What’s Ahead for Embedded Software?

2006.10.25(Wed)
Gilsoo Kim
gskim@mmlab.snu.ac.kr
Contents

- Introduction
- Frameworks
- Hardware-Software Partnership
- Real-time Scheduling
- Interfaces and types
- Meta framework
- Conclusion
Introduction

- Embedded software
  - Main task is to engage the physical world
  - Interact with sensors and actuator

- Research for embedded software
  - Past days: too small and retro area
  - Nowadays: complex and pervasive enough

- Most pressing problem

  How to adapt existing software techniques to meet the challenges of the physical world
Frameworks

- Framework
  - Defines models which governs the interaction
  - Supports component interaction mechanism
    - Ex) OS, Programming language, etc.

- Component
  - Any kind of building block (broader view)
    - Ex) Program, process, language primitive, etc.

- Framework and component in embedded software community must be seen
  - To get useful model
Frameworks

- Four service categories
  - Ontology: what it means to be a component
  - Epistemology: states of knowledge
  - Protocols: how components interact
  - Lexicon: vocabulary of component interaction

- More constraints $\rightarrow$ More specificity

- Properties of aggregate of components
  - Difficult to characterize

- The key challenge is to invent framework that better match the application domain
Frameworks

Concurrency

- Concurrency is useful in designing embedded systems
- Von Neumann framework
  - Universally accepted model of sequential computation
  - Total ordering (\(\uparrow\))
- In practice → Partially ordered at best
- Thus, practical designs will almost certainly have to combine techniques
  - ‘actors’ and ‘interaction patterns’
Frameworks

Sample frameworks

- So far, most designers are exposed to only one or two frameworks
- But, design practices are changed
  - Level of abstraction and domain-specificity rise
- Rich variety of frameworks
  - Myriad views being offered
  - Designers will need some way to reconcile those
- Example: different view of ‘Time’
  - Explicit vs. abstract
Frameworks (1/2)

Mixing frameworks

- A grand unified approach to modeling
  - Seek a framework that serves all purpose

- Possible approaches
  - Union of all the frameworks
    - Complex and hard to use
    - Designing would be difficult
  - Choose one concurrent frameworks and shows that all the others are special cases of that
    - Feasible way relatively easy to use
    - Sufficiently expressive to subsume others
    - Doesn't acknowledge each model’s strength and weakness
Frameworks (2/2)

Mixing frameworks

- Another approaches
  - Architecture Description Language (ADL)
    - Describe the component interactions
    - Provide a way to get good insight into the design
    - Poor match could be appear
  - Mix heterogeneously, but preserve their distinct identity
    - Finite time machines + continuous time model
Hardware-Software Partnership

- Software vs. hardware
  - S/W: sequential execution
  - H/W: parallel execution

- Most embedded system involve both design
  - Designer should balance b/w sequential and parallel execution styles
  - Rethink multitasking
    - Component interface definitions need to declare temporal properties
    - Composition of components must have consistent and non-conflicting temporal properties
Real-time Scheduling

- Real-time scheduler provides some assurances of time performance given certain component properties
- Priority inversion
  - There is no systematic way to provide assurance for the aggregate
    - Possible way can make deeper failure
- Need for a entirely different scheduling mechanism
Interface and types

- Type system
  - Ensure software’s correctness
  - Provides a vocabulary for talking about larger structure

- Type systems talk only about static structure for embedded software
  - Syntax of procedural programs
  - Nothing about concurrency or dynamics

- Work with active object and actors moves a bit in the right direction
  - Not enough about interface to safety, liveness, consistency, or real-time behavior
Interface and types

Type system techniques

- Type system typically defined by a subtyping relation or by lossless convertibility

- How a type system works
  - Data-level system type
  - System-level type

- Depend on partial order
Interface and types
The case for strong typing

- Many errors are not within the type system’s scope
  - Accessing an array out of bound..
  - Several researchers have shown how to augment type system

- Strong typing vs. without strong typing
  - Modularity and reuse / safety

- One solution
  - Polymorphism, reflection, and runtime type inference and type checking
Meta framework

- Stronger benefits come at the expense of stronger constraints
- Mix frameworks heterogeneously
  - Through specialization (analogous to subtyping)
  - Mix frameworks hierarchically
- Domain polymorphism
  - Avoids pairwise design
  - Can operate in multiple domains
  - Still have clear semantic in other domain
Conclusion

- There are more problems
  - Human-computer interaction
  - Configurable hardware
  - Networking problem (QoS)

- The focus must move beyond a program’s functional correctness

- The key problem then becomes identifying the appropriate abstractions for representing the design
Q & A

- Thank you