What's Ahead for Embedded Software?

Edward. A. Lee, 2000 IEEE
University of California
November 06, 2014 Kihyun Im

Contents

- Introduction
- Frameworks
- Hardware-Software Partnership
- Real-Time Scheduling
- Interfaces and Types
- Metaframeworks
- Conclusion

Introduction

- What is Embedded Software?
  - Main task is to engage the physical world
  - Computer software what is specialized for the particular hardware
  - Interact directly with sensors and actuators
  - Example of Embedded Software :
    - Cars, Smartphones, Robots, Smart TVs, etc.

Introduction

- How to reconcile a set of domain-specific requirements with the demands of interaction in the physical world?

- How to adapt software abstractions to meet requirements?
  - Real-Time constraints
  - Concurrency
  - Stringent safety considerations

Frameworks

- Component
  - Any kind of building block
    - Example : A set of functions, modules, subroutines, etc.

- Definition
  - A set of constraints on components and interaction
  - A set of benefits that derive from those constraints

Frameworks

- Most frameworks have four service categories :
  - Ontology
  - Epistemology
  - Protocols
  - Lexicon

Frameworks

- Ontology
  - Defines what it means to be a component
  - Subroutine, State transform, Process, Object

- Epistemology
  - Defines state of knowledge
  - Sharing information, connectivity

Frameworks

- Protocols
  - How components interact?
  - Message passing, Semaphores, Timed events
- Lexicon
  - Vocabulary of component interaction
  - COBRA programming language

Frameworks

- Frameworks may be very broad or very specific
  - The more constraint, the more specificity
  - The more specificity, the more benefits
  - Examples
    - UNIX Pipe
      - Do not support feedback structures, but no deadlock
    - Internet
      - Constraints on lexicon & protocol, but platform independence

Frameworks

- Key challenge
  - Invent frameworks with properties that better match the application domain
  - Requirements
    - Reintroduction of time
    - Recognize that certain essential properties

Frameworks

Concurrency

- Framework with concurrency can perform some computation in parallel
  - Advantage : Execution time
  - Disadvantage : Complicate system design

- Von Neumann Framework
  - Sequential computation and execution
  - Total ordering is needed because of correctness

- Distributed Systems
  - Total ordering is expensive, partially ordering is best
  - Difficult to maintain a global system state

Frameworks

Sample Frameworks

- So far, designers exposed to few frameworks
- Design practices changing
  - Domain-specificity is rising
- Example : Different views of Time
  - Explicit view : Real number
  - Abstract view : Discrete

Frameworks

Mixing Frameworks

- Grand unified approach to modeling
  - Find a concurrent framework that serves all purpose

- Approaches
  - Create the union of all the frameworks
  - Choosing one concurrent framework
  - Using Architecture Description Language
  - Heterogeneously mixed frameworks

HW-SW Partnership

- Functionally has steadily shifted from HW to SW

- Software
  - Primarily sequential execution
  - Vary of functions use same HW resource

- Hardware
  - Primarily parallel execution
  - Resources are not shared

- Designer's Task
  - To explore balance between two styles

HW-SW Partnership

- Performance and specialization
  - If embedded processor performance improves
    -> Less need for hardware specialization

