Model-based Testing and Verification of Real-Time Systems

Brian Nielsen

bnielsen@cs.aau.dk
Research Profile

Distributed & Embedded Systems

Concurrency Theory
Foundation for system behavior

Verification and Validation
Tools for model checking

Networks and Operating Systems
Implementation and construction of platforms

Embedded Systems Methodology
Methods for specification, design, analysis, testing …
Industrial applications
Why CISS?

- 80% of all software is embedded
- Demands for increased functionality with minimal resources
- Requires multitude of skills
  - Software construction
  - Hardware platforms
  - Control theory
  - Comm. technology
- **Goal:**
  Give a qualitative lift to current industrial practice !!!!
CISS Structure

I KT Virksomheder

Institut for Datalogi

BRI CS@Aalborg
Modelling and Validation;
Programming Languages;
Software Engineering

Distributed Real Time Systems
Control Theory;
Real Time Systems;
Networking.

Embedded Systems
Communication;
HW/SW
Power Management

MVTU
25.5 MDKK

Nordjylland Amt Aalborg Kommune
12 MDKK

ES Oldenborg
ES Holland
ARTIS

AAU
12.75 MDKK

Virksomheder
12.75 MDKK
Partners

- Aeromark
- Analog Devices
- Blip Systems
- Danfoss
- Ericsson Telebit
- ETI
- Exhausto
- FOSS
- GateHouse
- Grundfos
- IAR Systems
- MAN B&W
- Novo Nordisk
- Motorola
- Panasonic
- RTX Telecom
- S-Card
- Simrad
- Skov
- SpaceCom
- TK Systemtest
- TDC Totalløsninger
- Aalborg Industries
Focus Areas

Applications

Home automation
Mobile robotter
Intelligente sensorer
Ad hoc netværk
Mobiltlf
Audio/Video
Konsum elektr
Kontrolsystemer
Automobile
X-by wire

Methods

SW-udvikling
Resource (Power) Management
Reliability
Test & Validering
Hybride systemer
Kommunikationsteori
Algoritrikk
Local ➔ Regional ➔ National

DaNES

- Danish Network for Intelligent Embedded Systems
- **PARTNERS**
  - CISS, IMM, MCI, PAJ Systemteknik
  - GateHouse A/S
  - ICE Power
  - Skov A/S
  - Terma A/S
  - Novo Nordisk A/S
  - IO Technologies
- **Funded** by Højteknologifonden
- **Budget**
  - 63 MDKK / 4 years
Local → Regional → National → International

ARTIST2 Network of Excellence
Information Society Technologies

6.5M Euro, 32 partners

EU’s 7th Framework
ARTEMIS Research Platform
Centers of Excellence

Testing & Verification
CISS koordinator
Quantitative System Properties in Model-Driven-Design of Embedded Systems

Service requirements
- QoS
- Availability
- Fault tolerance

Environment assumptions
- Timing constraints
- Hybrid behavior
- Arrival rates

Communication bandwidth

Computation resources

Power consumption

Costs

Memory usage
Course Outline

1. Modeling
   1. Model Based development
   2. Modelling Embedded systems
   3. Introduction to timed automata (TA)
   4. Modelling and Verification using Uppaal

2. Real-Time Conformance
   1. Testing theory
   2. Real-time extensions of the ioco testing theory

3. Off-Line Testing
   1. Off-line generation using model checkers
   2. (optimal) quantitative test-sequences(based on Priced TA)
   3. Testing strategies using Timed Games

4. On-Line Testing
   1. Online real-time testing
   2. Uppaal-Tron
   3. Monitoring and Environment Emulation;
   4. Industrial Case study

5. Future Challenges
Complex Systems
A very complex system
The first Ariane 5 rocket was launched in June, 1996. It used software developed for the successful Ariane 4. The rocket carried two computers, providing a backup in case one computer failed during launch. Forty seconds into its maiden flight, the rocket veered off course and exploded. The rocket, along with $500 million worth of satellites, was destroyed.

