Tutorial Objectives

- *Provide* an overview of the SAE AADL Standard
- *Introduce* architecture-based development concepts
- *Provide* a summary of AADL notation capabilities
- *Give* an overview of AADL tools
Outline: An Introduction & Overview

Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
- AADL-Based Development Environment
- Case Studies
- AADL Language Concepts
- Open Source AADL Tool Environment
- UML Profile of AADL
- Summary

SAE Architecture Analysis & Design Language (AADL)
- Specification of
  - Real-time
  - Embedded
  - Fault-tolerant
  - Securely partitioned
  - Dynamically configurable
- Software task and communication architectures
- Bound to
  - Distributed multiple processor hardware architectures
- Fields of application
  - Avionics, Automotive, Aerospace, Autonomous systems, …
An SAE Standard

- Based on 15 Years of DARPA funded technologies
- Core language standard has been approved
- Sponsored by
  - SAE International
  - Avionics Systems Division (ASD)
  - Embedded Systems (AS2)
  - AADL Subcommittee (AS-2C)
- Contact
  - Bruce Lewis AS-2C chair, bruce.a.lewis@us.army.mil
  - http://www.aadl.info
  - For Information email to info@aadl.info

SAE AS-2C AADL Subcommittee

- Bruce Lewis (US Army AMRDEC): Chair
- Peter Feiler (SEI): technical lead, author & editor
- Steve Vestal (Honeywell): co-author
- Ed Colbert (USC): UML Profile of AADL
- Joyce Tokar (Pyrrhus Software): Ada & C Annex

Other Voting Members
- Boeing, Rockwell, Honeywell, Lockheed Martin, Raytheon, Smith Industries, General Dynamics, Airbus, Axlog, European Space Agency, TNI, Dassault, EADS, High Integrity Solutions

Coordination with
- NATO Aviation, NATO Plug and Play, French Government COTRE, SAE AS-1 Weapons Plug and Play, OMG UML & SysML
Potential Users

- Airbus
- European Space Agency
- Rockwell Collins
- Lockheed Martin
- Smith Industries
- Raytheon
- Boeing FCS
- Common Missile
- System Plug and Play

New System Engineering Approach incorporates AADL

Modeling of Satellite Systems, Architecture Verification - ASSERT

Modeling of Avionics Computer System

Embedded System Engineering & AADL

Apply AADL for systems integration modeling & analysis

NATO/SAE AS1 Weapon System Integration

AADL Status

- Requirements document SAE ARD 5296
  - Input from aerospace industry
  - Balloted and approved in 2000
- SAE AADL document SAE AS 5506
  - Core language approved by SAE Sept 2004
- In review to be balloted Fall 2004
  - Graphical AADL notation
  - UML profile of AADL for UML1.4 and UML 2.0
  - XMI domain model, XML schema
  - Ada and C Annex
- In development
  - Error Model Annex
  - ARINC 653 Annex
MetaH: Proof of Concepts for 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
1998 Extensibility through MetaH-ACME Mapping
1998 Reliability modeling extension
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)
- 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)

AADL in Context

Research ADLs
- MetaH
  - Real-time, modal, system family
  - Analysis & generation
  - RMA based scheduling
- Rapide, Wright, ..
  - Behavioral validation
- ADL Interchange
  - ACMEx
- UML 2.0, UML-RT
- HOOD/STOOD
- SDL

AADL Extensible
Real-time Dependable

DARPA Funded
Research since 1990

Basis
Extension
Influence
Alignment
Enhancement
UML Profile
Airbus & ESA
Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
  - AADL-Based Development Environment
  - Case Studies
  - AADL Language Concepts
  - Open Source AADL Tool Environment
- Summary
Typical Software Development Process

- **Requirements Analysis**
- **Design**
- **Implementation**
- **Integration**

- **Manual, Paper Intensive, Error Prone, Resistant to Change**
- **High Development & Maintenance Cost**
- **High Risk System Integration**

Little Insight

Embedded Systems Development Concerns

- Incomplete capture of specification and design
- Little insight into non-functional system properties until system integration & test
  - Performance (e.g., Throughput, Quality of Service)
  - Safety - Reliability
  - Time Critical - Security
  - Schedulability - Fault Tolerance
- System Integration - high risk
- Evolvability – very expensive
- Life Cycle Support – very expensive
- Leads to rapidly outdated components
Model-Based System Engineering

