

Verification and Integration for Real-Time Control Software

Rajeev Alur
University of Pennsylvania

November 2006



Challenges for NECS

1. Correctness of software

- ❖ Programming abstractions for real-time, security...
- ❖ Formal verification of models and code

2. Integration of components

- ❖ Rich interfaces and composition

Formal Verification for Improving Reliability

- Specifying and verifying correctness
 - ❖ Mathematically precise specifications
 - ❖ Model checking, theorem proving, runtime monitoring...
- Recent success in hardware and systems software
 - ❖ Standardized specification languages for EDA: PSL
 - ❖ Program analysis tools for OS code
- Networked embedded control systems
 - ❖ In search of “killer app” to spur real industrial interest
 - ❖ Beyond safety: Real-time, Performance, Trust

Model-based Design

- Modeling of real-time control software
 - ❖ Raise the level of programming abstraction
 - ❖ Design automation
 - ❖ Detecting design errors at early stages
- Hybrid systems foundations
 - ❖ Integrated modeling of discrete software + continuous environment
 - ❖ Tools from control theory and computer science
- Ongoing research
 - ❖ Tools for model checking
 - ❖ Principles for compositional reasoning
 - ❖ Principles for integrating heterogeneous models

From Control Models to Reliable Systems

- ❑ Code generation from control models
 - ❖ Semantic gap: do verification results for model hold for code?
 - ❖ Distributed platforms
- ❑ Exposing communication/computation constraints of the platform at design level
 - ❖ What are appropriate abstractions?
 - ❖ How to make control designs reusable?
- ❑ Integrating control components
 - ❖ What should be an interface for an embedded software class
 - ❖ Must capture real-time/QoS constraints

Interfaces for Control Components

- ❑ **Classical solution: Periodic tasks**
 - ❖ Successful as a way of separating concerns between control engineers and RTOS
 - ❖ Challenges with predictability and composition
- ❑ **Formal methods solution: Behavioral interfaces**
 - ❖ Interface captures allowed sequences of inputs/outputs along with timing
 - ❖ Challenge: Tools, Scalability, Complexity
- ❑ **Opportunities: Time-triggered platforms**
 - ❖ Slotted allocation of computation and communication resources
 - ❖ Programming abstractions can exploit predictability
 - ❖ Current research: automata-based interfaces for scheduling