**Contract-Based Integration of Cyber-Physical Analyses**

**Abstract**

The original document contains color images.
Outline

• Analysis integration problem
• Analysis contracts approach
  – Specification
  – Verification
• Experimental results
Model integration in CPS

- Subtle mismatches between technical domains
- Lead to costly fixes or failures
Frequency scaling is applicable only when:
- used after Bin packing
- the system is behaviorally deadline-monotonic
Otherwise, frequency scaling may render the system not schedulable
Hence, model consistency is not sufficient
Analysis integration problem

- Out-of-order execution
- Invalidation of assumptions
Existing solutions

- Assume-guarantee component composition does not handle analytic integration of tools [1][2].
- Architectural views tackle model consistency, not analytic tool consistency [3][4]
- Meta-level AADL languages do not allow domain-specific semantics [5]
- Previous work on contracts: single domain only, unsound and incomplete verification [6]

Running example

Scheduling

Frequency scaling

Thread model checking

Bin packing

Data security

Battery

Battery Scheduling

Discharge

Charge

Thermal runaway

System

Thread

Thread

Thread

CPU

CPU

CPU

Battery

Battery
Outline

• Analysis integration problem

• **Analysis contracts approach**
  - Specification
  - Verification

• Experimental results
Analysis contracts approach

- Formalize analysis domains
- Specify dependencies and assumptions of analyses
- Determine correct ordering of analyses
- Verify assumptions of analyses
Outline

- Analysis integration problem
- Analysis contracts approach
  - Specification
  - Verification
- Experimental results
Running example

Scheduling

- Frequency scaling
- Thread model checking
- Bin packing
- Data security

Battery

- Battery Scheduling
  - Discharge
  - Charge
- Thermal runaway

System

- Thread
- CPU
- Battery
Verification domain

Scheduling domain $\sigma_{\text{sched}}$

- Frequency scaling
- Thread model checking
- Bin packing
- Data security

Battery domain $\sigma_{\text{batt}}$

- Battery Scheduling
  - Discharge
  - Charge
- Thermal runaway

System

- Thread
- CPU
- Battery

13
Verification domain

- Domain \( \sigma \) is a many-sorted signature \((A, S, R, T, \{ \} \sigma)\):
  - \( A \): set of sorts – system elements and standard sorts
    - E.g.: \( \mathcal{B}, \mathbb{Z}, \text{Threads, Batteries, SchedPol} \)
  - \( S: A_i \times \ldots \times A_n \rightarrow A_k \) – static functions that encode design properties
    - E.g.: \( \text{Period, Dline, CPUBind, Voltage} \)
  - \( R: A_i \times \ldots \times A_n \rightarrow A_k \) – runtime functions that encode dynamic properties
    - E.g.: \( \text{CanPrmpt: Threads} \times \text{Threads} \rightarrow \mathcal{B} \)
    - \( TN: \text{Batteries} \times \mathbb{Z} \rightarrow \mathbb{Z} \)
Verification domain

- Domain $\sigma$ is a many-sorted signature $(\mathcal{A}, \mathcal{S}, \mathcal{R}, \mathcal{T}, \{\} \sigma)$:
  - $\mathcal{T}$: execution semantics – set of sequences of $\mathcal{R}$ assignments
    - E.g.: thread scheduler state model for $\sigma_{\text{sched}}$
    - E.g.: battery state model for for $\sigma_{\text{batt}}$
  - $\{\} \sigma$: domain interpretation for $\mathcal{A}$ and $\mathcal{S}$
    - E.g.: $\{\text{SchedPol}\} \sigma = \{\text{RMS, DMS, EDF}\}$
- Architectural model $m$ is an interpretation $\{\} m$ of $\mathcal{A}$, $\mathcal{S}$, and $\mathcal{T}$
  - E.g.: $\{\text{Threads}\} m = \{\text{SensorSample, Ctrl}_1, \text{Ctrl}_2\}$
    - $\{\text{CPUBind}\} m = \{\text{(Ctrl}_1, \text{CPU}_1), (\text{Ctrl}_2, \text{CPU}_2), \ldots\}$
  - $\{\} \sigma \cup \{\} m$ is a full interpretation
Analysis contract

