A Multi-Modal Composability Framework for Cyber-Physical Systems

Linh Thi Xuan Phan  Insup Lee

PRECISE Center
University of Pennsylvania
Cyber-physical systems are everywhere…

Our daily lives depend on them!
Characteristic: Timing critical

If unstable, UAV must go to recovery mode and perform the recovery tasks within a deadline.

UAV may crash if a recovery task misses deadline!
Goal

• Provide **timing** and **performance guarantees** for cyber-physical systems

• ...an old problem, but many new **challenges**

**Increasing adaptivity**

**Increasing complexity**
Challenge #1: Adaptivity

- Systems run in multiple **modes** of operation
- Mode transitions triggered by system failure or environment changes
Problem with mode transitions

• Need to execute unfinished jobs of the old mode

⇒ Potential overload and deadline misses immediately after changing mode, even if each mode works correctly in isolation

How to guarantee timing during mode changes?
Challenge #2: Complexity

- Increasing number of software components
- Increasing resource sharing, due to SWaP constraints
- Problem: State space explosion

Source: NASA Study on Flight Software Complexity
Approach: Compositional analysis

- Idea: Analyze compositionally via component abstraction and interface composition
  - Interface exposes only as much information as is required
  - Analysis must capture *mode change* behaviors

- Traditional focus: functional and behavioral aspects
  - e.g., AADL interfaces

- Need abstraction of timing and resource aspects
  - CPS components manage their own resources

- Use *Multi-modal resource-aware interfaces*
Outline

• Motivation

• **Framework overview**
  – Modeling multi-modal components
  – Compositional analysis process

• Abstracting components into interfaces

• Composing interfaces

• Results
Modeling an adaptive component

- **Transition**: triggered by a timing constraint or an event
- **Mode change protocol**

**Mode**: a set of tasks + scheduling policy

**Task parameters**: WCET, deadline, arrival function of input data

Diagram:
- EDF
- FP
- [10, 30] 
- (timing constraint)
- (external event)
- collision-detected
- backlog($B_1$) < 15 
- (buffer constraint)
Mode change protocol

- Formally defines what exactly happens during a mode transition
  - Example: A is not affected, C is released immediately, unfinished jobs of D are discarded
ARINC 653: Compositional analysis

Partition Modes:
- **COLD_START/WARM_START**: Only the initialization process is executed
- **NORMAL**: All processes except the initialization process are executed
- **IDLE**: No process is executed

Mode changes: Triggered by either a partition process or an external event
ARINC 653: Compositional analysis

Captures the resources required to guarantee correct timing behaviors of the partition’s processes
• **Subsystem’s interface:** captures the resources required to guarantee correct timing behaviors of its partitions
Outline

• Motivation
• Framework overview
  – Modeling multi-modal components
  – Compositional analysis overview

• Abstracting components into interfaces
• Composing interfaces
• Results
Single mode: Use service function

- **Service function**: minimum number of resource units must be provided to the mode over any time interval of length $t$ to guarantee schedulability of all tasks.

- **Computed by combining demands of all tasks**

![Diagram showing resource demand and interface of the mode]

- $\beta$: min service required
- Mode
- Task B
- Task A

Interface of the mode

worst-case arrival pattern of A

A: deadline = 4, WCET = 1

0 4 8 12 16  

time

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  

time interval length

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  
# resource units
Multiple modes: A single transition

Demand of each task $T_i$ at mode $m$:

$$\beta_{m,i} = \max \begin{cases} 
[\max \text{Backlog}_m(B_i) + \alpha_{m,i}(t) - \text{sizeof}(B_i)] \cdot \text{wcet}_i \\
\text{initialDemand}_{m,i}(t) \\
\text{wcet}_i \cdot \alpha_{m,i}(t - \text{deadline}_i) 
\end{cases}$$

- No buffer overflow
- No deadline miss during initial duration after moving to mode $m$
- All new tasks meet deadlines
Multiple modes: An automaton

- Component may change modes one after another

**#3:** Need to capture cascading mode changes

Component C

Explore each transition of a reachable mode

Intermediate interface

Fixed point computation
Termination: always guaranteed

Converged

Interface of C
Multiple multi-mode components

- Interface needs to expose **synchronization** events
  - detect incompatible communication
  - avoid overestimating total demands of the composition

- **Composition of interfaces**
  - Compute the product of interfaces
  - Combine service functions of a composed mode
  - Interface refinements
Outline

• Motivation

• Framework overview
  – Modeling multi-modal components
  – Compositional analysis overview

• Abstracting components into interfaces

• Composing interfaces

• Results
Results

Accuracy: No deadline misses during mode transitions

Unimodal analysis can underestimate resource requirements!
Results

**Accuracy:** No deadline misses during mode transitions

**Efficiency:** Analysis is more precise
→ Same guarantees with fewer resources
Open challenges

• Interfaces for multicore, distributed settings

• Moving theory to practice
  – Implementation issues: reduce overheads (e.g., context switches)
  – Connecting with functional interfaces (e.g., AADL)

• Composability for safety certification
  – Apply multi-mode modeling to capture mixed-criticality systems with different certification levels
  – Integrate mode switches with fault-tolerant techniques
  – Combine compositional analysis with assurance cases
Summary

• **Problem**: Systems are becoming complex and adaptive

• **Existing techniques are not sufficient**
  – Focus primarily on composability of functional aspects
  – Do not consider mode change behavior (hence, can be unsafe)

• **Approach**: Compositional analysis based on multi-modal resource-aware interfaces

• **Benefits**
  – Accurate and scalable analysis
  – Analysis results can be used to optimize resource needs
  – Support incremental component-based development