPROACHES!
- **Hardware Independent**
  - No implementation specified in SW API
- **Hardware Modeling and Bindings**
  - Fully supported by AADL (auxiliary to the SW API)
- **Reliability, Safety, Security Support**
  - User specifies requirements; analyzers available
- **Real-Time**
  - User specifies timing requirements, analyzers available, concurrency handled automatically
- **Verifiable**
  - Strong support for predictable real-time architectures exhibiting high-reliability
- **Usable and Available**
  - Approach/formalism is SIMPLE/UNIFORM, PRACTICAL, and EASY TO USE, LEARN, AND INTERFACE WITH OTHER APPROACHES!
- **Usable and Available**
  - Approach/formalism is SIMPLE/UNIFORM, PRACTICAL, and EASY TO USE, LEARN, AND INTERFACE WITH OTHER APPROACHES!
- **Usable and Available**
  - Approach/formalism is SIMPLE/UNIFORM, PRACTICAL, and EASY TO USE, LEARN, AND INTERFACE WITH OTHER APPROACHES!
- **Usable and Available**
  - Approach/formalism is SIMPLE/UNIFORM, PRACTICAL, and EASY TO USE, LEARN, AND INTERFACE WITH OTHER APPROACHES!
The SAE AADL Standard

• Provides a standard & precise way to describe the architecture of embedded computer systems.
• Provides a standard way to describe components, assemblies of components, and interfaces to components.
• Describes how components are composed together to form complete system architectures.
• Describes the runtime semantics and thread scheduling protocols.
• Describes the mechanisms to exchange control and data between components.
• Describes dynamic run-time configurations.
AADL Concepts Overview

- Hierarchically composed & interconnected components: system.
- Application system components: thread, thread group, process, data, and subprogram.
- Execution platform components: processor, memory, bus, and device.
- Interaction in terms of directional flow via ports, call/return, and data sharing.
- Flow of signals/state (data), control (event & time triggered), and messages (event data) across multiple components.
- Dynamic behavior in terms of statically known alternate task & communication configurations (modes).
- Core AADL and Annex extensions.
Graphical & Textual Notation

```plaintext
system Data_Acquisition
features
  speed_data: in data port metric_speed;
  GPS_data:   in data port position_carthesian;
  user_input_data: in data port user_input;
  s_control_data: out data port state_control;
end Data_Acquisition;
```

- **Data type of port**
- **data port**
AADL Components

• Composite Components
  - System

• Software Components
  - Data
  - Subprogram
  - Thread
  - Thread Group
  - Process

• Execution Platform Components
  - Memory
  - Device
  - Bus
  - Processor
AADL Components

• Component Type -- specifies the interface to the component.
• Component Implementation -- zero or more specifications of the component’s internal representation.
System Components

- Specifies a well-formed interface.
- All external interaction points defined as features.
- Multiple implementations per component type.
- Properties as specified component characteristics.
- Components organized in nested hierarchy.
- Component interaction declarations must follow system hierarchy.
System

• Hierarchical composition.
• Execution platform and application software components.
• Data and bus sharable across system hierarchy.

<table>
<thead>
<tr>
<th>Features: port, subprogram</th>
<th>Subcomponents: process, system, memory, processor, bus, device, and data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides: data access, bus access</td>
<td>Connections: yes</td>
</tr>
<tr>
<td>Requires: bus access, data access</td>
<td>Modes: yes</td>
</tr>
</tbody>
</table>
Device

- Physical component interfacing with environment.
- Interacts with application components via port connections.
- Connects physically to processors via bus.
- May have associated software executes on connected processor.
- Examples
  - sensors and actuators
  - standalone systems such as a GPS

<table>
<thead>
<tr>
<th>Features: port, subprogram</th>
<th>Connections: no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires: bus access</td>
<td>Modes: yes</td>
</tr>
</tbody>
</table>
AADL Interfaces & Connections

- **Ports**
  - Data ports
  - Event Data ports
  - Event ports

- **Connections**
  - Immediate
  - Delayed

![Diagram showing AADL interfaces and connections](image-url)
AADL Interfaces & Connections

• Port Groups
• Connections
AADL Interfaces & Connections

- Subprograms
  - Local
  - Server

- Subprogram calls
  - Local
  - Remote
Shared Data & Bus Access

- Component requires access
- Component provides access
AADL Properties

• Predefined property sets

• User defined property sets

Diagram:
- Thread1: Periodic thread
- Thread2: Aperiodic thread
- Thread1: Sporadic thread
- Background thread
Application Component Hierarchy

Two ways of graphically visualizing component hierarchy
3 Ways to Interact: AADL Component Interaction

- **Unidirectional data & event flow**
- **Synchronous call/return (not shown)**
- **Managed shared data access: only two threads have access**

**Component Interaction Diagram**

- **Flight Mgr**
- **1553**
- **Weapons Mgr**
- **Warnings Annunciations**
- **MFD Pilot**
- **MFD Copilot**

**Data Flow**

- Data flows between components in unidirectional and synchronous ways.
- Managed shared data access ensures only two threads have access.
Application System & Execution Platform

[Diagram showing the connections between different components of an application system, including Flight Mgr, Weapons Mgr, Warnings Annunciations, MFD Pilot, MFD Copilot, Mission Processor, Display Processor, and CoPilot Display. The diagram also shows the 1553 bus and a high-speed network.]
AADL and Scheduling

- AADL provides precise dispatch & communication semantics via hybrid automata.
- AADL task & communication abstraction does not prescribe scheduling protocols
  - Cyclic executive can be supported.
- Specific scheduling protocols may require additional properties.
- Predefined properties support rate-monotonic fixed priority preemptive scheduling.

This scheduling protocol is analyzable, requires small runtime footprint, provides flexible runtime architecture.
AADL Threads

• Threads
  - Periodic -- execute at given time intervals.
  - Aperiodic -- are triggered by an event or remote procedure call.
  - Sporadic -- paced to limit execution rate.
  - Background -- run when there is available processor time.
Task & Interaction Architecture

Thread Dispatch Protocols
- Periodic
- Aperiodic
- Sporadic
- Background

Typed and constrained data streams
Immediate and delayed communication

Directional
Data, event, message ports
Queued and unqueued xfer

Call/Return
Local subprogram
Client/server subprogram

Shared Access
Persistent, shareable data
Access coordination

AADL Tutorial
Thread

- Is a schedulable unit executing on a processor under a scheduling protocol.
- Is dispatched based on time or arrival of events.
- Executes within the protected address space of a process.
- Interacts with other threads through port connections, remote subprogram calls, and shared data access.
- Can be logically organized into thread groups.