Ariane 5 was a much more powerful rocket and generated forces that were larger than the computer could handle. Shortly after launch, it received an input value that was too large. The main and backup computers shut down, causing the rocket to veer off course.
Rotterdam Storm Surge Barrier
Spectacular software bugs
U.S.S. Yorktown, U.S. Navy

- In 1998, the USS Yorktown became the first ship to test the US Navy's Smart Ship program. The Navy planned to use off-the-shelf computers and software instead of expensive U.S.S. Yorktown, courtesy of U.S. Navy custom-made machines. A sailor mistakenly entered a zero for a data value on a computer. Within minutes, Yorktown was dead in the water. It was several hours before the ship could move again.

- When the sailor entered the mistaken number, the computer tried to divide by zero, which isn't possible. The software didn't check to see if the inputs were valid before computing and generated an invalid answer that was used by another computer. The error cascaded several computers and eventually shut down the ship's engines.
Spectacular software bugs
Moon or Missiles

- The United States established the Ballistic Missile Early Warning System (BMEWS) during the Cold War to detect a Soviet missile attack. On October 5, 1960 the BMEWS radar at Thule, Greenland detected something. Its computer control system decided the signal was made by hundreds of missiles.

- The radar had actually detected the Moon rising over the horizon. Unfortunately, the BMEWS computer had not been programmed to understand what the moon looked like as it rose in the eastern sky, so it interpreted the huge signal as Soviet missiles. Luckily for all of us, the mistake was realized in time.
Spectacular software bugs
Therac 25

- The Therac-25 radiation therapy machine was a medical device that used beams of electrons or photons to kill cancer cells. Between 1985-1987, at least six people got very sick after Therac-25 treatments. Four of them died. The manufacturer was confident that their software made it impossible for the machine to harm patients.

- The Therac-25 was withdrawn from use after it was determined that it could deliver fatal overdoses under certain conditions. The software would shut down the machine before delivering an overdose, but the error messages it displayed were so unhelpful that operators couldn’t tell what the error was, or how serious it was. In some cases, operators ignored the message completely.
INTEL Pentium II floating-point division
470 Mill US $

Baggage handling system, Denver
1.1 Mill US $/day for 9 months

Mars Pathfinder

.......
Ordinary Software Bugs

BMW 745i software Defect:

"On certain passenger vehicles, due to a software error, a desynchronization of the valvetronic motors for engine banks I and II may occur. If this occurs, the engine could stall. In those cases, the driver may not be able to restart the engine. Depending on the level of engine roughness, or stalling, as well as traffic conditions and the driver’s reactions, this could lead to a crash."

15000 recalled

•70-100 ECU’s in modern cars
•SW major part of development cost
Why T&V?

- Boom in Embedded systems
- Errors in (Embedded) software are extremely expensive

Michael Williams
Research Director, Ericsson, SE
**STOP: 0x0000000A (0x802aa502,0x00000002,0x00000000,0xF084001C)**

