Architecture Modeling and Analysis for Embedded Systems

Overview of AADL and related research activities in RTG

Oleg Sokolsky
September 19, 2008
Overview

• Background
  - Architecture description languages
  - Embedded and real-time systems
• AADL: ADL for embedded systems
• Analysis of embedded systems with AADL
  - Basic analysis
  - Schedulability analysis with ACSR
  - Performance analysis with Real-Time Calculus
Architecture vs. behavior

- How it is constructed vs. is does

- Traditionally, behavior was considered more important
Components, ports, and connections

- Components are boxes with interfaces
- Component interfaces described by ports:
  - Control, data, resource access
- Connections establish control and data flows
- The nature of components may be abstracted
  - Hardware or software, or hybrid
- Example of ADL:
  - Software ADLs, e.g., Wright or ACME
  - Some UML diagrams
Why architectural modeling?

• Helps structure the system into manageable pieces with
  - well-defined functionality
  - clear interfaces
• Avoids integration problems by checking connections between components
  - Helps manage change!
• Supports code generation
Overview

• Background
  - Architecture description languages
  - Embedded and real-time systems
• AADL: ADL for real-time systems
• Analysis of embedded systems with AADL
  - Basic analysis
  - Schedulability analysis with ACSR
  - Performance analysis with Real-Time Calculus
Embedded system architectures

- Tight resource and timing constraints
  - Resource contention: main source of timing violations
- Include both hardware and software
  - Increasingly distributed and heterogeneous
  - Message transmission affect timing as much as processor execution
- Analysis is important to assess system designs early in the development cycle
Architectural vs. analysis modeling

Architectural modeling

Model transformation

Performance and timing analysis

- Close to the application domain, easy to build and understand.
- (Semi-)automatic and traceable
- Approximate and scalable
Real-time systems

- The science of system development under resource and timing constraints
  - System is partitioned into a set of communicating tasks
  - Tasks communicate with sensors, other tasks, and actuators

- Impose precedence constraints
Task execution

- Tasks are invoked periodically or by events
  - Must complete by a deadline
- Tasks are mapped to processors
- Tasks compete for shared resources
  - Resource contention can violate timing constraints

```
<table>
<thead>
<tr>
<th>State</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dormant</td>
<td>invoke, complete</td>
</tr>
<tr>
<td>running</td>
<td>invoke, complete</td>
</tr>
<tr>
<td>preempted</td>
<td>invoke</td>
</tr>
<tr>
<td>blocked</td>
<td>invoke</td>
</tr>
</tbody>
</table>
```

![State transition diagram](image)
Real-time scheduling

- **Processor scheduling**
  - Task execution is preemptable
  - Tasks assigned to the same processor are selected according to priorities
  - Priorities are assigned to satisfy deadlines
    - Static or dynamic
- **Resource scheduling**
  - Mutual exclusion
    - Often non-preemptable
  - Correlated with processor scheduling
Overview

• Background
  - Architecture description languages
  - Embedded and real-time systems
• AADL: ADL for real-time systems
• Analysis of embedded systems with AADL
  - Basic analysis
  - Schedulability analysis with ACSR
  - Performance analysis with Real-Time Calculus
AADL highlights

- **Architecture Analysis and Design Language**
- **Oriented towards modeling embedded and real-time systems**
  - Platform and software components
  - Control, data, and access connections
- **Formal execution semantics in terms of hybrid automata**
- **SAE standard AS-5506**
AADL components

Software components

• Thread
• Thread group
• Data
• Subprogram
• Process

Platform components

• Processor
• Memory
• Bus
• Device

System components

• System
Component interfaces (types)

• Features
  - Points for external connections
    • E.g., data ports

• Flows
  - End-to-end internal connections

• Properties
  - Attributes useful for analysis
Component implementations

- Internal structure of the component
  - Subcomponents are type references
  - Connections conform with flows in the type
  - External features conform with the type
  - Internal features conform with subcomponent types
Features and connections

- **Communication**
  - Ports and port groups
  - Port connections

- **Resource access**
  - Required and provided access
  - Access connections

- **Kinds of port connections:**
  - Event or data event
  - Data
Port connections

