Safe and Reliable Embedded Systems

Gernot Heiser

Australian Government

Department of Communications, Information Technology and the Arts

Australian Research Council

NICTA Members

ANU THE AUSTRALIAN NATIONAL UNIVERSITY

UNSW THE UNIVERSITY OF NEW SOUTH WALES

NSW First for Business — Department of State and Regional Development

Victoria The Place To Be

THE UNIVERSITY OF MELBOURNE

The University of Sydney

Queensland Government

Griffith UNIVERSITY

QUT Queensland University of Technology

THE UNIVERSITY OF QUEENSLAND AUSTRALIA

NICTA Partners

Modern Embedded Systems

- Ubiquitous
  - dozens per person, part of everyday life
- Increasinly dependent on correct operation
  - security of data
    - protection of personal information
    - protection of valuable media content
  - device safety
    - faulty devices can injure or kill
    - faulty devices can interfere with wireless networks
  - device reliability
    - annoyance
    - cost to reputation, cost of recalls

Embedded Systems Challenges

- Embedded-systems functionality is exploding
- Software complexity is growing strongly
  - millions of lines of code
  - gigabytes of embedded software
- Complexity is the enemy of reliability
  - trustworthiness becomes harder to achieve
- Many embedded systems become open
  - user-installed untrusted software
- Faults require remote software upgrades
  - increased security problems
- Software cost requires component reuse across domains
  - especially OS software, comms stacks, GUIs etc

Problems with Existing Assurance

- Many domain-specific standards
  - impedes component re-use
- General standard (Common Criteria) only security-focussed
  - information flow, not reliability & safety
- Lack modularity
  - mostly certify systems, not components
  - doesn't scale to many millions of lines of code
- Focus on inputs, not outputs of development
  - concern with process and history, not semantics
- Assurance gap
  - prove properties of models, not code

Scalable Approach to Trustworthiness

- Small trusted computing base (TCB)
  - based on small, high-assurance operating system
  - *prove* correctness of TCB (i.e. its code)
  - prove once, use in arbitrary domain
- Design by composition
  - component behaviour restricted by TCB — provably!
  - can make guarantees about fault containment
  - can formally reason about composition
  - can *prove* correctness of critical components *in isolation*
  - re-use components and their verification
    - reduce cost of assurance
    - increase level of assurance
- Standards need to recognise and support this!

Requirements for Standards

- Domain-independent assurance
  - assure functionality, no matter what it is
  - assure all critical functionality, not just security
- Assurance of code
  - assure functionality, not process
- Real proof
  - testing proves presence, not absence of faults
- Assure components and their compositions
  - make assurance scalable and re-usable