Predictive Analysis Early In & Throughout Life Cycle

Architecture Modeling & Analysis

Requirements Analysis

System Integration

Rapid Integration
Predictable Operation
Upgradeability
Reduced Cost

Architecture-Driven Development

AADL-Based System Engineering

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

Software System Engineer

System Integration
- Runtime System Generation
- Application Composition
- System Configuration

Predictive System Engineering
Reduced Development & Operational Cost

SAE AADL

Composable Components

SAE AADL

Architecture Modeling
Abstract, but Precise

Application Software

Execution Platform

Target Recognition

Guidance & Control

Ambulat & Signal Processing

Information Fusion

GPS DB HTTPS Ada Runtime

Devices Memory Bus Processor

© 2004 by Carnegie Mellon University
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, …

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 system integration
- Verify source code compliance & middleware behavior

Model and analyze early and throughout product life cycle
Evolutionary Development

- A control systems simulation perspective
- A model-based architecture perspective
- An integrated perspective

A Control Engineer Perspective

Continuous feedback in a controller

Simulink

Tune parameters

Component Analysis

Matlab

Application Code

Validate simulation

Continuous feedback for a control engineer

with Text_IO
package Main is
begin
type real is digits 14;
type flag is boolean;
x : real := 0.0;
ready : flag := TRUE;

K1

K2s

+-

Matlab

Component Analysis

Application Code

Validate simulation

Continuous feedback for a control engineer

with Text_IO
package Main is
begin
type real is digits 14;
type flag is boolean;
x : real := 0.0;
ready : flag := TRUE;

K1

K2s

+-
A Software System Engineer Perspective

Continuous feedback for software system engineer

AADL Tools

AADL-based Architecture Model

AADL Runtime

Reliability analysis

Timing analysis

Refine properties

Application Components

Execution Platform

Runtime Data

Timing analysis

Reliability analysis

Refine properties

A Combined Perspective

Continuous interaction between Control engineer & system engineer

AADL Tools

AADL-based Architecture Models

AADL Runtime

Reliability analysis

Timing analysis

Refine properties

Matlab

Component Analysis

Application Code

Simulink

Tune parameters
Partitioning of Responsibilities: The Application Engineer

Application design perspective
- Data 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

Partitioning of Responsibilities: The Software System Engineer

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

Runtime System perspective
- Rate groups, priorities & dispatch order
- Coordinated dispatch & communication
- Double buffering where necessary
- Shared variables where appropriate
A Partitioned Portable Architecture

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

Interoperability/Portability
• Tailored Runtime Executive
• Standard RTOS API
• Application Components

Interoperability

Real-Time Operating System

Embedded Hardware Target

Predictable System Integration

• Requirements, predicted, and actual 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
Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering

AADL-Based Development Environment

- Case Studies
- AADL Language Concepts
- Open Source AADL Tool Environment
- Summary

Application Development Environment

AADL-Based Software & Systems Integration Toolset

- SimuLink
- Application Development Tools
- other specialized tools

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

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

- Textual AADL
- Declarative AADL
- Semantic Checking
- Execution Platform Binding
- Scheduling Analysis
- AADL Instance XML
- Reliability Analysis
- Safety Analysis
- Filter to Markov Analysis
- AADL Runtime Generator
- Graphical AADL Editor
- Graphical Layout XML
- Commercial Tool like TimeWiz
- Project-Specific In-House

Two-Tier Tool Strategy

- Open Source AADL Tool Environment (OSATE)
  - Developed by SEI
  - Low entry cost solution (no cost GPL)
  - Multi-platform based on Eclipse
  - Prototyping environment for project-specific analysis
  - Architecture research platform
- Commercial Tool Support
  - UML tool environment extension based on UML profile
  - Extension to existing modeling environment with AADL export/import
  - Analysis tools interfacing via XML or XML to native filter
  - Runtime system generation tools
Open Source AADL Tool Environment

• OSATE is
  – Developed by the Software Engineering Institute
  – Available at under a no cost General Purpose License (GPL)
  – Implemented on top of Eclipse Release 3 (www.eclipse.org)
  – Generated from an AADL meta model
  – A textual & graphical AADL front-end with semantic & XML/XMI support
  – Extensible through architecture analysis & generation plug-ins

