AADL For Analysis

Peter Feiler
Software Engineering Institute
phf@sei.cmu.edu
Outline

Checking AADL Semantics
  • Engineering Rules of Thumb
  • Flow Latency Analysis
  • What-if Performance Analysis
Checking of AADL Semantics

• Expected component connectivity
  – Required and optional port connections

• Matching Port Types
  – Accommodates incomplete sensor readings

• Instantiable model
  – Incomplete component declarations
  – Missing components

• Consistency in modal systems
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
Triple Redundant System

- Three processor system
- Triple redundant critical subsystem
- Physical & logical redundancy
- Three additional subsystems
- Two subsystems per processor
- Processor failure
  - Maintain logical triple redundancy
  - Represent alternative configurations by modes
  - Reduce load through mode switch
Static Subsystem Binding

Processor1
  S1Repl
  S4Crit
  S2Backup
  All_Working

Processor2
  S2Repl
  S5
  S1Backup

Processor3
  S3Repl
  S6
  S3Backup
  S4Backup
Static Subsystem Binding

- Statically allocated primary and backup subsystem
- Runtime mode switch between primary and backup system

**system implementation** Reconfiguration.Static_Binding
subcomponents

HW: `system` Hardware.Triple;

S1Repl: `system` Replicated_Software.impl
{Allowed_Processor_Binding => reference HW.P1;} in modes
(All_Working, P2_Failed, P3_Failed);

-- other primary subcomponents

S1Backup: `system` Replicated_Software.impl
{Allowed_Processor_Binding => reference HW.P2;} in modes
(P1_Failed);
Dynamic Subsystem Binding

Processor1

S1Repl

AllWorking

S4Crit

Processor2

S2Repl

S5

Processor3

S3Repl

S6

P3_Failed

S1Repl

S4Crit

S5

S6
Dynamic Subsystem Binding

- Runtime loading & migration of processes
- Modeled by mode-specific bindings

```plaintext
system implementation Reconfiguration.Dynamic_Binding

subcomponents

HW: system Hardware.Triple;
S1Repl: system Replicated_Software.impl {
    Allowed_Processor_Binding => reference HW.P1
        in modes (All_Working, P2_Failed, P3_Failed);
    Allowed_Processor_Binding => reference HW.P2
        in modes (P1_Failed); }

in modes (All_Working, P1_Failed, P2_Failed, P3_Failed);
-- more subcomponents
connections

    data port S2Repl.Output -> S1Repl.Input_1 in modes
        (All_Working, P3_Failed);
    data port S3Repl.Output -> S1Repl.Input_2 in modes
        (All_Working, P2_Failed);
```
Finding Inconsistencies

• Triple redundancy model
  – Part of mode concept paper
  – Has been reviewed by people

• Semantic checking of model by tool
  – Demo with OSATE

• Model constraints
  – Single incoming connection to data ports
  – Ports with required connections
  – Satisfaction for each mode combination

• Two inconsistencies
  – Incorrect connectivity
  – Use of inactive component
Modal Inconsistencies

```AADL
-- the problem below existed as a typo in the paper version of the model
--
data port S1Rep1_backup.Output -> S3Rep1.Input_2 in modes (P1_Failed);
data port S1Rep1.backup.Output -> S3Rep1.Input_1 in modes (P1_Failed);
--

data port S1Rep1.Output -> S2Rep1_backup.Input_1 in modes | P2_Failed);
data port S3Rep1.Output -> S2Rep1_backup.Input_2 in modes | P2_Failed);
data port S2Rep1_backup.Output -> S1Rep1.Input_2 in modes | P2_Failed);
data port S2Rep1_backup.Output -> S3Rep1.Input_1 in modes | P2_Failed);
data port S1Rep1.Output -> S3Rep1_backup.Input_2 in modes | P3_Failed);
data port S2Rep1.Output -> S3Rep1_backup.Input_1 in modes | P3_Failed);
data port S3Rep1_backup.Output -> S1Rep1.Input_2 in modes | P3_Failed);
data port S3Rep1_backup.Output -> S2Rep1.Input_2 in modes | P3_Failed);
-- a connection to represent data transfer during a mode switch
-- original statement has direction of flow incorrect

-- data port S4Crit_backup.State_Xfer -> S4Crit.State_Xfer in modes ((All_Working);
--
-- data port S4Crit_backup.State_Xfer -> S4Crit_Backup.State_Xfer in modes (All_Working);
```