### Features:
- port, server subprogram
- Requires: data access
- Provides: data access

### Requirements:
- Subcomponents: Data
- Connections: no
- Modes: yes
Faults and Modes

• AADL provides a fault handling framework with precisely defined actions.
• AADL supports runtime changes to task & communication configurations.
• AADL defines timing semantics for task coordination on mode switching.
• AADL supports specification of mode transition actions.
• System initialization & termination are explicitly modeled.
Hierarchical Modes

- Initial Mode A: Prc1, Prc2; Mode B: Prc1, Prc3;
- Initial Mode A: T1, T2, T3; Mode B: T1, T2;
- Shared data
- Process Prc1
- Process Prc2
- Process Prc3
- Thread T1
- Thread T2
- Thread T3
- Package
- Data 1: Pos
- Data 1: Pos
- Data 1: Pos
- E1
- E1
- E1
- Application Source Internal Mode
- Conditional code
- Mode as Alternative Configuration
A Mode Example

system implementation Main.Example is
   A: system Foo.Bar;
   B: system Oof.Rab;
   C_sub1, C_sub2: system;
   D_sub1, D_sub2: system;
   Mode1: initial mode (A, B, C_sub1, C_sub2);
   Mode2: mode (A, B, D_sub1, D_sub2);
behaviors
   mode Mode1 -[ A.Switch_To_D ]-> Mode2;
   mode Mode2 -[ A.Switch_To_C ]-> Mode1;
end Main.Example;
Behavior Modeling

• Core Features
  • Operational modes
  • Runtime reconfiguration
  • End-to-end flows

• Language Extensions
  • Interaction behavior
    • Port interaction pattern of component
    • Interaction protocol of connection
  • Error models & reliability analysis
System Safety Engineering

Capture the results of
- *Hazard analysis*
- *Component failure modes & effects analysis*

Specify and analyze
- *Fault trees*
- *Markov models*
- *Partition isolation/event independence*

Integration of system safety with architectural design
- Enables cross-checking between models
- Insures safety models and design architecture are consistent
- Reduces specification and verification effort

Supported by Error Model Annex
System & Execution Platforms

Processors, buses, memory, and devices as Virtual Machines

Threads as logical unit of concurrency
Binding Systems to Execution Platforms

Satellite_SW: system

Satellite_SW.Control => Satellite_plant.linuxbox1

Satellite_sw.guidance.observe => Satellite_plant.linuxbox2

Satellite_sys: system

Satellite_plant: platform

Satellite_bus: bus

Satellite: device

Satellite_mem: memory

Satellite_plant.pentium

Observe: thread

Decide: thread

Act: thread

Satellite_plant.linuxbox2

Guidance: process

Control: process

Linuxbox1: platform pentium

Linuxbox2: platform pentium

Linuxbox3: platform PPC
Extensibility

- Core standard plus optional annexes
- Add values for predeclared standard properties.
- Addition of properties.
- Component classifier libraries.
- Extension of component declarations.
- Refinement of subcomponent declarations.
- Modeling of source code data type inheritance.
Extensible Components

Component type (interface)
Component implementations
Subcomponents (hierarchy)
Component instance

Ports
Connections
Modes
Properties
Behavior
Extending an AADL Component
Extending AADL

- Component Types have multiple implementations → families.
- Component extension and refinement.
- Packages.
- Property Sets.
- Annex Subclauses.
- AADL Standard Annexes.
Component Evolution

• Partially complete component type and implementation.
• Multiple implementations for a component type.
• Extension & refinement
  ▶ Component templates to be completed.
  ▶ Variations and extensions in interface (component type).
  ▶ Variations and extensions in implementations.
Large-Scale Development

- Component type and implementation declarations in *packages*
  - Name scope for component types.
  - Grouping into manageable units.
  - Nested package naming.
  - Qualified naming to manage name conflicts.
- Supports independent development of subsystems.
- Supports large-scale system of system development.
AADL Language Extensions

• Core standard plus optional annexes.
• Examples
  - Error Model
  - ARINC 653
  - Behavior
  - Constraint sublanguage
• Annex as document
  - New properties through property sets
  - Annex-specific subclauses expressed in an annex-specific sublanguage
Summary of AADL Capabilities

- AADL abstractions separate application domain concerns from runtime architecture concerns.
- AADL combines predictable task execution with deterministic communication.
- AADL is effective for embedded, real-time, high-dependability, software-intensive application systems.
- AADL supports predictable system analysis and deployment through model-based system engineering.
- AADL component & communication semantics facilitate the dialogue between application and software experts.
- AADL provides an extensible basis for a wide range of embedded systems analyses.
- AADL builds on 13 years of DARPA investment + experiments.
Tutorial Outline

• Background & Introduction
• Introduction to AADL
• AADL in Use
• AADL Development Environment
• Benefits of AADL
What Is Involved In Using The AADL?

- Specify software & hardware system architectures.
- Specify component interfaces and implementation properties.
- Analyze system timing, reliability, partition isolation.
- Tool-supported software and system integration.
- Verify source code compliance & middleware behavior.

Model and analyze early and throughout product life cycle
AADL Analysis & Design Methodology

• Here we use the AADL as analysis & design methodology on an existing system
• AADL employs
  - Components with precisely defined semantics.
  - Explicit interactions.
  - Decomposition.
  - Separation of concerns.
• Pattern-based architecture analysis approach
  - Uses design patterns in analysis.
  - Identifies systemic problems early.
  - Enables the right choices with confidence.
  - Provides analysis-based decisions.
Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering
Avionics Systems

• Embedded avionics system designs are evolving to
  - Integrated systems from federated ones.
  - Predictable preemptive scheduling.
  - Extensible system architectures.

• There are distinct perspectives in the design
  - Control and domain engineers.
  - Application software engineers.
  - System software engineers.

• In the remainder of this tutorial we consider
  - A representative avionics system.
  - Design within the context of the AADL.
  - The distinct perspectives involved.
  - Issues associated with the evolution of avionics systems.
Avionics System Example

• Reflects current design approaches including
  - Time and space partitioning
  - Shared memory & port communication
  - Cyclic executive & preemptive scheduling
  - Deterministic communication
  - Distributed system with legacy hardware
  - Fault tolerance and reconfiguration
  - Efficient execution and footprint

Focus on performance-critical system properties
Sample Avionics System Hardware Configuration

- Pilot Multifunction Display1
- Pilot Multifunction Display2
- CoPilot Multifunction Display1
- CoPilot Multifunction Display2

- Display Processor
- Display Processor
- Display Processor
- Display Processor

- High speed network
- Redundant network and bus

- Mission Processor
- Mission Processor
- Mission Processor

- Auto-Pilot
- GPS
- Nav Radio

Does not have access to the 1553 buses
A Typical Context Diagram
Avionics Software Component Layers

- Display Manager
- Warning Annunciation Manager
- Part-Content Manager
- Flight Manager
- Flight Director
- Situation Awareness
- Weapons Manager
- Comm. Manager

1553 Access

- logical interface to 1553
- hides the fact that some processors do not have direct access to 1553 bus
Typical Software to Hardware Mapping

High speed bus

1553 bus
Observations: Hidden Information

- Multiple instances as separate components.
- Both dual and quad redundancy.
- Grouping of redundant instances.
- Documented in text.
- Difficult to understand and analyze.
Perspectives on Devices

- **Hardware Engineer**
  - Device is part of physical system

- **Application developer**
  - Device functionality is part of the application software

- **Control Engineer**
  - Device represents the plant being controlled
Flight Manager: Principal Functionality

- **Periodic I/O**: 20Hz
  - From other Partitions

- **Navigation Sensor Processing**: 20Hz
  - Integrated Navigation: 10Hz

- **Shared Data Area**:
  - Guidance Processing: 20Hz
  - Flight Plan Processing: 5Hz
  - Aircraft Performance Calculation: 2Hz

- **Processing functions**
  - Nav Radio: 20Hz
  - Auxiliary service

**Subsystems**
- Subsystem = CSCI = Partition
Preemptive Scheduling & Data Flow

- Preemptive scheduling & shared variables
- Data flow patterns
- Threads and port connections in AADL
- Efficient and correct implementations

Modernizing an embedded real-time system
From cyclic executive to preemptive scheduling
From shared variables to port communication
A Cyclic Executive Implementation

Switch clock mod
Hyperperiod
Case 20Hz:
call PIO
call NSP
call GP
Case 2*20Hz: -- 10Hz
call PIO
call NSP
call IN
call GP
Case 3*20Hz:
... 
Case 4*20Hz: -- 5Hz

Simple mapping to a cyclic callout implementation
Preemptive Scheduling and Port Communication

