Does it Pay Off?
Model-Based Verification and Validation of Embedded Systems!

Frits Vaandrager

Radboud Universiteit Nijmegen

PROGRESS Minisymposium, Eindhoven, 31 May 2006
Contents

Introduction

A Spectrum of Formal Methods

PROGRESS Project on V&V: HaaST

Conclusions
Embedded Systems Design

In general very complex task

- Failure of embedded systems often may have serious consequences (loss of lives, huge financial losses)
- Correctness and reliability are of vital importance
- It is common that > 75% of development cost go in V&V
Validation and Verification

**Validation:**
- Increase confidence in correct operation of implementation
- *Are we building the right system?*

There exist two basic validation strategies:

- **The objective of verification** is to show that implementation possesses a property prescribed by the specification: *Are we building the system right?*
- **The objective of falsification** is to show that the negation of a specification requirement holds in an implementation
- In practice, falsification weaker than verification
Models

Provide (mathematical) abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on relevant ones

- All forms of engineering rely on models to understand complex real-world systems
- Boosted by UML and advent of MDD, role of models in design of computer based systems has become much more important recently, and this is a very positive development
- Great opportunity for V&V: formal verification
Simulation remains the main tool to validate models, but the importance of formal methods for V&V is growing, especially for safety-critical systems

- Simulation of embedded systems is **challenging** because they are heterogeneous
- There is a lot to say about simulation! I will focus on formal verification because this is the main topic within the PROGRESS projects that I have been asked to discuss.
Mathematics has always been of great importance in engineering

- **Formal methods** is the applied mathematics of computer system engineering
- I focus on formal methods for V&V
- Most software engineering projects hold formal methods at arm’s length unless they involve critical systems
- Mathfobia? Lack of training? Methods not cost-effective?
Berry suggested to use term **automatic bug detection** in place of **formal verification** to underscore that it is too much to hope for a conclusive proof of any nontrivial design.

Instead the goal of formal verification should be a technology that will help designers to prevent problems in deployed systems.

The paradox is that **verification** at the level of a formal model often amounts to **falsification** of the real system!
The Formal Methods Approach

- Use **symbolic calculation** to provide **cheaper and better** methods of verification for software and systems
- A single symbolic calculation can subsume many individual numeric cases
  - Just as \( x^2 - y^2 = (x - y) \times (x + y) \)
  - Subsumes \( 36 - 16 = 2 \times 10 \) and \( 49 - 4 = 5 \times 9 \) and ...
- Symbolic calculation is mechanized using the methods of automated reasoning: **theorem proving, model checking, constraint solving**, etc.
- There has been sustained progress in these fields for several decades and they have recently broken through the barriers to practical application
A Spectrum of Formal Methods

Assurance

Effort

invisible formal methods

model checking

automated abstraction

theorem proving
Interactive Theorem Proving

Requires great skill and resources

- Can solve very hard problems
- **Verification of floating point operations of Intel’s Itanium processor**
- **Verification of software for Java Card smartcards**
- Probably not cost-effective for Dutch industry
- Important area for academic research
Model Checking

Analysis is automatic but must specify the model and property

- Can search huge state spaces (trillions of reachable states) efficiently
- Still state space explosion is the enemy
- Can also handle real-time, probabilistic and hybrid systems
- Numerous successful applications
- Cost-effective in many cases
Some Uppaal Case Studies from Nijmegen

Model checker for timed automata developed by Universities of Uppaal and Aalborg, with recent contributions by Nijmegen. Case studies we did include:

- Bang & Olufsen protocol
- biphase mark protocol
- IEEE 1394 “Firewire”
- distributed agreement protocol
- scheduling of lacquer production at Axxom
- throughput optimization for a wafer scanner from ASML
- car periphery supervision system from Bosch
- architecture evaluation for a distributed in-car navigation system by Siemens
Automated Abstraction

To check large systems, abstraction is a key paradigm

- Generic abstractions (e.g. symmetry reduction) greatly enhance applicability of model checkers
- SAL tool is attempt to bridge gap between model checking and theorem proving
- New technique: counterexample guided abstraction
- Software model checking tools SLAM and BLAST used within Microsoft for debugging device drivers
- Not yet off-the-shelf technology
Invisible Formal Methods

Model based development provides the artifacts needed for automated analysis

- Engineers prefer **push-button** V&V technology
- Some very **sophisticated techniques** have been proposed to make this possible
- Convenience more important than generality. Tools will not find all bugs in your design but they will find most of them fast and automatically

- Example: **visualState** from IAR systems
- Example: **Extended Static Checker for Java (ESC/Java2)**
Correctness of Implementations

Bridging the gap between high-level modelling abstractions and implementation platforms is one of the key challenges for embedded software research.

How do we know that the generated code is actually correct and meets real-time constraints?

Solution requires formal methods
Model Based Testing

Aims at automatic creation, execution and evaluation of test cases.

