What’s Ahead for Embedded Software?

Author: Edward A. Lee
@ University of California, Berkeley

Presenter: Minwoo Kwak
(2013-20742)

2015-11-17
Outline

- Introduction
- Frameworks
- HW-SW Partnership
- Real-Time Scheduling
- Interfaces and Types
- Metaframeworks
- Conclusion
What is *Embedded Software*?
- The software which engages the physical world by interacting directly with sensors and actuators.
  - Which has taken over what mechanical & dedicated electronic systems used to do.
- ex. telephones, pagers, systems for medical diagnostics and climate control

Why *Embedded Software* research now?
- Once deemed too small and retro for research
- Grown complex and pervasive enough to attract the computer scientists
Research issue about embedded software

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

✓ “How do you adapt software abstractions to meet the requirements?”
  ▪ Real-time constraints
  ▪ Concurrency
  ▪ Stringent safety considerations

✓ The answer to the question has given rise to some promising research angles.
Component
- Any kind of building block
- ex. set of functions, modules, subroutines

Framework
- A set of constraints on components and their interaction
- A set of benefits that derive from those constraints
- Defines a model of computation, which governs the interaction of components

The first step in understanding suitable models of computation is to understand what makes a framework useful for embedded system design.
Most frameworks have four service categories:

✓ Ontology: what it means to be a component
  ▪ ex. subroutine, state transformation, process, object

✓ Epistemology: state of knowledge
  ▪ ex. sharing information, scoping rules, connectivity

✓ Protocols: how components interact
  ▪ ex. rendezvous, semaphores, monitors, timed events

✓ Lexicon: vocabulary of component interaction
  ▪ ex. type system
A framework may be very broad or very specific

- The more constraints, the more specificity
- The more specificity, the more benefits

Examples

- UNIX pipe: Not support feedback structure, but no deadlock
- Internet: Constraints on lexicon (byte stream), protocol (HTTP), but provides platform independence

KEY: “To invent framework that better match the application domain”

- Requirements
  - Reintroduction of time
  - Recognize of essential properties when components become an aggregate
Concurrency

✓ A framework with concurrency can perform some computation in parallel.
  ▪ However, concurrency also seriously complicate system design.

Examples for concurrency

✓ Von Neumann framework
  ▪ A universally accepted model of sequential computation
  ▪ It reduces time to a total order of discrete events for correctness

✓ Distributed systems
  ▪ Maintaining such a total order globally is expensive
  ▪ Events are partially ordered at best.
  ▪ This partial ordering makes it difficult to maintain a ‘global system state’.
Sample frameworks

- So far, most designers are exposed to only one or two frameworks.

- But, design practices has changed
  - the level of abstraction and domain specificity rise-

- The diversity will make it hard to select a framework.
  - Designers need some way to reconcile the views-

- Example answer: Different views for ‘Time’
  - Explicitly: as a real number
  - Abstractly: as a discrete number
Mixing frameworks

A grand unified approach to modeling would seek a concurrent framework that serves all purposes.

Possible approaches

- To create the union of all the frameworks:
  - Complex and hard to use (+Design would be difficult)

- To choose one concurrent framework and show that all the others are special cases of that:
  - Relatively easy to use
  - But it doesn’t acknowledge each model’s strengths and weaknesses

- To use an Architecture Description Language (ADL):
  - Describe the component interactions
  - It provides a good insights into the design, and sometimes it gives poor match.

- To heterogeneously mix frameworks, preserving their distinct identity
Since 1970, functionality has steadily shifted from HW to SW.

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. (or at least, not as much)

Most embedded systems involve both HW and SW design, a designer’s task is to explore the balance between the two styles.
For hard-real-time functions (i.e., signal processing), designers often assign concurrent tasks to distinct processors.

- ex. the speech coders and radio modems in a digital cellular telephone

In theory, as embedded processor improves, there should be less need for such HW specialization.

- Until then, designers must use dedicated HW or use processors that so greatly exceed minimum performance.

However, Real-Time OSs cannot yet reliably handle many hard-real-time tasks.

- The embedded system community must rethink multitasking.
  - Component interface need to declare temporal properties, not just a fixed priority.
  - Compositions of components must have consistent and non-conflicting temporal properties.
◆ Real-time scheduler

✓ It provides assurance of timely performance given certain component properties.
✓ ex. A component’s invocation period or task deadlines

◆ Rate-monotonic scheduling principle

✓ It translates the invocation period into priorities.
✓ Priorities may also be based on semantic information about the application.

◆ Problem: most methods are not compositional.

✓ A method can provide assurances individually to each component.
✓ There is no systematic way to provide assurance for the aggregate of the two or more components.
✓ ex. priority inversion
Real-Time Scheduling

◆ Priority Inversion

✓
✓
Interfaces and Types

◆ Type systems
  ✓ One of the great practical triumphs of contemporary software.
  ✓ Ensure correctness of software
  ✓ Provide a vocabulary for talking about larger structure

◆ Disadvantage for embedded software
  ✓ Type systems talk only about static structure
    - the syntax of procedural programs

  ✓ There is nothing about the program’s concurrency or dynamics.

  ✓ Work with active objects and actors moves a bit in the right direction
    ▪ But it does not say enough about interfaces to ensure safety, liveness, consistency or real-time behavior
Interfaces and Types

◆ Type system technique

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

✓ How a type system works
  ▪ Data-level type system
    : subtyping relation or lossless convertibility

  ▪ System-level type system
    : dynamic properties using non-deterministic automata
    – A type is less than another if the other simulates first
How a type system works: Data-level type

A data type is “less than” another type if it can be converted to the other type without loss of information.

ex. Integer < Double
 Interfaces and Types

◆ How a type system works: System-level type

- Domain polymorphic
  - Process networks
  - Discrete events
  - Rendezvous
  - Dataflow
  - Continuous time
  - NaT

A system type is “less than” another if the other simulates the first.

ex. Continuous time < Dataflow
The case for strong typing

- Strongly typed languages (i.e., Java, ML)
  - Emphasize catching error ASAP—often the compiler catches them
  - Vulnerable to other programming errors
    - ex. accessing an array out of bounds
  - Compromise modularity and discourages reuse

- Languages without strong typing (i.e., Lisp, Tcl)
  - Emphasize modularity and reusability
  - Difficult to identify the source of the problems and guaranteeing the code may be impossible

For embedded systems, the extra degree of safety that strong typing offers overwhelms even the desire for modularity and reuse.
  - The question then becomes how to achieve modularity and reuse without discarding strong typing.
  - to use polymorphism, reflection, and runtime type inference and type checking
**Metaframeworks**

- **Stronger benefits come at the expense of stronger constraints.**
  - Frameworks become rather specialized as they seek these benefits.
  - **Drawback**: They are unlikely to solve all the framework problem for any complex system.

- **To avoid giving up the benefits of specialized frameworks, designers will have to mix frameworks heterogeneously.**
  - Through specialization (= subtyping)
  - To mix frameworks hierarchically

- **Examples**
  - Ptolemy project at UC Berkeley
  - The gravity system and its visual editor Orbit
We have studied some interesting embedded system research problems.

The author has focused on constructing embedded software, since it become a first-class of programming exercise.
- Embedded system designers need more!

The focus must move beyond a program’s functional correctness to its temporal correctness.

The key problem then becomes identifying the appropriate abstractions for representing the design.