Problems: 3
AADL Property Values:

<table>
<thead>
<tr>
<th>Description</th>
<th>Resource</th>
<th>In Folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>In mode “P1_Failed” data port “Input_2” has more than one connection</td>
<td>triplepledudant.aadl</td>
<td>ExampleModels/aadl</td>
</tr>
<tr>
<td>Connection in mode transitions: source or destination does not support a specified mode</td>
<td>triplepledudant.aadl</td>
<td>ExampleModels/aadl</td>
</tr>
</tbody>
</table>
Outline

- Checking AADL Semantics
- Engineering Rules of Thumb
- Flow Latency Analysis
- What-if Performance Analysis
Engineering Rules of Thumb

• Accepting occasional deadline misses
• Reduced network demand brings down system
• Validating application domain assumptions
An Engineering Tradeoff

A real customer experience

• Control system for diesel-electric motor
  – Preemptively scheduled software
  – Rate monotonic analysis places high resource requirement

• Examination of timing profile
  – Average is 55% of worst case
  – 99.9% of time less than 80% of worst case

• Missed deadlines & fault tolerance
  – Outliers as deadline misses
  – Controller already tolerant to bad elements in signal stream
Making the Tradeoff Safe

• Will they remember
  – New sensor with different failure profile
  – New controller with different stability profile
  – Signal flow over different hardware

• Documenting the tradeoff
  – Stream miss rate property
  – Acceptable incoming miss rates
  – Miss rate contributors

• An analysis tool
  – Cumulative miss rates along critical flows
  – Miss rate contributions based on execution platform binding
Performance Improvement Gone Bad

A real customer experience

• Ground station to accommodate sensor load growth
  – Reduce load in network
  – Two subsystems communicate state change instead of state

• The impact
  – Other subsystems increase network load sporadically
  – Receiving subsystem goes down

• The cause
  – Transmission protocol without guaranteed delivery
  – Overload result in dropping of transmitted state deltas
  – Missing deltas result in inconsistent receiver state
Avoiding Future Mistakes

• Relevant characteristics as properties
  – State vs. state-change communication through ports
  – Bus protocols with or without guaranteed delivery

• Annotating the model
  – Application engineer characterizes data stream
  – Embedded system engineer characterizes hardware & protocols

• The analysis tool
  – Check that connections carrying state changes are bound to buses with guaranteed delivery
Capturing Domain Characteristics

• Safe upgrading of controllers
  – Data range limits and units of measurement for input & output
  – Documenting setpoint constraints as bounded value deltas
  – Consistency checking along connections

• Security levels & information flow
  – Components with security levels
  – Security levels & containment hierarchy
  – Security levels and connections
  – Security levels & execution platform components

• Safety criticality & control of components
  – Component have safety criticality levels
  – Impact on high criticality components
Introducing New Properties

• New properties and property types

property set SEI is
    Priority: aadlinteger 1..256 applies to (thread);
    StreamMissRate: aadlreal units MissRateType applies to (port);
    MissRateType: type units (perSecond, Consecutive);
    SecurityLevel : inherit aadlinteger applies to (thread, thread group, process, system, processor, bus, device, memory);
    SafetyCriticality : aadlinteger applies to (thread, thread group, process, system, processor, bus, device, memory);
end SEI;

-- use of property
InSignal: in data port Sensor { StreamMissRate => 0.01 perSecond;};
Codified Engineering Knowledge

• Incremental enrichment of architecture models
  – Domain specific characteristics through AADL properties
  – Recording of assumed relevant component properties
  – Models of different fidelity