Real-Time Scheduling

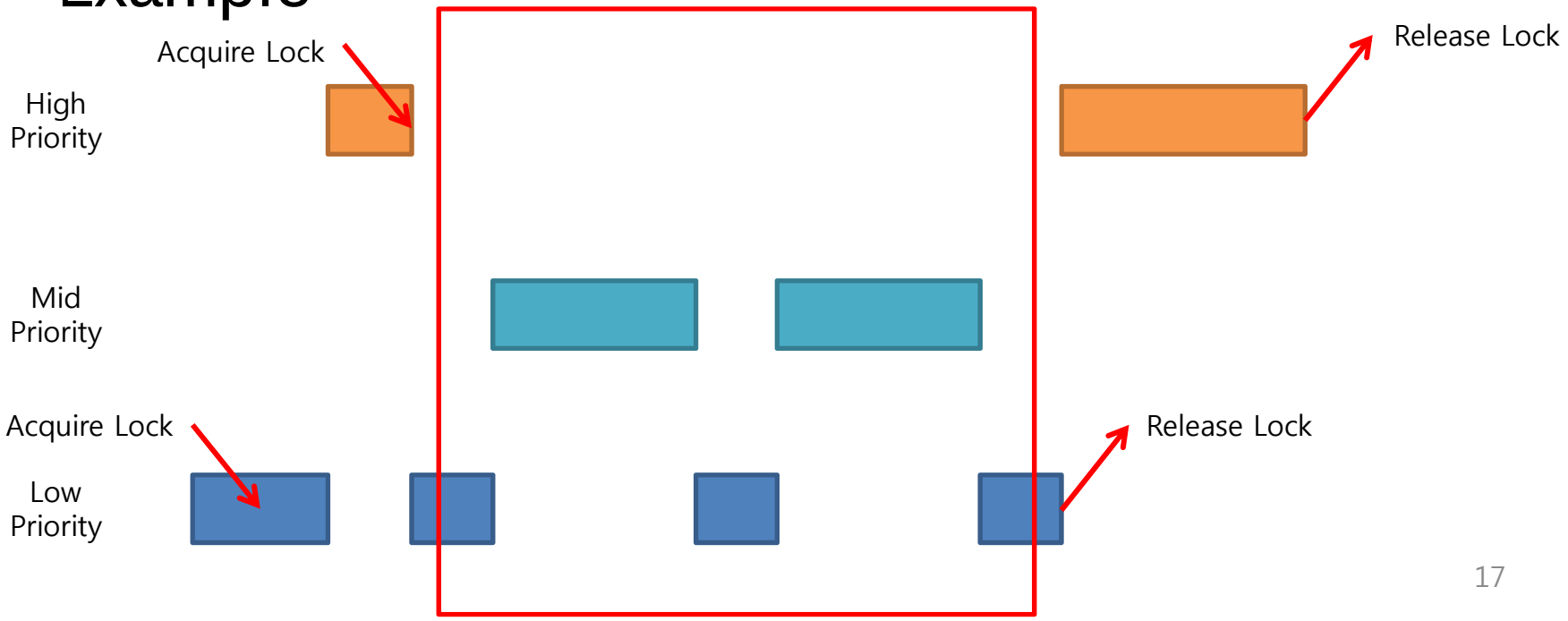
- Real-Time Scheduler
  - Provides assurance of timely performance given certain component properties
    - Example : Component's invocation period and Task deadlines

- Problem
  - Most methods are not compositional
  - Example : Priority Inversion

Real-Time Scheduling

- Priority Inversion
  - The low-priority task blocks the high-priority task
  - Due to shared resource contention

- Example



Interface and Types

- Type system
  - Ensure correctness of software
  - Provide a vocabulary for talking about larger structure

- Disadvantage of embedded software
  - Type systems talk only about static structure
  - Nothing about concurrency or dynamic