• OSATE offers
  – Low cost entrypoint to the use of SAE AADL
  – Platform for in-house prototyping of project specific architecture analysis
  – Platform for architecture research with access to industrial models & industry exposure to research results

Potential Tool Support Areas

• Architecture extraction/import from existing representations
  – UML designs, Simulink models, application code
• Requirements tracing to the AADL design
• Non-functional properties analysis
  – Scheduling, Real-time Simulation, Throughput, Latency, Concurrency, Reliability, Security, Safety, …
• AADL Architecture consistency analysis
  – High/low risk patterns and properties
• AADL Design Risk Assessment
• AADL Architectural Design Optimizer and Quality Metrics
• Auto-document Generation
• Runtime system generation & optimization
Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
- AADL-Based Development Environment

Case Studies

- AADL Language Concepts
- Open Source AADL Tool Environment
- Summary

Two Case Studies

- Pattern-based analysis of systemic issues
  - Modernized avionics system architecture
  - Change in real-time architecture concepts
- Full-scale analysis & integration
  - Port of missile guidance system
  - Tool-supported analysis & generation
AADL-Based Pattern Analysis

- SAE AADL employs
  - Components with precisely defined execution semantics
  - Explicit component interactions
  - 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

An Avionics System Case Study

- Migration from static timeline to preemptive scheduling
  - Identified issues with shared variable communication
  - Migration potential from polling tasks to event-driven tasks
- Flexibility, predictability & efficiency of port-based communication
  - Support for deterministic transfer & optimized buffers
- Effectiveness of connection & flow semantics
  - Bridge to control engineers
  - Insulate from partition scheduling decisions
  - Support end-to-end latency analysis
- Analyzable fault-tolerant redundancy patterns
  - Orthogonal architecture view without model clutter
A Cyclic Executive Implementation

1. Periodic I/O
2. Navigation Sensor Processing
3. Integrated Navigation
4. Guidance Processing
5. Flight Plan Processing
6. Aircraft Performance Calculation

Switch clock mode

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

Cyclic callout implementation

A Naïve Thread-based Design

1. Periodic I/O
2. Navigation Sensor Processing
3. Integrated Navigation
4. Guidance Processing
5. Flight Plan Processing
6. Aircraft Performance Calculation

Potential non-deterministic communication due to preemption

Potential priority inversion due to priority assignment

Tasks must complete within frame => cyclic executive behavior
Flight Manager in AADL

MetaH Case Study at AMCOM

• Reengineered Missile Application
  – Missile on-board software and 6DOF environment simulation originally in Jovial
  – Ported to Ada83, executing on dual i80960MC, Tartan Ada, VME Boards
  – Built to Generic Missile Reference Architecture
  – Specified in MetaH, 12 to 16 concurrent processes
  – Timing analysis early in reengineering effort
  – Runtime executive generated by MetaH toolset
  – MetaH reduced total re-engineering cost 40% on first project it was used on. Missile prime estimated savings at 66%.
MetaH Case Study at AMCOM - 2

- 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
  - Execution platform description and binding specification in MetaH model
  - Port of runtime executive virtual machine to new processor & O/S
  - Ports took a few weeks rather than 10 months

AMCOM Effort Saved Using MetaH

Total project savings 50%, re-target savings 90%

- First integration of reengineered system
- Retargeting to new execution platform
- Reengineering & MetaH model analysis
Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
- AADL-Based Development Environment
- Case Studies

AADL Language Concepts
- Components
- Component interaction & flows
- Faults & modes
- Large-scale development & extensions

- Open Source AADL Tool Environment
- Summary

AADL: The Language

Components with precise semantics
- Thread, thread group, process, system, processor, device, memory, bus, data, subroutine

Completely defined interfaces & interactions
- Data & event flow, synchronous call/return, shared access
- End-to-End flow specifications

Real-time Task Scheduling
- Supports different scheduling protocols incl. GRMA, EDF
- Defines scheduling properties and execution semantics

Modal, configurable systems
- Modes to model transition between statically known states & configurations

Component evolution & large scale development support

AADL language extensibility
Component-Based Architecture

- Specifies a well-formed interface
- All external interaction points defined as features
- Multiple implementations per component type
- Properties to specify component characteristics
- Components organized into system hierarchy
- Component interaction declarations must follow system hierarchy