IRQL_NOT_LESS_OR_EQUAL

**Address fa84001c has base at fa840000 - i8042prt.SYS**

**CPUID: GenuineIntel 5.2.c irql:1f**

**SYSVER 0xF0000565**

<table>
<thead>
<tr>
<th>Dll Base</th>
<th>Date Stamp</th>
<th>Name</th>
<th>Dll Base</th>
<th>Date Stamp</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>80100000</td>
<td>2be154c9</td>
<td>ntoskrnl.exe</td>
<td>80400000</td>
<td>2bc153b0</td>
<td>hal.dll</td>
</tr>
<tr>
<td>80200000</td>
<td>2bd49628</td>
<td>ncr710.sys</td>
<td>8025c000</td>
<td>2bd49688</td>
<td>SCSIPORT.SYS</td>
</tr>
<tr>
<td>80267000</td>
<td>2bd49683</td>
<td>scsidisk.sys</td>
<td>802a6000</td>
<td>2bd496b9</td>
<td>Fastfat.sys</td>
</tr>
<tr>
<td>fa800000</td>
<td>2bd49666</td>
<td>Floppy.SYS</td>
<td>fa810000</td>
<td>2bd496db</td>
<td>Hpfs_Reo.SYS</td>
</tr>
<tr>
<td>fa820000</td>
<td>2bd49676</td>
<td>Null.SYS</td>
<td>fa830000</td>
<td>2bd4965a</td>
<td>Beep.SYS</td>
</tr>
<tr>
<td>fa840000</td>
<td>2bd4a0b0</td>
<td>i8042prt.SYS</td>
<td>fa850000</td>
<td>2bd5a020</td>
<td>SERMOUSE.SYS</td>
</tr>
<tr>
<td>fa860000</td>
<td>2bd4966f</td>
<td>kbdclass.SYS</td>
<td>fa870000</td>
<td>2bd49671</td>
<td>MOUCLS.A.SYS</td>
</tr>
<tr>
<td>fa880000</td>
<td>2bd9c0be</td>
<td>Videopr.SYS</td>
<td>fa890000</td>
<td>2bd49638</td>
<td>NCR77C22.SYS</td>
</tr>
<tr>
<td>fa8a0000</td>
<td>2bd4a4ce</td>
<td>Vga.SYS</td>
<td>fa8d0000</td>
<td>2bd496d0</td>
<td>Msfs.SYS</td>
</tr>
<tr>
<td>fa8c0000</td>
<td>2bd496c3</td>
<td>Npfs.SYS</td>
<td>fa8e0000</td>
<td>2bd496c9</td>
<td>Ntfs.SYS</td>
</tr>
<tr>
<td>fa940000</td>
<td>2bd496df</td>
<td>NDIS.SYS</td>
<td>fa930000</td>
<td>2bd49707</td>
<td>wdlan.sys</td>
</tr>
<tr>
<td>fa970000</td>
<td>2bd49712</td>
<td>LDI.SYS</td>
<td>fa950000</td>
<td>2bd5a7fb</td>
<td>nbf.sys</td>
</tr>
<tr>
<td>fa990000</td>
<td>2bd7210f</td>
<td>stream.sys</td>
<td>fa9b0000</td>
<td>2bd4975f</td>
<td>wbm.bor</td>
</tr>
<tr>
<td>fa9c0000</td>
<td>2bd5bf7</td>
<td>mcsm.sys</td>
<td>fa9d0000</td>
<td>2bd4971d</td>
<td>netbios.sys</td>
</tr>
<tr>
<td>fa9e0000</td>
<td>2bd49678</td>
<td>Parallel.sys</td>
<td>fa9f0000</td>
<td>2bd4969f</td>
<td>serial.SYS</td>
</tr>
<tr>
<td>faa0000</td>
<td>2bd49739</td>
<td>mup.sys</td>
<td>faa40000</td>
<td>2bd4971f</td>
<td>SMBTRSUP.SYS</td>
</tr>
<tr>
<td>faa1000</td>
<td>2bd6f2a2</td>
<td>srv.sys</td>
<td>faa50000</td>
<td>2bd4971a</td>
<td>afd.sys</td>
</tr>
<tr>
<td>faa6000</td>
<td>2bd6fd80</td>
<td>rdr.sys</td>
<td>faa80000</td>
<td>2bd49735</td>
<td>browser.sys</td>
</tr>
</tbody>
</table>

**Address dword dump Build [1381] - Name**

- i8042prt.SYS
- SCSIPORT.SYS
- ntoskrnl.exe
- ntoskrnl.exe
- ntoskrnl.exe
- ntoskrnl.exe
- ntoskrnl.exe
Why T&V?

