Contract-Based Integration of Cyber-Physical Analyses

Ivan Ruchkin
Dionisio De Niz
Sagar Chaki
David Garlan

October 14, 2014
14th International Conference on Embedded Software
Contract-Based Integration of Cyber-Physical Analyses

Dionisio de Niz Ivan Ruchkin; Sagar Chaki; David Garlan

Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213

Approved for public release, distribution unlimited

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
Analytic aspect of integration

- 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
Verification domain

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

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

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

- Battery Scheduling
- Thermal runaway

System

- Thread
- CPU
- Battery
Verification domain

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

- Domain $\sigma$ is a many-sorted signature $(A, S, R, T, \{\}\_\sigma)$:
  - $T$: execution semantics – set of sequences of $R$ assignments
    - E.g.: thread scheduler state model for $\sigma_{\text{sched}}$
      battery state model for $\sigma_{\text{batt}}$
  - $\{\}\_\sigma$: domain interpretation for $A$ and $S$
    - E.g.: $\{\text{SchedPol}\}_\sigma = \{\text{RMS, DMS, EDF}\}$
- Architectural model $m$ is an interpretation $\{\}\_m$ of $A$, $S$, and $T$
  - E.g.: $\{\text{Threads}\}_m = \{\text{SensorSample, Ctrl}_1, \text{Ctrl}_2\}$
    $\{\text{CPUBind}\}_m = \{(\text{Ctrl}_1, \text{CPU}_1), (\text{Ctrl}_2, \text{CPU}_2), ...\}$
  - $\{\}\_\sigma \cup \{\}\_m$ 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
  - $:\; \text{means } \Rightarrow$ in case of $\forall$, and $\land$ in case of $\exists$
Outline

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

Scheduling

Thread model checking

Bin packing

Data security

Frequency scaling

Battery

Battery Scheduling

Discharge

Charge

Thermal runaway

Thread

Thread

Thread

CPU

CPU

CPU

Battery

Battery
Frequency scaling assumption

Behavioral equivalence to deadline-monotonic scheduling:

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

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

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

Battery

- 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

\[ \forall b: \text{Batteries} \cdot G \left( \sum_{i=0}^{3} K(b, i) \cdot TN(b, i) \right) \geq 0 \]
Battery modeling

Battery

Battery Scheduling
Discharge    Charge

Selects a battery scheduler
Guarantee: ∀ b: Batteries • G (∑_{i=0..3} K(b, i) * TN(b, i) ) ≥ 0
Verified with battery Promela/Spin model

Determines $K(b, i)$ via simulation

$K(b, i)$
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

Model DB

SMT verification engine

Sched verification engine

Batt verification engine

SMT problem

Z3

Sched Promela model

Spin

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 $\mathcal{T}$ implementation
  – Analysis contracts for multiple views
Contracts

Security Analysis

- $\text{An}_{\text{sec}} \cdot C : I = \{T, \text{ThSecCl}\}, O = \{\text{NotColoc}\}, A = \emptyset, G = \{g\}$
  - $g : \forall t_1, t_2 : \text{ThSecCl}(t_1) \neq \text{ThSecCl}(t_2) \Rightarrow t_1 \in \text{NotColoc}(t_2)$

Multiprocessor scheduling: (Binpacking + scheduling)

- $\text{An}_{\text{sched}} \cdot C : I = \{T, C, \text{NotColoc}, \text{Per}, \text{WCET}, \text{Dline}\}, O = \{\text{CPUBind}\}, A = \emptyset, G = \{g\}$
  - $g : \forall t_1, t_2 : t_1 \in \text{NotColoc}(t_2) \Rightarrow \text{CPUBind}(t_1) \neq \text{CPUBind}(t_2)$

Frequency Scaling

- $\text{An}_{\text{freqsc}} \cdot C : I = \{T, C, \text{CPUBind}, \text{Dline}\}, O = \{\text{CPUFreq}\}, G = \emptyset, A = \{a\}$
  - $a : \forall t_1, t_2 : \text{CPUBind}(t_1) = \text{CPUBind}(t_2) : G(\text{CanPrmpt}(t_1, t_2) \Rightarrow \text{Dline}(t_1) < \text{Dline}(t_2))$

Model checking periodic program (REK):

- $\text{An}_{\text{rek}} \cdot C : I = \{T, C, \text{Per}, \text{Dline}, \text{WCET}, \text{CPUBind}\}, O = \{\text{ThSafe}\}, G = \emptyset, A = \{a_1, a_2\}$
  - $a_1 : \forall t : \text{Per}(t) = \text{Dline}(t), a_2 : \forall t_1, t_2 : G(\text{CanPrmpt}(t_1, t_2) \Rightarrow G \neg \text{CanPrmpt}(t_2, t_1))$

Thermal runaway:

- $\text{An}_{\text{therm}} \cdot C : I = \{B, \text{BatRows}, \text{BatCols}, \text{Voltage}\}, O = \{K\}, A = \emptyset, G = \emptyset$

Battery Scheduling

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