• Low-cost value-added analysis capabilities
  – Single architecture model attributed with project specific properties
  – Reusable collections of consistency checkers
  – Applicable to modal systems
  – Analysis coverage
**Domain Assumption Specification**

```plaintext
system Sensor
features – … Filter system features
annex assumptions {**
  componentAssumptions name=
    DataCollector {
      -- Dependent component DataCollector
      about Sensor {

        -- Assumptions
        assumes validReadings (int validReadingsID,
                                int startLatency, int frequency)
        {
          validReadingsIDs > startLatency/frequency
        }
        {Criticality=CRITICAL_LEVEL_5}
        {ChangeInterval=STATIC};

        -- Guarantees
        guarantees minFrequecy {
          int frequency = 100;
        };}
    }
  }
};
**
```

Ph.D. work by Ajay Tirumala
U. Illinois Urbana-Champaign
XML-Based Assumption Management

- Example of simple search queries.
  - Check status of all critical assumptions.
    - //assumptions/@criticality='CriticalLevel_5'
  - Get all assumptions for a particular component ‘x’
    - //assumptionSet/
      @dependentComponentName='x'.

```
//tinyos_req/@criticality="critical"
Assn x_1
Assn x_2,...
Assn x_n
```

```
//tinyos_req/@changeInterval="system_config"
Assn y_1...
Assn y_n
```
Outline

- Checking AADL Semantics
- Engineering Rules of Thumb
- Flow Latency Analysis
- What-if Performance Analysis
Architecture Consistency - Flows

• Flow latency in partitioned systems
  – Use of flow specification
  – Partition execution semantics
  – Low fidelity, high precision models

• Flow specification across data and flow types
  – Logical application flow

• Spec-based flow analysis
  – Early latency analysis for integrated systems and systems of systems
  – Latency budgeting
  – Specification validation
Description of Cockpit Display Problem

- Pilot multifunction display is comprised of screen, and depending on information displayed, can contain a number of soft keys
- The MFD provides menu-based navigation to status pages from subsystems
- Data for particular display comes from avionics subsystems
- What is the response time for changing to a new page
- Impact of migrating from cyclic executive to pre-emptive scheduling
Realistic Event Processing Rates

• Example: 20 Hz keypad polling
  – 20 Hz execution rate reserved
  – Command transactions
  – Keypad input -> page content update -> keypad input
  – Multiple partitions reserve resources
  – Realistic event interarrival time

• Engineering issues
  – Determination of command execution rate
  – Validation of response time requirements

• Analysis approach
  – Flow analysis for system of (sub)system
Worst Case Use Scenario

- Longest path for page change processing
- Display Manager
- Warning Annunciation Manager
- Page Content Manager
- Flight Manager
- Flight Director
- Situation Awareness
- Weapons Manager
- Comm. Manager
- Cockpit Display
- 1553 Access

Indirect connectivity to Page Content Manager
Flight Director Command Flow

- Cockpit Display
- Display Manager
- Page Content Manager
- Flight Manager
- Flight Director

Request for new page

New page content
Subsystems as Partitions

system Flight_Manager
features
  mil_1553_comm: port group;
  flight_director_comm: port group;
  page_content_mgr_comm: port group;
flows
  pass_through_to_FD: flow path
    page_content_mgr_comm -> flight_director_comm;
  pass_through_from_FD: flow path
    flight_director_comm -> page_content_mgr_comm;
properties
  SEI::partition_latency => 50 ms;
end Flight_Manager;

Subsystem assumed to execute in a partition
Response Time Analysis

• Count the subsystem hops
  – Adds latency corresponding to the partition execution rate

• Account for device latency
  – Captured as latency property on flow specification

• Account for subsystem processing
  – Execution time within partition may exceed partition period
  – Reflected as flow specification latency
  – Rounded up to next partition period

Lower bound on flow latency
A Naïve Thread-based Design

Flight Manager

From other Partitions

Pr 1

20Hz

Periodic I/O

Pr 2

20Hz

Navigation Sensor Processing

Pr 3

10Hz

Integrated Navigation

Pr 4

20Hz

Guidance Processing

SEI::Priority => 4;