- Boom in Embedded systems
- Errors in (Embedded) software are extremely expensive
- 30-40% of development time spent on (often ad-hoc) testing.
- There is a enormous potential for improved methods and tools.
- “Time-to-market” can be reduced through early verification and performance analysis
Testing vs. Verification
Verification and Test

Model

/* Wait for events */
void OS_Wait(void);

/* Operating system visualSTATE process. Mimics a OS process for a
visualSTATE system. In this implementation this is the main
interfacing to the visualSTATE basic API. */
void OS_VS_Process(void);

/* Define completion code variable. */
unsigned char cc;

void HandleError(unsigned char ccArg)
{
    printf("Error code %c detected, exiting application.\n", ccArg);
    exit(ccArg);
}

/* In d-241 we only use the OS_Wait call. It is used to simulate a
system. It purpose is to generate events. How this is done is up to
you. */
void OS_Wait(void)
{
    /* Ignore the parameters; just retrieve events from the keyboard
    and put them into the queue. When EVENT_UNDEFINED is read from the
    keyboard, return to the calling process. */
    SEM_EVENT_TYPE event;
    int num;
}

• Verification Code/Model wrt Spec
• Test System wrt Model/Spec
Test versus Verification

Airbus Control Panel

Beolink

2^n sequences of length n

Deadlock identified by VERIFICATION after sequence of 2000 msgs / < 1min.
A Self-Assessment Test
[Myers]

- “A program reads three integer values. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.”

- Write a set of test cases to test this program
## Triangles

**Equilateral Triangle**
- Three equal sides
- Three equal angles, always 60°

**Isosceles Triangle**
- Two equal sides
- Two equal angles

**Scalene Triangle**
- No equal sides
- No equal angles

### Acute Triangle
- All angles are less than 90°

### Right Triangle
- Has a right angle (90°)

### Obtuse Triangle
- Has an angle more than 90°
A Self-Assessment Test
[Myers]

Test cases for:

1. valid scalene triangle ?
2. valid equilateral triangle ?
3. valid isosceles triangle ?
4. 3 permutations of previous ?
5. side = 0 ?
6. negative side ?
7. one side is sum of others ?
8. 3 permutations of previous ?
9. one side larger than sum of others ?
10. 3 permutations of previous ?
11. all sides = 0 ?
12. non-integer input ?
13. wrong number of values ?
14. for each test case: is expected output specified ?
15. check behaviour after output was produced ?
Testing

Testing:
- to check the **quality** (functionality, reliability, performance, ...) of an (software) object
  - by performing experiments
  - in a controlled way

- In avg. 10-20 errors per 1000 LOC
- 30-50 % of development time and cost in embedded software

- To find errors
- To determine risk of release
Risk

- **Make best possible use of resources by identifying and prioritizing quality aspects and subsystems**
  - Higher risk ⇒ more testing
  - No risk ⇒ no testing

- **Risk** = chance of failure × damage

  - Use frequency
  - Chance of error being present
    - Complexity
    - New tools/techniques
    - Inexperienced developers
  - Cost of repair
  - Loss of market share
  - Legal claim
What is a Test?

Test Data → Software under Test → Output

Test Cases

Oracle → Correct result?
Model-based Approach
Suggested Solution?

Model based validation, verification and testing of software and hardware
Traditional Software Development

The Waterfall Model

- Problem Area
  - Analyse
  - Design
  - Coding
  - Testing

- Costly in time-to-market and money
- Errors are detected late or never
- Application of models as early as possible
Introducing, Detecting and Repairing Errors

Liggesmeyer 98
Introducing, Detecting and Repairing Errors

Liggesmeyer 98

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Conceptual Design</th>
<th>Programming</th>
<th>Design Test</th>
<th>System Test</th>
<th>Operation</th>
</tr>
</thead>
</table>

![Graph showing the lifecycle of software development with error detection and repair costs.](CISS)
Model-Driven Development

- Analysis
- Validation
- Design Model
- Specification
- Verification & Refusal

- Implementation
- Testing
- Monitoring

- Automatic Code generation
- Automatic Test generation
- Automatic Monitoring

- FORMAL METHODS
- UML

CISS
Tamagotchi

ALIVE

Passive

Feeding

Meal

Health := Health - 1

Light

Care

A

A

B

B

Snack

A

A

Meal

B

A

Health := Health - 1

A

Clean

A

A

Medicine

A

A

Discipline

A

A

Play

A

A

Tick

Health := Health - 1; Age := Age + 1

A

DEAD

Health = 0 or Age = 2.000
Real-time Systems

**Plant/Env**

*Continuous*

- Realtime Protocols
- Pump Control
- Air Bags
- Robots
- Cruise Control
- ABS
- CD Players
- Production Lines