- **Semantic port connection**
  - Ultimate source to ultimate destination
    - Thread, processor, or device
- **Type checking of connections**
  - Directions and types must match
Thread components

- Thread represents a sequential flow of control
  - Can have only data as subcomponents
- Threads are executable components
  - Execution goes through a number of states
    - Active or inactive
  - Behaviors are specified by hybrid automata
Thread states

- **Active**
  - Member of current mode

- **Inactive**
  - Not member of current mode
Thread Hybrid Automata
Thread properties

• Dispatch protocol
  - periodic, aperiodic, sporadic, or background
• Period
  - For periodic and sporadic threads
• Execution time range and deadline
  - for all execution states separately (initialize, compute, activate, etc.)
Thread dispatch

- Periodic threads are dispatched periodically
  - Event arrivals are queued
- Non-periodic threads are dispatched by incoming events
- Events can be raised
  - By executing threads
  - Via external connections
  - By the environment (faults etc.)
Other software components

- **Process**
  - Represents virtual address space
  - Provides memory protection
- **Thread group**
  - Organization of threads within a process
  - Can be recursive
- **Subprogram**
  - Represents entry points in executable code
  - Calls can be local or remote
Platform components

- Processor
  - Abstraction of scheduling and execution
  - May contain memory subcomponents
  - Scheduling protocol, context switch times
- Memory
  - Size, memory protocol, access times
- Bus
  - Latency, bandwidth, message size
Component bindings

- Software components are bound to platform components
- Binding mechanism:
  - Properties specify allowed and actual bindings
  - Allows for exploration of design alternatives
Putting it all together: systems

• Hierarchical collection of components
Putting it all together: systems

• A different perspective on the same system
Modes

- **Mode**: Subset of components, connections, etc.
- **Modes** represent alternative configurations
Overview

• Background
  - Architecture description languages
  - Embedded and real-time systems
• AADL: ADL for real-time systems
• Analysis of embedded systems with AADL
  - Basic analysis
  - Schedulability analysis with ACSR
  - Performance analysis with Real-Time Calculus
Static architectural analysis

- Type checking
  - Types of connected ports
  - Allowed bindings
  - Ultimate connection sources and destinations
- Constraint checking
  - Capacity of memory component for data components bound to it?
  - Bus capacity for bound connections
Connections to conventional tools

- Relies on thread semantics
- Processor scheduling

```
T1
  Period => 20ms
  Compute_Deadline => 20ms
  Compute_Execution_Time => [200us,500us]

T2
  Period => 35ms
  Compute_Deadline => 35ms
  Compute_Execution_Time => [1ms,5ms]

T3
  Period => 100ms
  Compute_Deadline => 100ms
  Compute_Execution_Time => [2ms,7ms]
```

- Scheduling_protocol => RM

RMA tool
Overview

• Background
  - Architecture description languages
  - Embedded and real-time systems
• AADL: ADL for real-time systems
• Analysis of embedded systems with AADL
  - Basic analysis
  - *Schedulability analysis with ACSR*
  - Performance analysis with Real-Time Calculus
Dynamic architectural analysis

- Advanced processor scheduling

State space exploration
ACSR basics: events and actions

- Process: a modeling unit
- Steps of a process
  - (Logically) instantaneous events
  - Timed actions
- Events are used for communication
  - Inputs, outputs, and internal: $a? b! \tau$
- Actions require resource access
  - Take one or more units of time
Modeling basics: processes

- **Sequential execution**
  - $P_1$ performs an event and becomes $P_1'$; $P_1'$ performs an action and becomes $P_1$

- **Choice of steps**
  - $P_2$ can input an event or idle
Modeling basics: time progress

- Timing model
  - Time is global
  - All concurrent processes need to pass time together
  - Passing time is an **explicit** choice
    - \( P_1 \) cannot pass time, but \( P_2 \) can
Timeouts and interrupts

- Execution can be abandoned by time progress or external events

```
{ }  {compute}

P_2  P_2'

go?

P_t

t_{max}

P_i

stop?
```
Task skeleton

- A preemptable task $T$ with execution time $[c_{\text{min}}, c_{\text{max}}]$
Task skeleton