- Given a domain $\sigma$, *analysis contract* $C$ is a tuple $(I, O, A, G)$
  - Inputs $I \subseteq A \cup S$
  - Outputs $O \subseteq A \cup S$
  - Assumptions $A \subseteq F_\sigma$
  - Guarantees $G \subseteq F_\sigma$

- Where:
  - $F_\sigma ::= \{\forall|\exists\} v_1..v_n \cdot \varphi \mid \{\forall|\exists\} v_1..v_n \cdot \varphi : \psi$
  - $\varphi$ is a static logical formula over $A$ and $S$
  - $\psi$ is an LTL formula over $A$, $S$, and $R$

- $F_\sigma$ semantics is given in a standard way
  - $:\Rightarrow$ means $\Rightarrow$ in case of $\forall$, and $\wedge$ in case of $\exists$
Outline

• Analysis integration problem
• Analysis contracts approach
  – Specification
  – **Verification**
• Experimental results
Contract I/O dependencies

Scheduling

- Frequency scaling
  - Power vs. Exec Time

- Thread model checking
  - Deadlock

- Bin packing

- Data security

Battery

- Battery Scheduling
  - Discharge
  - Charge

- Thermal runaway

- CPU

Thread

Thread

Thread

Battery

Battery
Frequency scaling assumption

Behavioral equivalence to deadline-monotonic scheduling:

- $\forall t_1, t_2: \text{Threads} \cdot t_1 \neq t_2 \land \text{CPUBind}(t_1) = \text{CPUBind}(t_2)$:
  
  $G (\text{CanPrmpt}(t_1, t_2) \Rightarrow Dline(t_1) < Dline(t_2))$
Assumption verification

• SMT solver finds solutions for static fragment $\varphi$
  - $\forall t_1, t_2 : \text{Threads} \mid t_1 \neq t_2 \land \text{CPUBind}(t_1) = \text{CPUBind}(t_2)$

• Model checking property $\psi$ in a behavioral Promela model for each SMT solution:
  - $G (\text{CanPrmpt}(t_1, t_2) \Rightarrow Dline(t_1) < Dline(t_2))$
Battery modeling

- Abstraction: circuits
- Selects a scheduler for cell connections
- Oblivious of heat: treats any configuration as acceptable heat-wise

- Restrictions on acceptable thermal configurations
- Guarantee: unacceptable ones don't occur

- Abstraction: geometry
- Simulates heat propagation
- Cannot scale to dynamic scheduling: simulates only fixed cell configurations
Battery scheduling guarantee

- “Bad” thermal configurations not reachable

- $TN(b, i) \in \mathcal{R}$ – number of cells in $b$ with $i$ thermal neighbors
- $K(b, i) \in \mathcal{S}$ – experimental coefficient for $TN(b, i)$
- Guarantee:
  \[ \forall b: \text{Batteries} \cdot G \left( \sum_{i=0..3} K(b, i) \cdot TN(b, i) \right) \geq 0 \]
Battery modeling

Battery Scheduling
Discharge  Charge

Selects a battery scheduler
Guarantee: $\forall b: \text{Batteries} \cdot G \left( \sum_{i=0..3} K(b, i) * T_N(b, i) \right) \geq 0$
Verified with battery Promela/Spin model

$K(b, i)$

Determines $K(b, i)$ via simulation

Thermal runaway
Temp

Battery
Outline

• Analysis integration problem
• Analysis contracts approach
  – Specification
  – Verification
• Experimental results
Framework implementation

OSATE Execution Environment

AADL model instances → AADL types → AADL-DB converter → Analysis tools → Model DB

AADL types

SMT verification engine

SMT problem → Z3

Sched verification engine

Sched Promela model → Spin

Batt verification engine

Batt Promela model

Legend:

Data Object Executable → Dataflow
# Scalability evaluation

- SMT solving typically takes less than 0.1 second
- Spin model checking times:

<table>
<thead>
<tr>
<th>Threads</th>
<th>(R/D)MS time</th>
<th>EDF time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>4</td>
<td>0.01</td>
<td>0.52</td>
</tr>
<tr>
<td>5</td>
<td>0.07</td>
<td>33.4</td>
</tr>
<tr>
<td>6</td>
<td>0.37</td>
<td>2290.0</td>
</tr>
<tr>
<td>7</td>
<td>2.18</td>
<td>Out Mem</td>
</tr>
<tr>
<td>8</td>
<td>12.4</td>
<td>Out Mem</td>
</tr>
<tr>
<td>9</td>
<td>71.2</td>
<td>Out Mem</td>
</tr>
<tr>
<td>10</td>
<td>421</td>
<td>Out Mem</td>
</tr>
<tr>
<td>11</td>
<td>Out Mem</td>
<td>Out Mem</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cells</th>
<th>FGURR time</th>
<th>FGWRR time</th>
<th>GPWRR time</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0.13</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>12</td>
<td>0.61</td>
<td>2.34</td>
<td>3.94</td>
</tr>
<tr>
<td>16</td>
<td>44</td>
<td>31.4</td>
<td>127</td>
</tr>
<tr>
<td>20</td>
<td>1060</td>
<td>619</td>
<td>Out Mem</td>
</tr>
<tr>
<td>25</td>
<td>Out Mem</td>
<td>Out Mem</td>
<td>Out Mem</td>
</tr>
</tbody>
</table>

All times are in seconds
Summary

• Analysis integration is error-prone
  – Incorrect ordering
  – Violation of implicit assumptions
• Our solution:
  – Contract specification language
  – Contract verification algorithm
• Effective, extensible, and scalable
• Future work:
  – Assumptions behind $\tau$ implementation
  – Analysis contracts for multiple views
Contracts

Security Analysis

\[ An_{sec} \cdot C : I = \{ T, ThSecCl \}, O = \{ NotColoc \}, A = \emptyset, G = \{ g \} \]
\[ g : \forall t_1, t_2 \cdot ThSecCl(t_1) \neq ThSecCl(t_2) \Rightarrow t_1 \in NotColoc(t_2) \]

Multiprocessor scheduling: (Binpacking + scheduling)

\[ An_{sched} \cdot C : I = \{ T, C, NotColoc, Per, WCET, Dline \}, O = \{ CPUBind \}, A = \emptyset, G = \{ g \} \]
\[ g : \forall t_1, t_2 \cdot t_1 \in NotColoc(t_2) \Rightarrow CPUBind(t_1) \neq CPUBind(t_2) \]

Frequency Scaling

\[ An_{freqsc} \cdot C : I = \{ T, C, CPUBind, Dline \}, O = \{ CPUFreq \}, G = \emptyset, A = \{ a \} \]
\[ a : \forall t_1, t_2 \cdot CPUBind(t_1) = CPUBind(t_2) \Rightarrow G(CanPrmpt(t_1, t_2) \Rightarrow Dline(t_1) < Dline(t_2)) \]

Model checking periodic program (REK):

\[ An_{rek} \cdot C : I = \{ T, C, Per, Dline, WCET, CPUBind \}, O = \{ ThSafe \}, G = \emptyset, A = \{ a_1, a_2 \} \]
\[ a_1 : \forall t \cdot Per(t) = Dline(t), a_2 : \forall t_1, t_2 \cdot G(CanPrmpt(t_1, t_2) \Rightarrow G(CanPrmpt(t_2, t_1))) \]

Thermal runaway:

\[ An_{therm} \cdot C : I = \{ B, BatRows, BatCols, Voltage \}, O = \{ K \}, A = \emptyset, G = \emptyset \]

Battery Scheduling

\[ An_{bsched} \cdot C : I = \{ B, BatRows, BatCols \}, O = \{ BatConnSchedPol, HasReqLifetime, SeriqlReq, ParalRea \}, A = \emptyset, G = \{ g \} \]
\[ g : G(K(0) \times TN(0) + K(1) \times TN(1) + K(2) \times TN(2) + K(3) \times TN(3) \geq 0) \]