# What's Ahead for Embedded Software?

Author: Edward A. LEE

@ University of California, Berkeley

PRESENTER : DONGYEON SHIN

# Outline

- ◆Introduction
- ◆Frameworks
- ◆HW-SW Partnership
- ◆Real-Time Scheduling
- ◆Interfaces and Types
- ◆Metaframeworks
- ◆More Research Problems

# Introduction

- Embedded software appears in everything
- Engage the physical world, interacting directly with sensors and actuators
- Research issue about embedded software
  - “How to reconcile a set of domain-specific requirements with the demands of interaction in the physical”

# Frameworks

- A set of constraints on components
- A set of benefits that derive from those constraints
- Defines a mode of computation, which governs the interaction of components
- “Understanding suitable models of computations is to understand what makes a framework useful for embedded system design. “

# Frameworks

- Four service categories
  - Ontology : Defines what it means to be a component
  - Epistemology – State of knowledge
  - Protocols : Provides mechanisms that dictate how components interact
  - Lexicon : Vocabulary of component interaction
- ”The more constraints there are, the more specific it is. Ideally, this specificity comes with benefits”
- Key challenge – Invent frameworks match the application domain

# Frameworks

- Concurrency
  - No universal concurrent framework

  - Von Neumann framework
    - It reduces time to a total order of discrete events for correctness

  - In practices, partially ordered at best.
  - Partial ordering makes it difficult to maintain a global system state.

# Frameworks

- Sample frameworks
  - Most designers are exposed to only one or two frameworks.
  - Diversity make it hard to select a framework
  - Ex) Time
    - View time as a real number
    - View time as discrete
    - View time as partially ordered

# Frameworks

- Mixing frameworks
  - Create the union of all the frameworks – extremely complex and difficult to use
  - Choose one concurrent framework and show that all the others are special cases of that – does not acknowledge each model's strengths and weakness
  - Use an architecture description language – design are ADL are a poor match
    - Ex) does not cleanly describe asynchronous message passing
  - Heterogeneously mix frameworks – instead of forming union, preserve their distinct identity
    - Ex) hybrid systems combine finite state machines with continuous time model

# Hardware – Software Partnership

- Since the 1970s, functionality has steadily shifted from hardware to software
- Software
  - Primarily sequential execution with a single instruction stream
    - HW resources are multiplexed in time to perform a variety of functions
- Hardware
  - Primarily parallel execution
    - HW resources are not shared

# Hardware – Software Partnership

- Designer's task to balance between their sequential and parallel execution styles
- In theory, as embedded processor performance improves, there should be less need for such HW specialization
- OS cannot reliably handle many hard-real-time tasks
  - Component interface definitions need to declare temporal properties, not just a fixed priority
  - Compositions of components must have consistent and non-conflicting temporal properties

# Real-Time Scheduling

- It provides some assurances of timely performance given certain component properties
- Rate-monotonic scheduling principle
  - It translate the invocation period into priorities
  - Most methods are not compositional
- Priority inversion
  - Processes interact by entering a monitor to exclusively access a shared resource
  - Priority-based scheduling scheme
    - Processes interact both through the scheduler and through the mutual-exclusion mechanism
    - No coherent compositional semantics, which points to the need for a different scheduling mechanisms entirely

# Interfaces and Types

- Formal methods to ensure software’s correctness
- Type systems talk only about static structure
- Types system techniques
  - Constrains what a component can say about its interface
  - How to ensure compatibility when designers compose components
  - Describe an interface’s dynamic properties using nondeterministic automata
    - Interface properties -> partial order

# Interfaces and Types

- Strong typing
  - Extending type systems to program dynamics.
  - How to achieve modularity and reuse without discarding strong typing
    - use polymorphism, reflection, and runtime type inference and type checking
  - Key part of future embedded software research
    - Sufficient syntactic languages support

# Metaframework

- All frameworks impose some constraints to achieve certain benefits
  - Stronger constraints, stronger benefits
  - They are unlikely to solve all the framework problems for any complex system
- Mix frameworks heterogeneously
  - One framework is simply a more restricted version of another
  - Mix frameworks hierarchically
    - Ex) Ptolemy project at UC Berkeley

# More research problems

- Human-computer interaction
- Networking problem
- Hardware and software design techniques that minimize power consumption