• A *non-preemptable* task $T$ with execution time $[c_{\text{min}}, c_{\text{max}}]$
Task activation

- An activator process invokes the task and keeps track of deadlines
  - Periodic activation with period $p$ and deadline = period
  - Aperiodic activation by the completion of task $T'$ with deadline $d$
Parallel composition

- Event synchronization

  \[ P_1 \xrightarrow{go!} P_1' \parallel P_2 \rightarrow P_2' \]

  \[ P_1 || P_2 \xrightarrow{\tau} P_1'|| P_2' \]

- Time synchronization

  \[ P_1 \xrightarrow{\{cpu\}} P_1' \parallel P_2 \xrightarrow{\{bus\}} P_2' \]

  \[ P_1 || P_2 \xrightarrow{\{cpu, bus\}} P_1'|| P_2' \]
Resource conflicts

- Resources are used exclusively

- Alternatives must be provided
Priorities and preemption

- **Access to resources in action steps and to event channels is controlled by priorities:**
  \[ (r_1, p_1), (r_2, p_2) \] (e?, p)
- **Preemption relation on events and actions**
  \[ (cpu, 1), (bus, 2) \preceq (cpu, 2) \]
  \[ (cpu, 1), (bus, 2) \preceq (\tau, 1) \]
Scheduling with priorities

- Priorities in a task reflect scheduling policy
- Static or dynamic priorities
  - A task with EDF priorities:
Enforcing progress: resource closure

- Resource-constrained progress
  - Processes should not wait unnecessarily
- In a closed system, processes have exclusive use of system resources
Schedulability analysis

• Detect two kinds of problems:
  – Resource conflicts
  – Timing violations
• Schedulable systems are deadlock-free
• Analysis method:
  – Deadlock detection
  – Efficient methods for state-space exploration exist
  – Execution trace to a deadlocked state is produced
Translation of AADL into ACSR

• For each thread
  - generate skeleton
    • thread states
    • resources and dependencies (thread connections)
  - populate skeleton
    • timing: period, deadlines (thread properties)
    • events to raise (out event connections)
  - generate activator (dispatch policy property)

• For each processor
  - generate priorities for mapped threads
    • scheduling policy (processor property)
Overview

• Background
  - Architecture description languages
  - Embedded and real-time systems
• AADL: ADL for real-time systems
• Analysis of embedded systems with AADL
  - Basic analysis
  - Schedulability analysis with ACSR
  - Performance analysis with Real-Time Calculus
Performance of stream processing

• Many embedded systems process streams of events/data
  – Media players, control systems
• Each event triggers task execution to process
  – While the task is busy, events are queued
• Performance measures:
  – End-to-end latency
  – Buffer space
• Resource bottlenecks
Modular Performance Analysis

- Developed at ETH Zurich since 2003
- Based on:
  - Max-Plus/Min-Plus Algebra [Quadrat et al., 1992]
  - Network Calculus [Le Boudec & Thiran, 2001]
  - Real-Time Calculus [Chakraborty et al., 2000]
- Supported by a Matlab toolbox

- Next 8 slides courtesy of Ernesto Wandeler, ETHZ
Abstraction for Performance Analysis

Concrete Instance

Abstract Representation

Processor/Network

Task/Message

Service Model

Task / Processing Model

Load Model

Input Stream
Load Model

Event Stream

Arrival Curve $\alpha$ & Delay $d$

deadline = $d$
Load Model

Event Stream

number of events in
in t=[0 .. 2.5] ms

deadline = d

Arrival Curve \( \alpha \) & Delay \( d \)

demand

\begin{align*}
\text{number of events in } \quad & \text{in } t=[0 \ldots 2.5] \text{ ms} \\
\text{deadline } & = d \\
\text{Arrival Curve } & \alpha \text{ & Delay } d
\end{align*}
Load Model

Event Stream

deployment = d

Arrival Curve α & Delay d

maximum / minimum arriving demand in any interval of length 2.5 ms
Load Model

Event Stream

Arrival Curve $\alpha$ & Delay $d$

deadline = $d$

demand

$\alpha^u$

$\alpha^l$

2.5

$\Delta$ [ms]