- Towards Rate-Monotonic Scheduling
  - Fixed priority preemptive scheduling based on rate-monotonic analysis (RMA)
  - Better resource utilization, more extensible architecture
- Towards port connections
  - Shared variable communication: efficiency and legacy
  - From shared variables to data port connections
- Example: flight manager
- Scheduling and communication in a partition
Benefit of Preemptive Scheduling

- Thread A: 20Hz 20ms
- Thread B: 10Hz 11ms
- Thread C: 20Hz 20ms
- Not schedulable with cyclic executive

Using RMA guarantees deadlines are met

- Schedulable with
  - preemption
  - correct priority (EDF, RMA)
From other Partitions

• Fixed-priority threads
• Priority assignment by developer

Pr 1
Pr 2
Pr 3
Pr 4
Pr 5
Pr 6
Pr 7
Pr 8
Pr 9

10Hz
5Hz
2Hz
20Hz

Periodic I/O

To other Partitions

Shared data area

A Naïve Thread-based Design

Aircraft Performance Calculation
Flight Plan Processing
Guidance Processing
Navigation Processing
Integrated Navigation
Sensor Processing
Navigation

Decreasing Priority
Design Decisions Taken

- Shared variable communication within partition
  - Achieve efficient resource utilization
  - Accommodate legacy code
- Preemptive fixed-priority thread scheduling
  - Used Schedulability analysis (RMA) to confirm schedulability
  - Benefit of more flexible system and efficient resource usage
- Priority assignment for precedence ordering to achieve desired flow
  - Needed because of shared data area
- Periodic I/O
  - Port data to shared data area (support legacy code)
  - Deterministic communication
  - Output packaging of time consistent data set
Impact of Decisions

- Rate-Monotonic scheduling analysis (RMA)
  - Priority inversion by 10Hz thread
- 10Hz thread (IN) may prevent lower priority 20Hz from running if IN’s deadline > .05 sec
  - Assume IN pre-period deadline of .05 sec or less
- 10Hz thread preemption: shared data inconsistency and non-deterministic sampling
  - Atomic shared data read/write operations assures data consistency
  - pre-period deadline of .05sec assures determinism
- .05 sec pre-period deadline for 10Hz
  - Threads must complete at .05 sec
  - Deadline monotonic scheduling analysis

Effectively cyclic executive

Expressed in AADL

Promoted by AADL
Observations on Scheduling

- Situation: Preemptive scheduling and shared data
- Communication via shared data for efficiency
- Precedence ordering to achieve desired flow
- Priority assignment to achieve precedence ordering
- Causes priority inversion
- Requires pre-period deadline
- Leads to cyclic executive like scheduling behavior at expense of preemptive scheduling overhead
Consistent Communication of State

- Situation: Communication by shared variables
- Issue: preemptive scheduling introduces concurrency
- Read/write of consistent data values
- Deterministic read/write order
Need for Atomic Read/Write

- Thread C updates state in multiple write operations
- Thread A preempts thread C while write is in progress
- Techniques for atomic read/write
  - Hardware supported
  - Locking
  - Temporary non-preemption
Deterministic State Communication

- Write & read (send & receive) performed by application
- Variation in actual write & read time
- Preemption & completion time affect receiver sampling
- Example:
  - Thread A -> thread B
  - Thread A twice rate of thread B
  - Thread C preemption adds to variation
- Desired sampling pattern 2X: n, n+2, n+4 (2,2,2,…)
- Worst-case sampling pattern: n, n+1, n+4 (1,3,…)

Timeline

Thread NSP
Thread IN
Thread NavRadio
Timeline
Thread NSP
State Communication in AADL

• AADL precisely specifies the time of data transfer
• AADL semantics assure consistent and deterministic state communication
• Implementation may require double buffering
• Requires runtime system support
Preemptive Scheduling & Data Flow

- Preemptive scheduling & shared variables
- Data flow patterns
- Threads and port connections in AADL
- Efficient and correct implementations

Modernizing an embedded real-time system
From cyclic executive to preemptive scheduling
From shared variables to port communication
Intended Data Flow

Implemented via shared data
Achieved via precedence ordering
Observations on Data Flow

- Data flow described in text and tables.
- Phase delayed data flow hidden in text.
- Output phase delay embedded in periodic I/O thread.
- Application component code contains communication mechanism decision.
Observations on Data Detailing

• More explicit documentation of information flow is needed in this example

• Periodic I/O as focal point
  - Loss of cross-partition flow information

• It is important to explicitly document the flow of data and control information.

• Towards thread interface specification

• Explicit documentation of all interaction points

• Data vs. Message Communication Choice
  - State vs. state change
  - Control system signal stream -> data ports
Preemptive Scheduling & Data Flow

- Preemptive scheduling & shared variables
- Data flow patterns
- Threads and port connections in AADL
- Efficient and correct implementations

Modernizing an embedded real-time system
From cyclic executive to preemptive scheduling
From shared variables to port communication
A Control System Example

\[ X_{i+1} : = K2S(S_{i+1},X_i) \]

Used for iterative/recurrence functions (e.g. integrators, filters)

Discretization of time continuous function

Delayed communication
Observations: Control Processing

- AADL supports mid-frame communication & single sample delay
- Application component developer specifies communication timing in terms of application domain
- AADL analysis identifies mid-frame communication cycles
- AADL determines schedulability of implementation
- Opens dialogue between application developers and software system engineer
Ports and Connections

- Port (flow of data and control)
  - data (state)
  - event-data (queued)
  - event (queuing possible)

Data type checking on connected ports. Checking of data value & stream characteristics.

: out data boat_speed : in data boat_speed
Immediate Data Connections

- Immediate data connections
  - initiated when source thread completes
  - start of receiving thread execution is delayed until completion of the sending thread
  - data transfer occurs only when the dispatch for both the sending and the receiving threads are logically simultaneous

- Effect of immediate communication
  - all threads see input from same sample period
  - constrain order of execution and communication

- Restrictions: No cycles
Immediate Communication

Vertical leveling does not imply different priorities.

Transmission is initiated when read_data thread completes.

Data available to control thread

(deadline = period)
Delayed Connections-Periodic Threads

- For delayed data connections
  - Data transmission is initiated at the deadline of the connection source thread
  - Data is available at the destination port at the next dispatch of the destination thread

- Value:
  - Controlled communication delays
  - Allow feedback loops
  - Break connection cycle
  - Recipient samples at its dispatch rate and time
Delayed Communication

Transfer initiated at the deadline of the source thread

Data available to thread

(\text{deadline} = \text{period})
Flight Manager in AADL

- **Navigation Sensor Processing**
  - From Partitions
  - Nav signal data
  - Nav sensor data

- **Integrated Navigation**
  - Phase delay of Periodic I/O
  - 20Hz
  - 10Hz
  - Nav data
  - Guidance data

- **Guidance Processing**
  - 20Hz

- **Flight Plan Processing**
  - 5Hz
  - FP data

- **Aircraft Performance Calculation**
  - 2Hz

- **Fuel Flow**

AADL Tutorial
End-To-End Flow Specification

- **Navigation Sensor Processing**
  - From Partitions
  - 20Hz
  - Nav signal data
- **Integrated Navigation**
  - 10Hz
  - Nav sensor data
  - 20Hz
  - Nav data
- **Guidance Processing**
  - 20Hz
  - Nav sensor data
  - Guidance
- **Flight Plan Processing**
  - 5Hz
  - FP data
- **Aircraft Performance Calculation**
  - 2Hz
  - Performance data
  - Fuel Flow

Phase delay of Periodic I/O

AADL Tutorial 104
Observations on Use of AADL

- Explicit interface specification for all threads.
- Named and typed ports with flow constraints supports flow consistency checking.
- Focus on data flow, not task priority assignment:
  - Flow expressed through data ports & connections,
  - No shared data object used.
- Explicit specification of communication timing through delayed connection.
- Specification of thread dispatch characteristics.
Deterministic Communication in AADL

- AADL semantics assure communication determinism
  - In terms of dispatch & deadline.
- Implementation considerations
  - Mutual exclusive port variable access,
  - Double buffering as appropriate,
  - AADL runtime system responsible for dispatch & communication.
Sampling & Immediate Connections

AADL assures
- Deterministic sampling
- Single buffer solution
Sampling & Delayed Connections

Delayed vs. immediate
- Both are deterministic
- They differ in amount of latency

Read every value twice

Delayed connection requires double buffering

Timeline

$T_{i,10Hz, 20Hz}$
$T_{i+1}$
$T_{i+2,20Hz}$
$T_{i+1,10Hz}$

Old data value
Up and Down Sampling

- S-C: overlapping lifespan – xfer at Controller rate
- C-A: overlapping lifespan – xfer at Actuator rate

Timeline

Read-only A in port: 3 buffers
Refresh A in port: 4 buffers
No C in out port: plus 1 buffer
Connection Sequences

- What is the effect of combining immediate & delayed connections?