System Type

```plaintext
system GPS
features
  speed_data: in data port metric_speed
    {sei::miss_rate => 0.001 mps;};
  geo_db: requires data access real_time_geoDB;
  s_control_data: out data port state_control;
flows
  speed_control: flow path
    speed_data -> s_control_data
properties
  sei::redundancy => 2 X;
end GPS;
```
System Implementation

system implementation GPS.secure
subcomponents
decoder: system PGP_DECODER.basic;
encoder: system PGP_ENCODER.basic;
receiver: system GPS_RECEIVER.basic;
connections
c1: data port speed_data -> decoder.in;
c2: data port decoder.out -> receiver.in;
c3: data port receiver.out -> encoder.in;
c4: data port encoder.out -> s_control_data;
flows
speed_control: flow path speed_data -> c1 -> decoder.fs1
               -> c2 -> receiver.fs1 -> c3 -> decoder.fs1
               -> c4 -> s_control_data;
modes none;
properties sei::redundancy_scheme => Primary_Backup;
end GPS;

Application Components

• System: hierarchical organization of components
• Process: protected virtual address space
• Thread group: organization of threads in processes
• Thread: a schedulable unit of concurrent execution
• Data: potentially sharable data
• Subprogram: Callable unit of sequential code
Thread Dispatch Protocols

- **Periodic thread**
  - represents periodic dispatch of threads with typically hard deadlines.

- **Aperiodic thread**
  - represents event-triggered dispatch of threads with typically hard deadlines.

- **Sporadic thread**
  - represents dispatching of threads with minimum dispatch separation and typically hard deadlines.

- **Background thread**
  - represents threads that are dispatched once and execute until completion.

Additional protocols can be introduced.

Thread Execution Semantics

- Nominal & recovery
- Fault handling
- Resource locking
- Mode switching
- Initialization & finalization
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";
Initialize_Entrypoint => "speed_control_init";
Source_Text => "waypoint.java";
Source_Code_Size => 12 KB;
Source_Data_Size => 5 KB;

Data Component

- Data component type represents data type
  - Used for typing ports
  - Optional modeling of operations
- Data component implementation
  - Substructure modeling
- Data component
  - Sharable between threads through data access connections
  - Access properties
  - Concurrency control protocol property
Execution Platform Components

- Processor – Provides thread scheduling and execution services
- Memory – provides storage for data and source code
- Bus – provides physical connectivity between execution platform components
- Device – interface to external environment

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 environment being controlled
Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
- AADL-Based Development Environment
- Case Studies
- AADL Language Concepts
  - Components
  - Component interaction & flows
  - Faults & modes
  - Large-scale development & extensions
- Open Source AADL Tool Environment
- Summary
Ports & Connections

Ports: directional transfer of data & control
Data port: state, sampled data streams
Event port: Queued, thread dispatch & mode switch trigger
Event data port: queued messages
Port group: aggregation of ports into single connection point
Immediate & delayed data connection: deterministic data transfer semantics

Deterministic Communication Issues

Fixed-priority preemptive scheduling
  – Better resource utilization
Efficient data stream communication
  • Shared variable communication within a processor
    – Preemptive scheduling introduces non-deterministic read-write order
  • Single buffer send/receive communication within and across processors
    – Preemption and concurrency of threads result in non-deterministic application level send/receive call ordering

The consequence
  – Non-deterministic variation in latency & phase delay
  – Appearance of noisy data to controllers
Deterministic Communication in AADL

- Data ports represent unqueued communication of data streams
- Immediate & delayed connection timing semantics assure deterministic data stream transfer
- Immediate connections constrain thread execution order
- Delayed connections increase concurrency
- Implementation considerations
  - Mutual exclusive port variable access
  - Double buffering as appropriate
  - AADL runtime system responsible for dispatch & communication

Sampling & Delayed Connections

- Deterministic sampling & phase delay
- Requires double buffering as necessary
Flows in AADL

Flow Specification
F1: flow path pt1 -> pt2
F2: flow path pt1 -> pt3

Flow Implementation
F1: flow path pt1 -> C1 -> P2.F5 -> C3 -> P1.F7 -> C5 -> pt2

End-To-End Flow Declaration
SenseControlActuate: end to end flow Sensor.FS1 -> C1 -> Controller.F1 -> C2 -> Actuator.FS1