Priority assignment by developer results in potential priority inversion

Shared data area

Pr 5

5Hz

Flight Plan Processing

Pr 6

20Hz

To other Partitions

Pr 7

2Hz

Aircraft Performance Calculation

Legacy code uses shared data area

Efficient thread communication

To other Partitions

From other Partitions

Decreasing Priority

Periodic I/O

20Hz

From other Partitions

Decreasing Priority

20Hz

From other Partitions

Decreasing Priority

20Hz

From other Partitions

Decreasing Priority

20Hz

From other Partitions

Decreasing Priority

20Hz
Latency Violation Cause

• Feature interaction of software system mechanisms
  – Periodic I/O
    • Communicate time consistent data
    • Achieved by phase-delayed sampling in application thread
  – Partition
    • Virtual processors scheduled on physical processors
    • Isolation from partition schedule change
    • Achieved by phase-delayed inter-partition communication
Outline

• Checking AADL Semantics
• Engineering Rules of Thumb
• Flow Latency Analysis
What-if Performance Analysis
Architecture Consistency - Performance

• Priority inversion
  – Manually assigned thread priorities

• Scheduling analysis
  – Earliest deadline first: Java-based
  – Rate-monotonic Analysis: Java-based

• Distributed system resource allocation
  – TimeWeaver binpacking
    • Load balance vs. resource minimization
  – Allocation constraints
    • Processor, memory, bus types & instances
    • Co-location
Handling of Processor Speed

Execution time expressed in absolute time

Modeling of processor-specific execution times

- Use of binding-specific property values
  
  \[
  \text{Compute\_Execution\_Time} \Rightarrow 700\text{us}..750\text{us} \\
  \text{in binding PowerPC\_Mhz350;} \\
  \text{Compute\_Execution\_Time} \Rightarrow 600\text{us}..630\text{us} \\
  \text{in binding PowerPC\_Mhz450;}
  \]

- Execution time in terms of reference processor
  
  \[
  \text{SEI::reference\_cycle\_time} \\
  \text{SEI::cycle\_time}
  \]

Estimated or measured time for each processor

Execution time scaled to processor speed differential

Not addressed in standard

Instruction cycle time for a processor

Instruction cycle time for reference processor

Not addressed in standard
How to Meet Timing Requirements

• If a single processor system is not schedulable
• Explore these options using AADL and analysis tools
  – Leverage operational modes
  – Processor speed dependent execution time
  – Rebind to different execution platform
  – Reduce worst-case execution time
  – Identify schedulable rate from sensitivity analysis results

• Might consider
  – Use faster processor
  – Add second processor
  – Rewrite code to make it faster
  – Consider lower signal processing rate for controller

Acceptable controller rates documented as property
Leverage Operational Modes

• Reduced resource demand
  – Active subset of threads affects processor schedule
  – Active subset of connections affects bus schedule

    calibrate: thread GPS.calibration in modes nominal;

• Modeling of user-selectable optional services
  – Activation triggered as mode switch based on device event

    nominal -[autopilot_on]→ cruise;

• Different service levels of components
  – Mode-specific execution times

    Compute_Execution_Time => 45us..50us in modes nominal;
Thread Distribution

**Logical**

- **Navigation Sensor Processing**: 20Hz
- **Integrated Navigation**: 10Hz
- **Guidance Processing**: 20Hz
- **Flight Plan Processing**: 5Hz
- **Aircraft Performance Calculation**: 2Hz
- **Fuel Flow**

**Physical**

- **Processor 1**
- **Memory**
- **Processor 2**
- **DualPortMemory**

- **Nav sensor data**
- **Nav signal data**
- **Nav data**
- **Performance data**
- **FP data**

From Partitions To Partitions
Summary

• AADL semantics establish a level of consistency
• Users can extend the analysis capability
• AADL timing semantics for execution & communication identify latency issues
• Effective what-if analysis to address performance issue