- $S_{20Hz} \rightarrow C_{10Hz} \rightarrow A_{20Hz}$
  - $S;C;A @ S;A$: $C_{D_{20Hz}}$ 0 pd
  - $S_{20Hz} \rightarrow\rightarrow C_{10Hz} \rightarrow A_{20Hz}$
  - $S|A @ S|(C;A)$: $C_{D_{20Hz}}$ 1 pd
  - $S_{20Hz} \rightarrow C_{10Hz} \rightarrow\rightarrow A_{20Hz}$
  - $(S;C) | A @ S|A$: $C_{D_{10Hz}}$ 2 pd
  - $S_{20Hz} \rightarrow\rightarrow C_{10Hz} \rightarrow\rightarrow A_{20Hz}$
  - $(S|C|A) @ (S|C|A):C_{D_{10Hz}}$ 3 pd

@ separates 20 Hz timeframes

| indicates concurrency for multi processing

Implied C deadline

Amount of phase delay
Application Design Tradeoff

- When do I need immediate data transfer?
- When do I need phase delayed transfer?
- When can I afford phase delayed transfer?

Reduction in resource constraints results in higher resource utilization
Observations: AADL Communication

• Focus on what communication is desired, not how it is implemented.
• Assures deterministic communication when desired
• Shows application rates & desired phase delay explicitly.
• Bridging the time gap:
  - Basis for tradeoff between application and system engineer;
  - Sensitivity analysis on variation in sampling rates & dispatch rates;
  - Connection timing affects latency;
  - Connection timing choice affects resource utilization & controller stability.
Preemptive Scheduling & Data Flow

- Preemptive scheduling & shared variables
- Data flow patterns
- Threads and port connections in AADL
- Efficient and correct implementations

Modernizing an embedded real-time system
- From cyclic executive to preemptive scheduling
- From shared variables to port communication
Efficient Communication

• Should not be application developer responsibility
• Optimization problem for software system engineer
• Built into a runtime generation tool
Port Communication & Buffers

Buffer reduction techniques
- Send/receive with or without copy or no send/receive
- Transfer with or without copy or no transfer
- Processing with or without copy

ARINC653, OSEK & other standards intend to support such optimizations
Efficient Communication Implementation

- No preemption within priority level is assumed
- Dispatch order determines precedence order within priority level
- Consider port & buffer variable lifespan to determine sharing of buffer variables

Similar to register allocation problem for compilers

Dispatch order S, C, A

Timeline
Going All The Way

- Specify common in & out port within component using **in out** port in AADL
- Same data type for both ports
- Thread B updates port data

**Diagram:**
- Sensor
- Controller
- Actuator
- S/C data
- C/A data
- S/C/A data

**Timeline:**
- $T_{i,20Hz}$
- $T_{i+1,20Hz}$

**Dispatch order:** S, C, A
Need For Double Buffering

- Delayed actuator connection: controller time extrapolation
- Dispatch order affects # of buffer variables
Periodic Task Communication Summary

<table>
<thead>
<tr>
<th>Periodic</th>
<th>ASR: IMT</th>
<th>PMT</th>
<th>DSR</th>
<th>CSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same period</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_C; \tau_P$</td>
<td>PD/1B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S/X$<em>{NC}$ R$</em>{NC}$</td>
<td></td>
<td>S/X$<em>{NC}$ R$</em>{NC}$</td>
<td>S/X$<em>{NC}$ R$</em>{NC}$</td>
</tr>
<tr>
<td>$\tau_P; \tau_C$</td>
<td>MF/1B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S/X$<em>{NC}$ R$</em>{NC}$</td>
<td></td>
<td>(S$\lor$X$\lor$R)$_C$</td>
<td>(S$\lor$X$\lor$R)$_C$</td>
</tr>
<tr>
<td>$\tau_P \neq \tau_C$</td>
<td>ND/1B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S/X$<em>{NC}$ R$</em>{NC}$</td>
<td></td>
<td>S/X$<em>{NC}$ R$</em>{NC}$</td>
<td>S/X$<em>{NC}$ R$</em>{NC}$</td>
</tr>
<tr>
<td>$\tau_P \mid \tau_C$</td>
<td>ND/3B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S/X$_C$ R$_C$</td>
<td></td>
<td>S/X$<em>{NC}$ R$</em>{NC}$</td>
<td>S/X$<em>{NC}$ R$</em>{NC}$</td>
</tr>
</tbody>
</table>

Application-level send & receive compounds problem

- **MF**: Mid-Frame
- **PD**: Period Delay
- **ND**: Non-Deterministic
- **DR**: Destructive Receive
- **NA**: Not Applicable

1B: Single buffer
2B: Two buffers
3B: Three buffers
4B: Four buffers

Op$_C$: S, X, R data copy
Op$_{NC}$: S, X, R no data copy
S/X: IMT combined send/xfer
S/X/R: CSR combined S, X, R
(o1$\lor$o2)$_C$: One operation copy
## Harmonic Task Communication Summary

<table>
<thead>
<tr>
<th>Harmonic</th>
<th>ASR</th>
<th>DSR</th>
<th>DMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow - Fast</td>
<td>IMT</td>
<td>PMT&lt;sub&gt;P&lt;/sub&gt;</td>
<td>PMT&lt;sub&gt;C&lt;/sub&gt;</td>
</tr>
<tr>
<td>( \tau_C ; \tau_P ; \tau_C )</td>
<td>( \text{PD}_C ) 1B</td>
<td>( \text{PD}_P ) 2B: X</td>
<td>( \text{PD}_{DC} ) 2B: X</td>
</tr>
<tr>
<td>( \tau_P ; \tau_C ; \tau_C )</td>
<td>( \text{MF} ) 1B</td>
<td>( \text{PD}_P ) 2B: X</td>
<td>( \text{PD}_{DC} ) 2B: X</td>
</tr>
<tr>
<td>( (\tau_P \neq \tau_C) ; \tau_C )</td>
<td>( \text{ND} ) 1B</td>
<td>( \text{PD}_P ) 2B: X</td>
<td>( \text{PD}_{DC} ) 2B: X</td>
</tr>
<tr>
<td>( (\tau_P \neq \tau_C) ; \tau_C )</td>
<td>( \text{ND} ) 3B: S/X R</td>
<td>( \text{PD}_P ) 2B: X</td>
<td>( \text{ND} ) 3B: S X</td>
</tr>
</tbody>
</table>

**PD<sub>C</sub>** : Consumer period delay (T)<br>**PD<sub>P</sub>** : Producer period delay (2T)<br>**PD<sub>D</sub>** : \( D_P \leq T_C \) then PDC else PDP

This can be analyzed!!!