Avionics Subsystem Architecture

Observation: No direct connection between flight director and page content manager

Opportunity for connectivity analysis
Response Time Analysis

- Worst-case scenario
  - Period delay per partition communication
    - DM sampling latency (max. = partition period)
    - Six periods of partition communication latency
    - DM execution latency (max. = partition period)
    - 0.4 seconds worst case response time
    - 0.3 seconds best case response time
- Single processor static timeline
  - One direction immediate, opposite direction phase delayed
  - Reduces partition communication latency to three periods
- Multiple processor synchronous system
  - Take into consideration bus/network latency
- Multiple processor asynchronous system
  - Asynchronous sampling with max. sampling latency = period
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

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
• Acknowledgement semantics in terms of flows
Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
- AADL-Based Development Environment
- Case Studies
- AADL Language Concepts
  - Components
  - Component interaction & flows
  - Faults & modes
  - Large-scale development & extensions
- Open Source AADL Tool Environment
- Summary
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

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
Primary/Backup Configurations

Passive Backup
CSS1 Primary
SS1.1
SS1.2
CSS1 Backup
SS1.1
SS1.2

Hot Standby
CSS1 Primary
SS1.1
SS1.2
CSS1 Backup
SS1.1
SS1.2

Continuous State Exchange
CSS1
SS1.1
State
SS1.2

Voted Output
CSS1
SS1.1
SS1.2
SS1.3

Primary Backup Synchronization

• External and internal mode control
• Errors reported as events
• Supports reasoning about Primary/Backup logic
Dual Redundancy Pattern

system PrimaryBackupPattern
features
  insignal: data port;
  outsignal: data port;
end PrimaryBackupPattern;

system implementation PrimaryBackupPattern.impl
subcomponents
  Primary: system sys;
  Backup: system sys;
connections
  inPrimary: data port insignal -> Primary.insignal;
  inBackup: data port insignal -> Backup.insignal;
  outPrimary: data port Primary.outsignal -> outsignal;
  outBackup: data port Backup.outsignal -> outsignal;

modes
  Primarymode: initial mode;
  Backupmode: mode;
  Reinitmode: mode;
  Backupmode -[restart]-> Reinitmode;
  Reinitmode -[Primary.Complete]-> Primarymode;
end PrimaryBackupPattern.impl;

Defines a dual redundant pattern

Refined Dual Redundancy Pattern

system PassivePrimaryBackup extends PrimaryBackupPattern
features
  restart: in event port;
end PassivePrimaryBackup;

system implementation PassivePrimaryBackup.impl extends PrimaryBackupPattern.impl
subcomponents
  Primary: refined to system in modes ( Primarymode );
  Backup: refined to system in modes ( Backupmode );
  Reinit: system reloadsys in modes ( Reinitmode );
connections
  inPrimary: refined to data port in modes ( Primarymode );
  inBackup: refined to data port in modes ( Backupmode );
  outPrimary: refined to data port in modes ( Primarymode );
  outBackup: refined to data port in modes ( Backupmode );

modes
  Reinitmode: mode;
  Backupmode -[restart]-> Reinitmode;
  Reinitmode -[Reinit.Complete]-> Primarymode;
end PassivePrimaryBackup.impl;

Provides externally restart control

Defines who is active when

Defines restart logic
Modal Systems

- Operational modes
  - Alternate task and communication configurations
  - Reflect system operation
- Modal subsystems
  - Internal mode control to model autonomous subsystems
  - External mode control to model coordinated operational modes
- Alternate system configurations
  - Reachability of mode combinations
- Reduced analysis space
  - Less conservative analysis results
- Management of consistent configuration
  - Inconsistency identification through analysis
  - Inconsistency repair through selective reconfiguration

Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
- AADL-Based Development Environment
- Case Studies
- AADL Language Concepts
  - Components
  - Component interaction & flows
  - Faults & modes
  - Large-scale development & extensions
- Open Source AADL Tool Environment
- Summary
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

Example of Dual Redundancy pattern

Large-Scale Development

- Component type and implementation declarations in packages
  - Name scope for component types
  - Public and private package sections
  - 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