Interface and Types

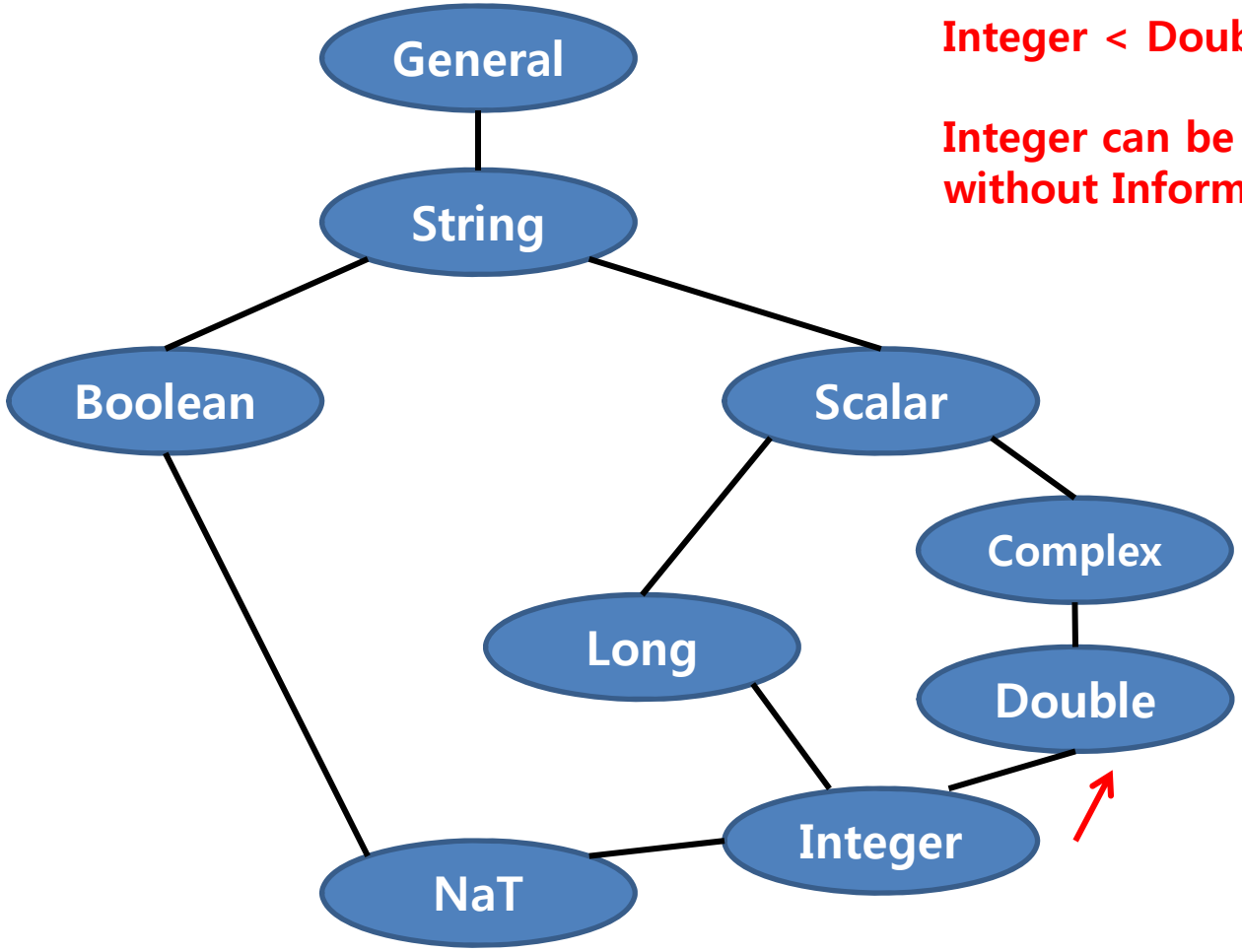
Type system techniques

- Type system constraints
  - What a component can say about its interface
  - How to ensure compatibility

- How a type system works
  - Data-level type system
    - A data type is "less than" another type if it can be converted to the other type without loss of information
  - System-level type system
    - A type is less than another if the other simulates first