Should be responsibility of AADL analyzer & executive generator
No AADL Generator?

• AADL supports application flow specification & preemptive scheduling.
• Complexity left to AADL tools.

• What if no AADL tool?
  - Plug-in as application component development style.
  - Application code unaffected by implementation decisions.
  - Runtime system implementation patterns used by system implementer.
Some Observations

• Legacy Option
  ▪ Static timeline scheduling.
  ▪ Deterministic execution & communication.
  ▪ Underutilized resources & brittle timing architecture.

• Modernization Option
  ▪ Predictable preemptive fixed-priority scheduling.
  ▪ Better resource utilization & flexible timing architecture.
  ▪ Explicit data flow modeling & communication timing.
  ▪ Buffer management is an optimization problem.
Analysis & Design of an Avionics System

• Preemptive Scheduling and Data Flow
• Scheduling and Real-time Performance
• To Poll or Not to Poll
• Partitions & Communication Timing
• End-to-end Flows
• Many Uses of Modes
• System Safety Engineering
AADL and Scheduling

- AADL does not prescribe a scheduling protocol.
- AADL task and communication abstractions support scheduling protocols:
  - A cyclic executive can be supported.
- Specific scheduling protocols may require additional properties.
- Predefined properties support rate-monotonic fixed priority preemptive scheduling that:
  - is analyzable,
  - requires a small runtime footprint,
  - provides a flexible runtime architecture.
Some Thread Properties

- Dispatch_Protocol => Periodic;
- Period => 100 ms;
- Compute_Deadline => value(Period);
- Compute_Execution_Time => 20 ms;
- Initialize_Deadline => 10 ms;
- Initialize_Execution_Time => 1 ms;
- Compute_Entrypoint => “speed_control”;
- Source_Text => “waypoint.java”;
- Source_Code_Size => 1.2 KB;
- Source_Data_Size => .5 KB;
Threads to Source Text and Executables

main()
{
    Initialize()
    {
        //initialize variables and states
        Speed_error = 0;
        .......
    }
    static float throttle_cmd
    speed_control()
    {
        //Computing throttle command....
        ...
        ...
        // output throttle command
        throttle_cmd = s*t;
    }
}
Execution View

AADL Tutorial
Scheduling Analysis

• Scheduling protocol determines analysis
  - Processor budget for static time line (cyclic executive)
  - Rate-monotonic Analysis (RMA) for preemptively scheduled fixed-priority tasks
  - 100% utilization for Earliest Deadline First (EDF)

• What if analysis of
  - Schedulability under different dispatch & scheduling schemes
  - Miss-estimated worst case execution times (WCET)

Commercial real-time system analysis tools provide such support
WCET & Sensitivity Analysis

Processor timing, application Periodic_IO, mode Periodic_IO, processor Ada95

<table>
<thead>
<tr>
<th>Module</th>
<th>Pri</th>
<th>Period</th>
<th>Budget</th>
<th>Critical</th>
<th>Maximum</th>
<th>% Margin</th>
<th>% Util</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic_IO_system</td>
<td></td>
<td>10000</td>
<td>100</td>
<td>588</td>
<td>8399</td>
<td>98.8</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Fast</td>
<td></td>
<td>10000</td>
<td>1050</td>
<td>6174</td>
<td>9349</td>
<td>88.8</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>&lt;self&gt;</td>
<td></td>
<td>1000</td>
<td>5880</td>
<td>9299</td>
<td>94.0</td>
<td>94.3</td>
<td>5.3</td>
<td>Fast</td>
</tr>
<tr>
<td>&lt;system&gt;</td>
<td></td>
<td>50</td>
<td>294</td>
<td>4425</td>
<td>98.9</td>
<td>99.4</td>
<td>0.3</td>
<td>Slow</td>
</tr>
<tr>
<td>Slow</td>
<td></td>
<td>20000</td>
<td>1051</td>
<td>6178</td>
<td>17649</td>
<td>94.0</td>
<td>5.3</td>
<td></td>
</tr>
<tr>
<td>&lt;self&gt;</td>
<td></td>
<td>1000</td>
<td>5880</td>
<td>17599</td>
<td>94.3</td>
<td>99.4</td>
<td>0.3</td>
<td>Slow</td>
</tr>
<tr>
<td>&lt;system&gt;</td>
<td></td>
<td>51</td>
<td>298</td>
<td>8350</td>
<td>99.4</td>
<td>99.4</td>
<td>0.3</td>
<td>Slow</td>
</tr>
</tbody>
</table>

Total utilization = 16.8%
Breakdown utilization = 98.5%
Critical scaling factor = 5.88
Processor schedule is feasible

Times are reported in microseconds
Nominal compute times were used in the compute paths.

How much all tasks can increase their WCET
Real-time Performance

- If a single processor system is not schedulable
- Might consider
  - Rewrite code to make it faster
  - Consider lower signal processing rate for controller (e.g. 40Hz)
  - Use faster processor
  - Add second processor
- Explore these options using AADL and support tools
  - Reduce worst-case execution time
  - Identify schedulable rate from sensitivity analysis results
  - Execution time properties specific to processor speed
  - Reconfigure to different execution platform
Thread Distribution

From Partitions

Nav signal data

Integrated Navigation

Nav sensor data

Nav data

Fuel Flow

Aircraft Performance Calculation

Nav data

Nav sensor data

20Hz

10Hz

2Hz

Guidance Processing

Nav data

Guidance

Flight Plan Processing

FP data

Performance data

5Hz

Processor 1

Memory

Processor 2

bus

AADL Tutorial
Observations: AADL and Performance

- Enables exploration of design alternatives without rewriting code
- Makes explicit critical design considerations
- Facilitates early decisions on real-time design issues
- Redistribution of components
  - Potentially even threads
  - Completely specified interfaces for threads

More on performance later.
Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering
Event Handling By Polling

• Establishes deterministic system execution patterns
  ▪ Fits easily into a cyclic executive.
  ▪ Helps avoid saturation (e.g. alarm overload).
• Minimizes response times
  ▪ Establishes a well-defined reaction latency.
  ▪ Possible to readily minimize reaction latency.
  ▪ Polling rate driven by minimum state change interval.
• Supports hardware environments that are not interrupt driven.
Avionics System Polling Examples

- 10 Hz keypad polling
  - 10 Hz execution rate reserved
  - Interaction transactions
  - Keypad input -> page content update -> keypad input
  - Multiple partitions reserve resources
  - Realistic event interarrival time
  - Keypad interrupt to get system attention

- 20 Hz Navigation radio channel setting
  - Always-on periodic polling
  - Placed with flight manager partition
  - Occasional pilot activity
  - Reserved resource
Nav Radio In Flight Manager

Where to place this thread in the priority hierarchy?