- New properties through property sets
- Sublanguage extension
  - Annex subclauses expressed in an annex-specific sublanguage
- Project-specific language extensions
- Language extensions as approved SAE AADL standard annexes
- Examples
  - Error Model
  - ARINC 653
  - Behavior
  - Constraint sublanguage

Example Annex Extension

```plaintext
THREAD t
  FEATURES
    sem1 : DATA ACCESS semaphore;
    sem2 : DATA ACCESS semaphore;
  END t;

THREAD IMPLEMENTATION t.t1
  PROPERTIES
    Period => 13.96ms;
    cotre::Priority => 1;
    cotre::Phase => 0.0ms;
    Dispatch_Protocol => Periodic;

ANNEX cotre.behavior (**
  STATES
    s0, s1, s2, s3, s4, s5, s6, s7, s8 : STATE.
    s0 : INITIAL STATE.
  TRANSITIONS
    s0 -> s1 (PERIODIC_WAIT);
    s1 -> s2 (COMPUTATION(1.9ms, 1.9ms));
    s2 -> sem1.wait ! (-1.0ms) -> s3;
    s3 -> s4 (COMPUTATION(0.1ms, 0.1ms));
    s4 -> sem2.wait ! (-1.0ms) -> s5;
    s5 -> s6 (COMPUTATION(2.5ms, 2.5ms));
    s6 -> sem2.release ! -> s7;
    s7 -> s8 (COMPUTATION(1.5ms, 1.5ms));
    s8 -> sem1.release ! -> s0;
**)
END t.t1;
```

COTRE thread properties

COTRE behavioral annex

Courtesy of
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

Error Modeling Approach

Error model annex clauses declare
- Error states and transitions
- Fault events & occurrence rates
- Error propagation & occurrence rates
- Masking of subcomponent and propagation errors

Architecture model provides
- Dependency information
- Basis for isolation analysis

[Diagram of error modeling approach with states and transitions]

Supported by Error Model Annex
Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
- AADL-Based Development Environment
- Case Studies
- AADL Language Concepts

Open Source AADL Tool Environment

- Developed by the SEI
- No cost CPL license
- OSATE Release 0.3.0 based on Eclipse Release 3
- Parsing & semantic checking of approved AADL
- Text to XML & XML to text
- Syntax-sensitive text editor
- Syntax-Sensitive AADL Object Editor
- AADL property viewer
- AADL to MetaH translator
- Online help
- Model instantiation
- First analysis plug-ins

Processed 21000 line AADL model

Next release Nov 2004
- Graphical editor
- Multi-file support
- Analysis plug-in development
AADL Meta Model

- Defined in Eclipse Modeling Framework (EMF)
  - Collection of packages with multiple graphical views
  - Separate from, but close to UML profile of AADL
- XML as persistent storage
  - XMI specification from Ecore meta model
  - Generated XML schema
- In-core AADL model
  - Generated methods for AADL model manipulation
  - Edit history, deep copy, object editor, graphical editor
  - Methods to support
    - AADL extends hierarchy
    - feature “inheritance”
    - property value “inheritance”

OSATE Plug-in Extensions

- OSATE
  - Textual AADL, Graphical AADL
  - XML/XMI AADL, AADL object model API
  - AADL extension support
- EMF
  - XML/XMI, Metamodel
  - Change notification
  - Multi-file support
- Eclipse
  - Platform independence
  - Extensible help
  - Task & Problem Mgt
  - Team support
  - Plug-in development
- AADL Front-end
  - Text editor
  - Object editor
  - Graphical editor
  - Text<->XML Semantics
- AADL Semantic API
  - Analysis template
  - Generation template
- Architecture Import
  - Simulink/Matlab model
  - Extraction via SVM
- Architecture Export
  - MetaH
- Architecture Analysis
  - Security level
  - Data stream miss rate
  - Latency
- Architecture Consistency
  - Required connectivity
  - Model completeness profiles
  - Connectivity cycles
- Architecture Transform
  - Conceptual architecture -> Runtime architecture
  - Rate group optimization
  - Port group identification
- External Models
  - <See later>
- Model Transformation
  - Timing analysis (RMA)
Tool Plug-in Example

```java
public Object caseConnection(Connection conn) {
    if (conn instanceof DataAccessConnection || conn instanceof BusAccessConnection) return DONE;
    PropertyHolder scxt = (PropertyHolder) conn.getSrcContext();
    PropertyHolder dcxt = (PropertyHolder) conn.getDstContext();
    if (scxt == null || dcxt == null) return DONE;
    if (scxt instanceof PortGroup)
        scxt = conn.getContainingComponentImpl();
    if (dcxt instanceof PortGroup)
        dcxt = conn.getContainingComponentImpl();
    IntegerValue spv = scxt.getSimplePropertyValue("SEI","SecurityLevel");
    IntegerValue dpv = dcxt.getSimplePropertyValue("SEI","SecurityLevel");
    if (spv == null || dpv == null) {
        ErrorHandling.userError(conn,"Security level specification missing");
        return DONE;
    }
    if (spv.getValue() > dpv.getValue())
        ErrorHandling.userError(conn,"Security level violation");
    return DONE;
}
```