9/19/2008

Architecture modeling with AADL

56 of 90
Load Model - Examples

- **periodic**
- **periodic w/ jitter**
- **periodic w/ burst**
- **complex**
Service Model

Resource Availability

available service in \( t = [0 \ldots 2.5] \) ms

Service Curves \([\beta^l, \beta^u]\)

maximum/minimum available service in any interval of length 2.5 ms
Service Model - Examples

- **Full resource**
  - $\beta^u$ for upper bound
  - $\beta^l$ for lower bound
  - Graph: $\Delta$ vs. number of cycles

- **Bounded delay**
  - $\beta^u$ for upper bound
  - $\beta^l$ for lower bound
  - Graph: $\Delta$ vs. number of cycles

- **TDMA resource**
  - $\beta^u$ for upper bound
  - $\beta^l$ for lower bound
  - Graph: $\Delta$ vs. number of cycles

- **Periodic resource**
  - $\beta^u$ for upper bound
  - $\beta^l$ for lower bound
  - Graph: $\Delta$ vs. number of cycles
Task / Processing Model

\[ \alpha \rightarrow \beta \rightarrow \beta' \rightarrow \alpha' \rightarrow d \]
Task / Processing Model

Real-Time Calculus

\[ \alpha'(\Delta) = f_\alpha(\alpha, \beta, d) \]
\[ \beta'(\Delta) = f_\beta(\alpha, \beta, d) \]
Real-Time Calculus

\[ \alpha'^u(\Delta) = \min \{ \inf_{\lambda \geq \Delta} \{ \sup_{\mu \leq \Delta} \{ \alpha^l(\mu + \lambda) - \beta^u(\lambda) \} + \beta^l(\Delta - \mu) \}, \beta^l(\Delta) \} \]

\[ \alpha'^u(\Delta) = \min \{ \sup_{\lambda \geq \Delta} \{ \inf_{\mu \leq \lambda + \Delta} \{ \alpha^u(\mu) + \beta^u(\lambda + \Delta - \mu) \} - \beta^l(\lambda) \}, \beta^u(\Delta) \} \]

\[ \beta'^l(\Delta) = \sup_{\Delta \leq \lambda} \{ \beta^l(\lambda) - \alpha^u(\lambda) \} \]

\[ \beta'^u(\Delta) = \max \{ \inf_{\lambda \geq \Delta} \{ \beta^u(\lambda) - \alpha^l(\lambda) \}, 0 \} \]
Scheduling / Arbitration

FP

EDF

GPS

TDMA
Analysis: Delay and Backlog

\[ [\alpha^l, \alpha^u] \rightarrow [\alpha', \alpha''] \rightarrow [\beta', \beta''] \rightarrow \text{RTC} \rightarrow [\beta^l, \beta^u] \]

Load Model

Processing Model

\[ \beta^l \]

\[ \alpha^u \]

\[ \text{delay } d_{\text{max}} \]

\[ \text{backlog } b_{\text{max}} \]
RTC performance analysis

• Construct the graph of abstract components
  - Connected by stream or resource edges
• Associate input arrival and service curves with source nodes
• If the graph is acyclic
  - Compute output curves of each node in the topological order
• Otherwise, break cycles and iterate to fixed point
• Supported by a MATLAB toolbox
Model transformation

• AADL model is transformed into an RTC model
• Load:
  – Input event streams + periodic tasks
• Service:
  – Processors + buses
• Processing components
  – Threads + connections
• Connections
  – Flows provide load connections
  – Mappings provide service connections
Transformation algorithm

- Traverse AADL model, collect processing components and input loads
- Construct graph of processing components based on flows, component mappings, priorities
- Test if the graph has cycles
  - If not, done
    - Analysis requires one iteration
  - Otherwise, cut the “back” edges
    - Analysis requires fixed point computation
    - Check convergence on the cut edges
Transformation illustrated
Transformation illustrated
Case study: wireless architecture

- Model a typical application-level architecture
  - ISA100 application layer as the basis
  - Study applicability of AADL
    - The need for AADL v2 extensions
- Perform analysis of several configurations
  - Find out which modeling approaches work
    - Modeling alarm timeouts as implicit flow did not work at all!
  - Study performance as function of model size
  - Scalability of RTC
ISA100 highlights