Nav Radio

Navigation Sensor Processing

Integrated Navigation

Guidance Processing

Flight Plan Processing

Aircraft Performance Calculation

Periodic I/O

From other Partitions

20Hz

10Hz

20Hz

10Hz

20Hz

5Hz

20Hz

20Hz

2Hz

From other Partitions

To other Partitions

Decreasing Priority

AADL Tutorial
Polling and Scheduling Analysis

• Consider a polling system with a 5x over sampling of a 10 Hz stream (scheduling rate of 50 Hz)
  ▪ High rate is to reduce reaction latency
  ▪ Reserve processing time at 50Hz: max. 20% resource utilization
  ▪ Assume processing at 10Hz: impact of variation in data arrival

• Many devices (e.g., switches) are polled
  ▪ Resource reservation for all
  ▪ Assume a subset of switches are can be manipulated at any one time

• Towards an event-driven system
  ▪ process only when new data arrives
  ▪ bounded minimum interarrival time
  ▪ results in aperiodic/sporadic thread sequence
Event-Based Processing

- Recognize bounded event arrival rates
- Manage event processing resource needs
- Leverage operational modes
Bounds on Interaction Rate

• Examine end-to-end flow of interaction
  - Utilize AADL flow specifications
  - Avionics example: Nav Radio
    DM -> PCM/domain -> FM -> 1553 IO -> 1553 device -> 1553 IO -> FM -> PCM -> DM
  - Each inter-partition step contributes phase delay
  - Interaction as single-threaded synchronous call/return
  - Realistic inter-arrival rate much lower

Events with minimum interarrival time
  - Bounded by physical world
  - Bounded by application logic
Manage Event Processing Resources

- **Sporadic Threads**
  - Dispatching of threads with hard deadlines and minimum dispatch separation
  - Spread the processing load over time

- **Sporadic Server**
  - Integrated into scheduler
  - Manage response time through queuing discipline
  - Application level implementations exist

- **Event sampling**
  - Periodic sampling of event queues
  - Bundled event processing
  - Example: cascading alarm processing

Approach used by:
- Process control systems
- Time-Triggered Architecture (TTA)

All three techniques are supported by AADL
Slack Scheduling Approach

- Guarantees deadline of scheduled threads.
- Static slack
  - Unscheduled time
- Dynamic slack
  - Unused scheduled time

Slack(t) is maximum time that scheduled tasks can be delayed at time t without missing their deadlines.
Slack Scheduling Performance

Partitioned aperiodic, incremental, and dynamic threads
- DEOS (Primus Epic RTOS, replaced deferred server)

COTS FTP/TCP/IP stack hosted in DEOS
- > 3X improvement in throughput
- 7X reduction in reserved processor utilization (70% down to 10%)
Leverage Operational Modes

- Limit on number of sporadic streams.
- Limit due to physical environment:
  - Pilot only has two hands.
- Operational mode limits pilot activity:
  - Landing gear switch disabled during dogfight.
Mode-Based Solutions

• Improvements for polled systems.
• Consider optional services
  - Active only in certain operational modes.
  - Smaller set of polling threads.
• Consider reachable mode combinations
  - Reduction in worst-case execution time requirements.
Mode Sensitive WCET

- Worst-case mode combinations
  - Example: Terrain Following (TF) & Reconnaissance (Recon)
  - Recon Hi & TF Hi operationally or logically infeasible

<table>
<thead>
<tr>
<th>Worst-Case Execution Time</th>
<th>TF Hi</th>
<th>TF Lo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recon Hi</td>
<td>50ms+50ms</td>
<td>50ms+10ms</td>
</tr>
<tr>
<td>Recon Lo</td>
<td>20ms+50ms</td>
<td>20ms+10ms</td>
</tr>
</tbody>
</table>
Observations on Polling

- AADL supports event driven system view.
- Co-existence with periodic processing.
- Realistic event rates for given flows.
- Techniques for managing resource demands.
- Real-time performance analysis sensitive to operational modes.
Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering
The Partition Concept

- Found in ARINC 653
- Runtime protected address space
- A virtual processor scheduled on a static timeline
- Contained threads (ARINC processes) are scheduled within the bounds of a scheduled partition
- Different partitions can use different thread scheduling protocols
- Communication of queued and unqueued data
- Inter vs. intra partition communication
Partitioning in AADL

• Core AADL has
  - Systems as logical units of composition.
  - Threads as schedulable units of execution.
  - Processes as protected address spaces.

• Extend process concept of AADL
  - New partition scheduling protocol.
  - New scheduler properties for process.
  - Clarification of communication timing.
Partition Order Side Effects

Partition communication via send/receive

![Diagram showing partition communication via send/receive]
Partition Order Side Effects: Periodic I/O

Periodic I/O adds phase delay
Periodic I/O is sensitive to partition order
Why Periodic I/O Does Not Work

• Partitions on single processor
  - Periodic I/O has partition dispatch as time reference.
  - Statically known communication timing characteristics.
  - Partition order affects communication timing.

• Partitions distributed on multiple processors
  - Concurrent partition execution.
  - Non-deterministic communication timing.

Partitions execute simultaneously
Network Loopback Approach

- Partition communication within processor via network.
- Latency unaffected by partition placement.

No latency reduction through partition relocation

- Network load unaffected by partition placement

No network load reduction through partition relocation
Partition Communication in AADL

- Communication via ports and connections.
- Common reference time for data transfer across partitions.
- AADL runtime system manages
  - Partition execution.
  - Inter-partition communication with respect to processor time.
  - Assumes synchronous system.

AADL Annex on partitions must address this.

Periodic system bus plays such a role in Time-Triggered Architecture (TTA)
Partitioned System Design in AADL

- Focus on partition order isolation
  - Delayed connections insensitive to partition order.
  - Delayed connections insensitive to partition concurrency for synchronous systems.
  - Delayed connections contribute to latency.

- Focus on latency
  - Immediate connections reduce latency.
  - Immediate connections constrain partition order.
  - Immediate connection cycles
    - Direct cycle: $P_A.T1 \rightarrow P_B.T2 \rightarrow P_A.T3$
    - Pair-wise cyclic: $P_A.T1 \rightarrow P_B.T2$ & $P_B.T4 \rightarrow P_A.T3$

- Focus on flexibility
  - Acceptable variation in phase delay.
Asynchronous Systems

- Periodic sampling by independent clocks.
- Potential clock drift
- Sliding [0-period] sampling delay variance for inter-processor communication.
- Partition placement results in latency variation.

A periodic thread sequence is independent of clock drift.
Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering
Acceptable AADL Connection Patterns

- Connection sequences
  - Pipeline, flow

- Connection tree
  - Branching flow
  - Different endpoint latencies

- Directed acyclic graph (DAG)
  - Flow with merge points
  - Phase delay difference of branches at merge point
  - Effects of phase delay oscillation in non-deterministic case

- Cyclic connections
  - Feedback control, action/observation
  - Phase delay breaks cycle
How Does Periodic IO Stack Up?

- Periodic IO as application thread
  - Phase delay within source partition.
  - Unnecessary merge-point phase delay.

Expressed in AADL
Time Consistent Data Sets

- Aggregated data transfer
  - Use of port group concept
  - Property to indicate aggregation semantics
Flow Specification in AADL

**Flow Specification**

- **flow path** F1: pt1 -> pt2
- **flow path** F2: pt1 -> pt3

**Flow Implementation**

- **flow path** F1: pt1 -> C1 -> P2.F5 -> C3 -> P1.F7 -> C5 -> pt2
End-To-End Flow Semantics

End-To-End Flow Declaration

flow SenseControlActuate: S0.FS1 -> C1 -> S1.F1 -> C2 -> S2.FS1

End-To-End Flow Semantics of SenseControlActuate

T0.FS1 -> <SemCon1> -> S1.P1.T1.FX1 -> <SemCon3> -> S1.P1.T2.FX2 -> <SemCon3> -> S2.P2.T5.FS1
Data Stream Latency Analysis

• Flow specifications in AADL
  ▪ Properties on flows: expected & actual end-to-end latency
  ▪ Properties on ports: expected incoming & end latency

• End-to-end latency contributors
  ▪ Delayed connections result in sampling latency
  ▪ Immediate periodic & aperiodic sequences result in cumulative execution time latency

• Phase delay shift & oscillation
  ▪ Noticeable at flow merge points
  ▪ Variation interpreted as noisy signal to controller
  ▪ Analyzable in AADL

Potential hazard: Latency calculation & jitter accumulation
Reduce End-To-End Latency

Consider engineering alternatives

• Reduce execution time
  - Limited impact in periodic sampling systems.

• Eliminate unnecessary sampling steps
  - Change to immediate connections for critical flows.
  - Results in partition order constraint.