**Controller Program**

*Discrete*

- E.g.: Realtime Protocols
- Pump Control
- Air Bags
- Robots
- Cruise Control
- ABS
- CD Players
- Production Lines

---

**Real Time System**

A system where correctness not only depends on the logical order of events but also on their **timing**!!
Real-time Modeling

Plant
Continuous

Controller Program
Discrete

sensors

actuators

Model of Environment (non-deterministic/User-supplied)

inputs
outputs

UPPAAL Model

Model of Tasks (user supplied/automatic?)

CISS
Real-time Model-checking

Plant
Continuous

Controller Program
Discrete

sensors

actuators

Model of Tasks (user supplied / automatic?)

Model of Environment (non-deterministic/User-supplied)

inputs

outputs

UPPAAL Model

SAT \( \phi \)??
Real-time Controller Synthesis

Plant
Continuous

Controller Program
Discrete

Synthesis of Tasks/Scheduler (automatic)

Model of Environment
(non-deterministic/User-supplied)

Partial UPPAAL Model

inputs

outputs

SAT $\phi$ !!
Real-time Model-Based Testing

Plant

Controller Program

Test generation (offline or online) wrt. Design Model

UPPAAL Model

CISS
Real-time Monitoring

Plant
Continuous

Controller Program
Discrete

Model of Environment
(non-deterministic/User-supplied)

Observed trace $\sigma \in M$?

inputs

outputs

CISS
UPPAAL Tools

- Uppaal Model-checker:
  - Efficient reachability analysis of network of timed automata

- TIGA: Timed games (reachability and safety)
- CORA: Cost Optimal reachability from priced TA
- TRON: Testing Real-time Online

www.uppaal.com

CISS
Modelling and Analysis

Software Model

Requirement

TOOL

Yes,
Prototypes
Executable Code
Test sequences

No!
Debugging Information

Tools: UPPAAL, visualSTATE,
ESTEREL, SPIN, Statemate, FormalCheck,
VeriSoft, Java Pathfinder,…
Home-Banking?

```java
int accountA, accountB; //Shared global variables
//Two concurrent bank costumers

Thread costumer1 () {
    int a,b; //local tmp copy
    a=accountA;
    b=accountB;
    a=a-10; b=b+10;
    accountA=a;
    accountB=b;
}

Thread costumer2 () {
    int a,b;
    a=accountA;
    b=accountB;
    a=a-20; b=b+20;
    accountA=a;
    accountB=b;
}
```

- Are the accounts in balance after the transactions?
A[1] (pc1.finished and pc2.finished) imply (accountA+accountB==200)?
int accountA, accountB;  //Shared global variables
Semaphore A,B;          //Protected by sem A,B
//Two concurrent bank costumers

Thread costumer1 () {
    int a,b;  //local tmp copy
    wait(A);
    wait(B);
    a=accountA;
    b=accountB;
    a=a-10; b=b+10;
    accountA=a;
    accountB=b;
    signal(A);
    signal(B);
}

Thread costumer2 () {
    int a,b;
    wait(B);
    wait(A);
    a=accountA;
    b=accountB;
    a=a-20; b=b+20;
    accountA=a;
    accountB=b;
    signal(B);
    signal(A);
}

Semaphore FSM Model

**Binary Semaphore**
- open
- closed
- wait?
- signal?

**Counting Semaphore**
- \( c := \text{init\_count} \)
- \( c > 0 \)
- \( c := c - 1 \)
- \( c := c + 1 \)
- signal?
Composition

IO Automater (2-vejs synkronisering)
Composition

IO Automater (2-vejs synkronisering)
Semaphore Solution?

1. Consistency? (Balance)
2. Race conditions?
3. Deadlock?

1. A[] (mc1.finished and mc2.finished) imply (accountA+accountB==200) ✓
2. E<> mc1.critical_section and mc2.critical_section ✓
3. A[] not (mc1.finished and mc2.finished) imply not deadlock ✗