- The network contains multiple sensor nodes connected to the wired network through gateways
  - Wired network is the source of various loads
- Three flow types:
  - Periodically published sensor data (TDMA)
  - Parameter traffic (client/server, CSMA)
  - Alarm traffic (client/server, CSMA)
ISA100 highlights

• Parameter cache in the gateway
  - If the requested parameter is in the cache, it is returned to the operator
  - Otherwise, a request to the relevant sensor node is sent
    • The response is placed in the gateway and returned to the operator

• Alarm queue
  - If queue is full, alarm is dropped
    • Node times out and retransmits
  - O/w, alarm is queued and acknowledged
Architecture model – overall
Properties

• **Component mapping**

  subcomponents
  
  software: process GatewaySoftware_impl;
  hardware: processor GatewayHW;

  properties
  
  Actual_Processor_Binding =>
      reference hardware applies to software;

• **Connection mapping**

  connections
  
  edconn0: event data port sensor.publish -> gateway.publish {
      Actual_Connection_Binding =>
          reference mediumWless.mediumTDMA; }

Properties

• Computation

logger: thread AlarmLogger { RTC::Priority => 4; };
thread AlarmLogger
  properties
    Dispatch_Protocol => Aperiodic;
    Compute_Execution_Time => 10 Ms .. 20 Ms;
end AlarmLogger;

• Transmission

bus WirelessTDMA data ParamMsg
  properties
    Propagation_Delay => 500 Us .. 1 Ms;
    Bandwidth => 100 Kbps;
end WirelessTDMA;
end ParamMsg;
Challenges

• Modeling cache effects
  - Flow depends on cache lookup
    • Split flow with a scaling factor
  - Cache is a shared data component
    • Resource contention not modeled
• Modeling alarm queue
  - Alarms may be dropped and retransmitted
    • Hard to model directly
  - Instead, model conditions for no retransmits
More challenges

• Resource partitioning
  - CSMA and TDMA are the same medium
    • Modeled separately, need to be kept coherent when parameters change
  - Virtual buses in AADL v2 – more natural

• Multiplicity of components
  - Many sensor nodes
    • huge model, lots of copy & paste => errors
  - Arrays in AADL v2 – more compact
Additional properties

- Several aspects necessary for analysis are not captured by standard properties of AADL
  - Some are proposed for v2 (need to be amended)
- Property set for missing properties: RTC
  - Input stream properties
    - Input_Timing, Input_Jitter
  - Output stream properties
    - Output_Rate
Analysis model - I
Analysis model - II
Adding multiple nodes

- More processing blocks, more CSMA flows
Analysis results

• Interesting values:
  - End-to-end delays of flows
  - Buffer requirement $b_Q$ for alarm delivery
    - $b_Q < \text{alarm queue length} \Rightarrow \text{alarms are never lost}$
  - Buffer requirements
    - High values indicate that the system does not have enough throughput for the load

• Configurations analyzed:
  - Firmware download – infrequent, long
  - Network noise – frequent, bursty; short
End-to-end delays – alarm flow

- Linear for ample throughput
End-to-end delays – alarm flow

• ... dramatic increase for low throughput
Alarm queue requirements

- Same for both loads; mostly depends on downstream
Scalability – total analysis time

- network noise
- firmware download, low jitter
- firmware download, high jitter

Nodes vs. Analysis Time (seconds)
Scalability – time per iteration

- **Experiments require 4-6 iterations**

![Diagram showing analysis time vs nodes](image)
Scalability results

• Analysis time is much more sensitive to
  - curve shapes
  - ranges of timing constants
    • which, of course, affect curve shapes
  than to the number of blocks to process
• Lots of simple nodes are much more efficient to analyze than even a few complex nodes
• “Divide and conquer” approaches are possible to explore isolated changes
Summary

• **Architectural modeling and analysis**
  - aids in design space exploration
  - records design choices
  - enforces architectural constraints

• **AADL**
  - Targets embedded systems
  - Builds on well-established theory of RTS
  - As a standard, encourages tool development

• **Architectural analysis (+component semantics)**
  - Schedulability (by transformation to ACSR)
  - Performance (by transformation to RTC)