• Merge partitions
  - Eliminates partition order constraint.

• Aperiodic sequencing of periodic streams
  - Eliminates sampling phase delay.
  - Quasi-periodic stream processing by individual threads.
  - Worst-case latency: sum of deadlines.
Insights into Flow Characteristics

• Miss rate of data stream
  - Accommodates incomplete sensor readings.
  - Allows for controlled deadline misses.

• State vs. state delta communication
  - Data reduction technique.
  - Events as state transitions.
  - Requirement for guaranteed delivery.

• Data accuracy
  - Reading accuracy.
  - Computational error accumulation.

• Information flow as state transition
End-To-End Response

Flow failures
Guaranteed delivery
Completion/Exception reporting
Independent observation
Analysis & Design of an Avionics System

• Preemptive Scheduling and Data Flow
• Scheduling and Real-time Performance
• To Poll or Not to Poll
• Partitions & Communication Timing
• End-to-end Flows
• Many Uses of Modes
• System Safety Engineering
System Redundancy

High speed bus

1553 bus

AADL Tutorial 168
Redundancy Specification

- Grouping into subsystems for configuration
- Non-redundant component replication

Redundancy characteristics as properties
Runtime Co-location Constraints

AADL Tutorial
Replication as AADL Pattern

- Encapsulate redundancy replication in system
- Use mode to specify alternative configurations
- Specify processor and memory binding constraints
Master/Slave Configurations

- **Passive Slave**
  - CSS1 Master
  - SS1.1
  - SS1.2
  - CSS1 Slave
  - SS1.1
  - SS1.2

- **Hot Standby**
  - CSS1 Master
  - SS1.1
  - SS1.2
  - CSS1 Slave
  - SS1.1
  - SS1.2

- **Continuous State Exchange**
  - CSS1
  - SS1.1
  - State
  - SS1.2

- **Voted Output**
  - CSS1
  - SS1.1
  - SS1.2
  - SS1.3
Master Slave Synchronization

- External and internal mode control
- Errors reported as events
- Supports reasoning about master/slave logic

```
Master

Slave

WAM

state

20Hz

init

masterfail

masterok

Mode

Init/restart

20Hz

Observer

AADL Tutorial
```
Observations On System Redundancy

- Redundancy as an abstraction
  - Multiple redundant instances
  - Grouping of redundant instances
  - Redundancy protocol selection
  - Deployment constraints

- Redundancy mechanism as pattern
  - An orthogonal architecture view
  - Nominal & anomalous behavior
  - Modeling of redundancy logic

Understandable and analyzable
AADL Fault Handling

• Application language exception handling
  - not represented in AADL.
• Thread-level recovery
  - recovery entrypoint execution to prepare for next dispatch.
• Thread unrecoverable
  - Reported as error event.
Thread States

Uninitialized Thread

Initialize

Initialized Thread

DeactivateComplete:

ActiveInInitMode:

Activate

InactiveInInitMode:

ActiveInNewMode:

Dispatch:

Deactivate:

InactiveInNewMode:

DispatchComplete:

Fault:

Compute:

Recovered

Recover

Terminate:

Finalize:

FinalizeComplete:

Thread Halted

Thread State

Execute composite state
Thread Entrypoints & Source Text

main()
{
    Initialize()
    {
        //initialize variables and states
        Speed_error = 0;
        ........
    }
    static float throttle_cmd;
    speed_control( ) {
        //Computing throttle command....
        ........
        // output throttle command
        throttle_cmd = s*t;
    }
}
Fault Management

- Fault containment
  - Process as a runtime protected address space.

- Fault recovery
  - Within application code & thread local.
  - Error propagation.

- Propagated error management
  - Propagation through event connections.
  - Trigger reconfiguration through mode switching.
  - Monitoring & decision making through health monitor.

Event queue processing by aperiodic & periodic threads
Temporary Load Shedding

• Scheduling priority-based shedding
  - Fixed priority: lowest rate first; urgency property controls who.
  - EDF: spread the shedding across threads.

• Occasional deadline misses
  - Data sample miss already handled by controller.
  - Multiple misses per time window.

• AADL: Flow/connection property
  - Expected and provided degree of stream completeness.

Affected by scheduling protocol
Explicit Load Management

- Detection mechanism modeled in AADL
  - Deadline miss as processor event.

- Service level reduction via AADL mode
  - Mode-specific property values.
  - Thread internal modes
    - Different service levels within application component.
    - Different precision, different algorithm, …
  - Alternative active thread configurations
    - Load balancing via AADL mode.
    - Preconfigured alternate bindings.
Modal Systems

- Operational modes
  - Reflecting system operation.
- Modal subsystems
  - Independent & coordinated mode switching.
- Alternate system configurations
  - Reachability of mode combinations.
- Reduced analysis space
  - Worst-case scheduling analysis.
- Dynamic configuration management
  - Inconsistency identification through analysis.
  - Inconsistency repair through selective reconfiguration.
Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering
System Dependability and Safety

• Enables integration of system dependability and safety analyses with architectural design
  ➣ Capture the results of
    • Hazard analysis.
    • Component failure modes & effects analysis.
  ➣ Specify and analyze
    • Partition isolation/event independence.
    • Fault trees.
    • Markov models.

• Advantages
  ➣ Ensures consistency between dependability and safety models and architectural design.
  ➣ Enables cross-checking among analysis models.
  ➣ Reduces specification and verification effort.
Fault Avoidance With AADL

- Support to identify and prevent design faults and to ensure correct runtime characteristics.
- Language features for fault avoidance that
  - Encourage and enforce safe software engineering practices.
  - Provide explicit space and time partitioning.
  - Clearly define system fault and error behaviors.
  - Provide clear specification of safety and certification properties.
  - Ready incorporates specialized safety/fault tolerance hardware.
Fault-Tolerance and Safety Features

• Processes may be time and space partitioned
• Safety/design assurance level can be specified for any component
• Hazardous run-time capabilities enabled on a per-process basis
• Executive consensus protocol is plug-replaceable
• Message data errors detected and reported
• Well-defined process error handling semantics
An Error Model Extension

- Supports reliability, availability, maintainability, safety, and related specification and analysis activities.
- Useful in system safety certification and qualification.
- Optional set of declarations and associated semantics.
- Facilitates a variety of analyses
  - Top-down hazard analysis.
  - A bottom-up failure modes and effects analysis.
  - Fault tree and stochastic process analyses.
  - Safety, reliability, availability, maintainability.
  - Integrated analyses.
- Provides a representation of
  - System hazards.
  - Component failure modes.
Error Annex Foundations

- Based on the concepts and terminology defined by IFIP WG10.4.
- Compatible with the core AADL standard.
- Coupled with AADL core capabilities provides
  - Fault avoidance,
  - Fault tolerance,
  - Integrated dependability analyses.
Error Model Description

Error model descriptions may declare:
- Fault events (in subcomponents section)
- Additional error models (in modes section)
- Error mode transitions (in modes section)

Diagram:
- error_free
  - detected_fault
    - fail_stopped
    - propagate
    - fail_stopped
  - undetected_fault
    - propagate
    - babbling
    - propagate
    - babbling
Reliability Modeling

• Model generators output human-readable & structured models.
• Capture data from top-down hazard analysis.
• Capture data from bottom-up failure modes and effects analysis.
• Enables multiple integrated system safety checks and analyses.
Integrated Analysis Capabilities

- Expandable set of analyses of a system specification
  - Basic error model states and transitions.
  - Partition isolation analysis.
- Stochastic concurrent process model
  - Interfacing with a Markov chain as the reachable non-death states of stochastic concurrent process model.
  - Interfacing with Markov chain analysis tools (e.g. SURE from NASA Langley, UltraSAN from University of Illinois, SHARPE from Duke).
Partition Isolation Analysis