Security Level Example

```aadl
property set SEI is
  SecurityLevel : aadlinteger
  applies to (thread, thread group, process, system);
end SEI;

thread peter
  features
    pe: in event port;
    pd: out data port signal;
  properties
    SEI::SecurityLevel => 2;
end peter;

thread pierre
  features
    pd: in data port signal;
    pe: out event port;
  properties
    SEI::SecurityLevel => 1;
end pierre;

process sys
  end sys;

process implementation sys.impl
  subcomponents
    T1: thread peter.impl;
    T2: thread pierre.impl;
  connections
    data port T1.pd -> T2.pd;
    event port T2.pe -> T1.pe;
end sys.impl;
```
OSATE and External Tools

- **Embry-Riddle**
  - Reliability Analysis

- **AADL Extensions**
  - Error model
  - Concurrency behavior

- **Concurrency Analysis**

- **Company**
  - In-house tools

- **OMNET++**
  - Network simulation

- **TimeWiz Commercial Tool**
  - Scheduling analysis
  - Execution trace analysis

- **TimeWeaver (CMU)**
  - Distributed resource allocation
  - Multi-platform runtime system generation

- **MetaH Toolset (Honeywell)**
  - Scheduling analysis
  - Reliability analysis
  - Isolation analysis
  - Runtime system generation

- **AMDML**
  - Object Model Interface
  - Network model

- **Model Export Filters**
  - Timing model

- **Architecture Export**
  - MetaH, TTA

- **Architecture Import**
  - SVM

- **Architecture Extraction**

- **System Verification Manager (CMU)**
  - Simulink/Matlab, Dymola models

- **OSATE Community Development**
  - www.aadl.info website
  - OSATE Plug-in update site
  - Bugzilla error reporting
  - SEI-Hosted CVS Development Server
  - Availability of Source Code (CPL)
  - Plug-in contributions
    - Syntax-sensitive text editor by York U.
    - Graphical layout editor by USC
    - AADL to MetaH translator by SEI
    - Error modeling support by Embry-Riddle
Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
- AADL-Based Development Environment
- Case Studies
- AADL Language Concepts
- Open Source AADL Tool Environment

UML Profile of AADL
- Summary

UML Profile of AADL

- The UML 1.4 and 2.0 profiles of AADL
- Example
  <placeholder for Ed Colbert material>
Outline: An Introduction & Overview

- Overview of SAE AADL Standard
- Model-Based Architecture-Driven System Engineering
- AADL-Based Development Environment
- Case Studies
- AADL Language Concepts
- Open Source AADL Tool Environment
- UML Profile of AADL
- Summary

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 (Tool-Supported)
  - performance
  - reliability
  - system safety
- Component Compliance Verification (Tool-Supported)
  - functional interface
  - resource requirements
  - system safety
- System Integration and Verification (Tool-Supported)
  - workstation testing
  - system performance
  - system safety verification

SAE AADL: An Enabler for Predictable Embedded Systems Engineering

- Industry standard architecture modeling notation & model interchange format facilitates
  - Interchange of architecture models between contractors & subcontractors
  - Integration of architecture models for system of systems analysis
  - Common architecture model for non-functional system property analysis from different perspectives
  - Interoperability of modeling, analysis, and generation tools
- Open Source AADL Tool Environment offers
  - Low cost entrypoint to the use of SAE AADL
  - Platform for in-house prototyping of project specific architecture analysis
  - Platform for architecture research with access to industrial models & industry exposure to research results
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

The End

• For information go to www.aadl.info