- Claimed benefits are better coverage, faster and cheaper testing
- Very important technique, will eventually find its way to all MDD tools
- Spec Explorer helped to discover 10 times more errors including deep system level bugs
- Pretchner: No published evidence that promises of MBT are kept
Models!

Useful for

- Building systems
- Predicting their behavior (V&V)
- Monitoring their behavior
- Diagnosing faults (FINESSE)
- Model based control
- ...

Frits Vaandrager  Model-Based Verification and Validation
PROGRESS Projects on V&V

- CES.5009: Real-time Distributed Shared Data Space
- TES.4999: Verification of Hard and Softly Timed Systems (HaaST)
- CES.5008: Improving the Quality of Embedded Systems Using Formal Design Techniques
- EES.5141: Specification Tooling for Embedded Software Components
- TES.5417: Atom Splitting in Embedded Systems Testing
- DES.7015: Fault Diagnosis for Embedded Systems Dependability
HaaST Case Study: Address Configuration in Zeroconf

Protocol for **dynamic configuration** of IPv4 link-local addresses

**Standardized** by IETF in RFC 3927

**Philosophy**: internet should be like electricity, i.e., work when you plug in a cable

Several implementations available, notably **Bonjour** from Apple

See [www.zeroconf.org](http://www.zeroconf.org)
Our society increasingly depends on correct functioning of (implementations of) communication protocols.

Standards that define these protocols are written in informal language, with frequent ambiguities, omissions and inconsistencies.

We can blame the engineers (for not using formal methods), the companies (for playing political games), but also formal methods researchers (for using obscure notations and model hacking).
Our Results

1. Simple Uppaal **model** of critical part of Zeroconf, almost good enough for inclusion in standard
2. Very close **correspondence** between model and standard; only probabilistic aspects cannot be handled
3. Several **mistakes/ambiguities** found in standard
4. Manual **verification** easy, model checking difficult (atypical!)
5. Several suggestions for further **improving** TA technology
Zeroconf Address Configuration

counter < PROBE_NUM &&
x >= PROBE_MIN
send_req!
packet.senderHA := j,
packet.senderIP := 0,
packet.targetIP := IP[j],
packet.request := true,
counter++,
x := 0

counter == PROBE_NUM
x := PROBE_MAX

counter := 0,
x := PROBE_WAIT

COLLISION

x <= RATE_LIMIT_INTERVAL

counter <= ANNOUNCE_WAIT

counter := 0,
ConflictNum := 0,
x := ANNOUNCE_INTERVAL

counter < ANNOUNCE_NUM imply
x <= ANNOUNCE_INTERVAL

send_req!
packet.senderHA := j,
packet.senderIP := IP[j],
packet.targetIP := IP[j],
packet.request := true,
counter++,
x := 0,
UseIP[j] := true

counter := 0,
x := PROBE_WAIT

ConflictNum < MAX_CONFLICTS
urg!
ConflictNum++

ConflictNum >= MAX_CONFLICTS &&
x >= RATE_LIMIT_INTERVAL

counter < ANNOUNCE_NUM &
x == ANNOUNCE_INTERVAL

send_req!
packet.senderHA := j,
packet.senderIP := IP[j],
packet.targetIP := IP[j],
packet.request := true,
counter++,
x := 0,
UseIP[j] := true

INIT

WAIT

x <= PROBE_WAIT

x := PROBE_WAIT

x := PROBE_MAX

address : int[1, m]

IP[j] := address,
x := 0

Reset[j]?
IP[j] := 0,
x := 0

UseIP[j] := false

UseIP[j] := true

counter < ANNOUNCE_NUM imply
x <= ANNOUNCE_INTERVAL

Frits Vaandrager  Model-Based Verification and Validation
“Mistakes” in Standard

► It does not specify upper and lower bounds on time that may elapse between sending last ARP Probe and sending first ARP Announcement

► It does not specify whether a host may immediately start using a newly claimed address or whether it should first send out all ARP Announcements

► It does not specify tolerance on timing of ARP Announcements

► Although standard states that Zeroconf requires an underlying network that supports ARP (RFC 826), we identified some cases where Zeroconf does not conform to RFC 826

► It is not exactly clear in which situations a host may defend its address
Cost-Optimization of Zeroconf (DSN 2003)

In Zeroconf there is a trade off between the time needed to acquire an address and the probability of address collision.

To study this trade off, a family of simple discrete time Markov reward models was defined.

Optimal configuration parameters of the network were derived.

We showed that usually it suffices to send only two probes.
Conclusions

1. Embedded system design is becoming mature engineering discipline
2. MDD offers great opportunity to improve V&V
3. Formal methods still far from mainstream but increasingly important in niche areas
4. General cost/benefit analysis difficult
5. Challenge is to recognize situations where formal methods are cost-effective; this requires expertise
6. ... available at universities, accessible via LaQuSo, ESI,..
7. Pentium bug style disaster needed before Dutch companies set up full-fledged formal V&V groups