- Partitions support software error containment.
- Safety level properties for analysis and certification
  - RTCA/DO-178B defines 5 failure categories
    - Catastrophic
    - Hazardous
    - Major
    - Minor
    - No effect
  - Lower category defect must not affect higher category component.
- Significant reduction of re-certification costs.
Isolation Verification Algorithm

Analysis tool constructs “can affect” relationships:

- Port-to-port data connection (unless specified to the contrary).
- Software component hosted-on hardware component.
- Shared data structures.
- Scheduling attributes impacts on potential timing interference.
- Software component affects on processor or system.
Example Redundant System

Computer Fault Event Rates
Permanent => 0.0001
Transient  => 0.001
Stochastic Concurrent Process Reliability Model

Error models mapped to system components
Error model interactions

masking expression, (e.g. 2 or more A, B, C)
Reachable States: A Markov Chain

Generated model

A 1 B 1
A 2 B 1
A 1 B 2
A 2 B 2

λ
B
A

λ
A

λ
B

λ
A

λ
B
### Example Analysis Results

SURE V7.9.8   NASA Langley Research Center

<table>
<thead>
<tr>
<th>LOWERBOUND</th>
<th>UPPERBOUND</th>
<th>COMMENTS</th>
<th>RUN #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.05514e-06</td>
<td>3.05513e-06</td>
<td>&lt;prune 3.5e-13&gt;</td>
<td>&lt;ExpMat&gt;</td>
</tr>
</tbody>
</table>

103055 PATH(S) TO DEATH STATES 11939 PATH(S) PRUNED
HIGHEST PRUNE LEVEL = 1.29776e-15
Q(T) ACCURACY => 7 DIGITS
480.733 SECS. CPU TIME UTILIZED
Dependability and Safety Summary

- Integration of dependability and system safety with architectural design
  - Ensures consistency of error models and design architecture.
  - Extensible to include additional models.
  - Enables cross-checking between models.
  - Reduces specification and verification effort.
Tutorial Outline

• Background & Introduction
• Introduction to AADL
• AADL in Use
• AADL Development Environment
• Benefits of AADL
Evolutionary Development With Continuous Feedback

- A control systems simulation perspective
- A model-based architecture perspective
- An integrated perspective
A Control Engineer Perspective

Continuous feedback in a controller

Matlab

Component Analysis

Tune parameters

Simulink

Application Code

Validate simulation

Continuous feedback for a control engineer

AADL Tutorial
A Software System Engineer Perspective

Continuous feedback for software system engineer

AADL Tools

AADL Runtime

Timing analysis

Reliability analysis

Refine properties

AADL-based Architecture Model

Application Components

Execution Platform

AADL Tools

AADL Runtime

Timing analysis

Reliability analysis

Refine properties

AADL-based Architecture Model
A Combined Perspective

Continuous interaction between Control engineer & system engineer

AADL Tutorial
Predictable System Integration

• Required, predicted, and actual runtime properties.
• Application components designed against functional and non-functional properties.
• Application code separated from task dispatch & communication code.
• Consistency between task & communication model and implementation through generation.
• Feedback into model parameters: refinement of estimated performance values.
Application Components as Plug-ins

Strong Partitioning
- Timing Protection
- OS Call Restrictions
- Memory Protection

Interoperability/Portability
- Tailored Runtime Executive
- Standard RTOS API
- Application Components
Partitioning of Responsibilities:
The Control Engineer

Application design perspective
- Signal content properties
- Stream completeness characteristics
- Phase delay & timeliness

Application implementation perspective
- Ports accessible as variables
- Port variable values not overwritten during execution
- Control flow via events & messages
- Initialize, activate, deactivate, compute, recover, finalize entrypoints

AADL Tutorial
Partitioning of Responsibilities: The Software System Engineer

Task & Communication Perspective
Task dispatch & deadlines
Timely & deterministic communication
Dynamic reconfiguration

AADL Runtime System
Executive code generated from AADL
Real-time OS API

Runtime System perspective
Rate groups, priorities & dispatch order
Coordinated dispatch & communication
Double buffering where necessary
Shared variables where appropriate
Application Development Environment

AADL-Based Software & Systems Integration Toolset

- target hardware specifications
- re-engineering of legacy software
- traditional development

SimuLink

Application Development Tools

other specialized tools

Complete, Validated Executable System
An XML-Based AADL Tool Strategy

- Textual AADL
- Graphical AADL

AADL Model

- XML

AADL Instance

- XML

- Scheduling Analysis
- Reliability Analysis
- Safety Analysis

- Complete Execution Platform Binding
- AADL Runtime Generator
- Project-Specific In-House
- Filter to Markov Analysis

Commercial Tool like TimeWiz
An Open Source AADL Environment

Eclipse Platform
- Workbench
- JFace
- SWT
- Workspace
- Platform Runtime
- Help
- Team
- Debug

Java Development Tools (JDT)
- Plug-in Development Environment (PDE)

AADL Environment
- AADL Parser
- AADL Textual Editor
- AADL Graphical Editor
- AADL Object API

XML Document Persistence

Standalone Generation Tool
Analysis Tool Via Java
Analysis Tool Via XML
AADL/UML Strategy

2003

Extensibility
Terminology update
Execution platform

ACME Influence

AADL V1

UML class diagrams
UML stereotypes

UML statecharts

AADL V2

Expansion
Discrete event systems
New analyses

2006

MetaH → AADL V1 → AADL V2

UML 1.3 → UML 2.0

UML/AADL

ADL Influence

Real-time Influence

UML Profile

ACME Influence

UML class diagrams
UML stereotypes

UML statecharts

AADL V1 → AADL V2

UML 2.0 → UML/AADL

UML-RT

AADL Tutorial
AADL/UML Relationship

Extensible AADL Annexes
Add to Language, provide UML extensions

To Be submitted to OMG for Adoption

AADL Core

AADL UML Profile

UML 2.0

UML 1.4
Detailed design

Security

AADL Components
Interactions

ADL

UML-RT
Performance
Timeliness

Dependability
Tutorial Outline

- Background & Introduction
- Introduction to AADL
- AADL in Use
- AADL Development Environment
- Benefits of AADL
Final Observations

• Abstract but precise task & communication architecture
  - A bridge between control engineer & software system engineer
• Early insight into para-functional properties
• Runtime system generation leads to model-consistent implementation
• Leads to architecture patterns with predictable characteristics
• An extensible basis for a wide range of embedded systems analyses
Summary of AADL Capabilities

- AADL abstractions separate application architecture concerns from runtime architecture concerns.
- AADL incorporates a run-time architecture perspective through application system and execution platform.
- AADL is effective for specialized views of embedded, real-time, high-dependability, software-intensive application systems.
- AADL supports predictable system integration and deployment through model-based system engineering.
- AADL component semantics facilitate the dialogue between application and software experts.
Value of AADL-Based Development

• Early Prediction and Verification Tools
  - performance
  - reliability
  - system safety

• Component Compliance Verification Tools
  - functional interface
  - resource requirements
  - system safety

• System Integration and Verification Tools
  - workstation testing
  - system performance
  - system safety verification
Benefits

• Model-based system engineering benefits
  
  Predictable runtime characteristics addressed early and throughout life cycle greatly reduces integration and maintenance effort

• Benefits of AADL as SAE standard
  
  AADL as standard provides confidence in language stability, broad adoption, and strong tool support
Contacts

- Bruce Lewis AS2C Chair, bruce.a.lewis@us.army.mil
- Joyce Tokar, AS2C Member, tokar@pyrrhussoft.com
- www.aadl.info
Thank You

www.aadl.info