Interface and Types
Type system techniques

- Data-Level Type System

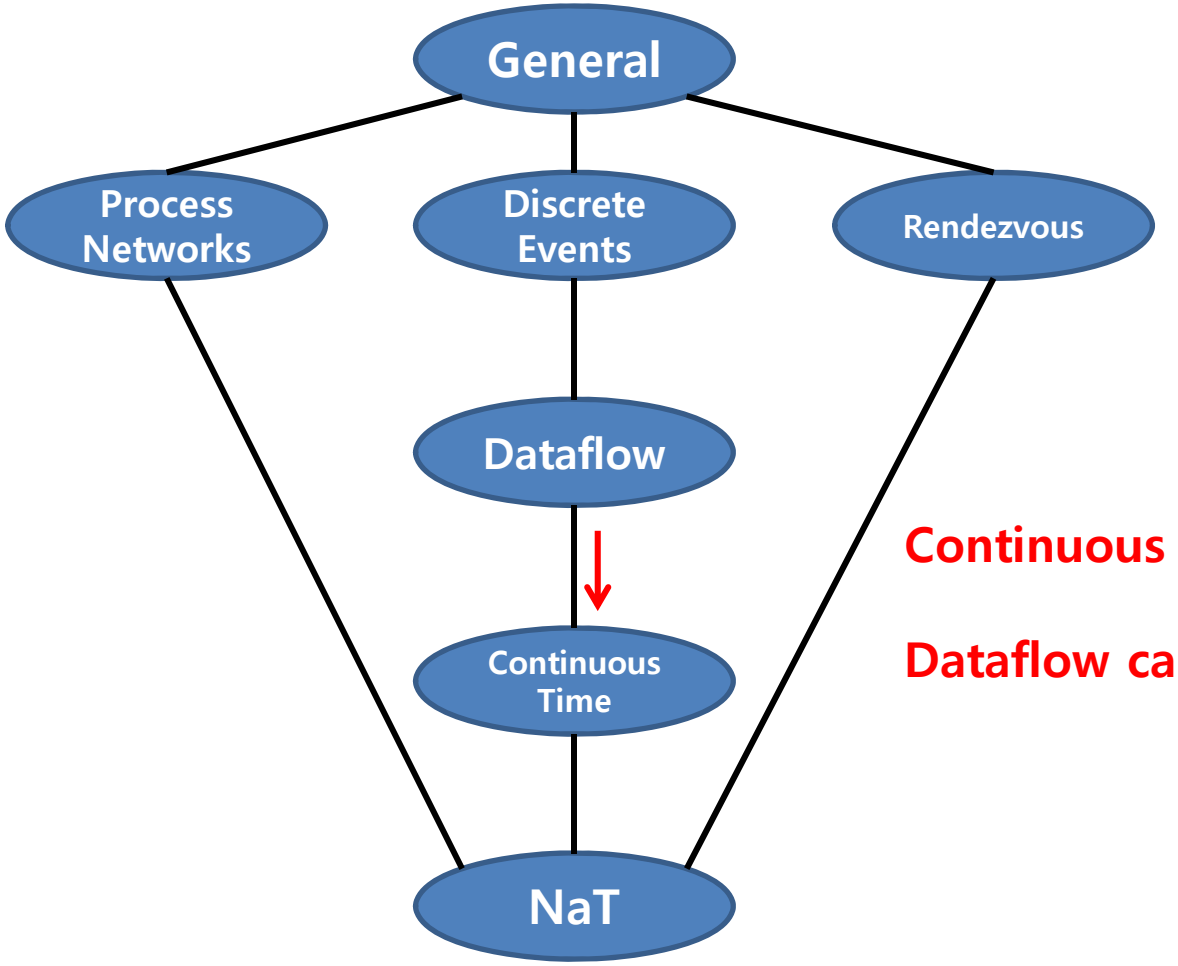


Integer < Double

Integer can be converted to Double without Information Loss

Interface and Types
Type system techniques

- System-Level Type System



Continuous Time < Dataflow

Dataflow can simulate Continuous time

Interface and Types

The case for strong typing

- Strong typing (Java, ML)
  - Emphasize catching errors ASAP
  - Compiler catches errors
  - Weakness to dynamic errors
    - Array out of bound

- Without strong typing (Lisp, Tcl)
  - Modularity and reusability
  - Identifying the source of problem can be difficult

Interface and Types

The case for strong typing

- To overcome strong typing's weakness
  - Obtain modularity and reusability with strong typing
    - Polymorphism
      - Same data structure can contain different types
    - Reflection
      - Examine and modifying the structure and behavior of the program at runtime
      - C#, Objective-C, Smalltalk, etc.
    - Runtime type interface and type checking
  - Components must give their dynamic properties as part of interface definition

Metaframework

- Stronger constraints -> stronger benefits
  - Framework become rather specialized as they seek these benefits
  - But unlikely to solve all the problems for any complex system

- Avoiding give up the benefits of specialize
  - Mix frameworks heterogeneously
    - Through specialization
    - Mix frameworks hierarchically

Metaframework

- Ptolemy project at UC berkeley
  - Hierarchical approach
  - Domain-polymorphism
    - Domain : A framework using software infrastructure
    - 22 domains are contained in Ptolemy II
    - If a component can operate multiple-domains, it is called domain-polymorphic

- The Gravity system / Orbit
  - Mix computation modeling techniques heterogeneously
  - Facet : A model in domain
    - Heterogeneous models are multifaceted designs

Conclusion

- There are more research problems
  - Human-Computer interaction
    - Embedded software becomes transparent
  - Configurable hardware
    - Selecting appropriate computational models
  - Network problem
    - Providing quality-of-service in face of unreliable resources
  - HW & SW design technique
    - Power consumption are critical for portable devices

Conclusion

- The author focused on
  - Constructing embedded software
  - And the focus moves beyond a functional correctness

- The key problem in the future
  - Identifying the appropriate abstractions for representing the design