Model-driven, Stress testing of Embedded Systems: A Search-Based Approach

University of Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust
Software Verification and Validation Lab (www.svv.lu)

March 21, 2013

Lionel Briand, IEEE Fellow
FNR PEARL Chair
Objectives

• Experiences with MBT of embedded and real-time systems

• Stress testing: Worst case scenarios

• Search metaheuristics: Scalability and practicality
  – Stochastic optimization: Algorithms and techniques which employ some degree of randomness to find optimal (or as optimal as possible) solutions to hard problems.

• Different communities that should work together: Search-Based Testing (SBT) and MBT

• Go through example projects, and draw lessons learned
Using Search Heuristics in MBT

**Objective Function**

- A small part of the search space is traversed
- Guidance to worst case scenarios in space
- Heuristics: Extensive empirical studies are required
Early Work

L. Briand, Y. Labiche, and M. Shousha, 2006
Schedulability Theory

• Real-time scheduling theory
  – Given priorities, execution time, periods (periodic task), minimum inter-arrival times (aperiodic task), …
  – Is a group of (a)periodic tasks schedulable?
  – Theory to determine schedulability
    • Independent periodic tasks: Rate Monotonic Algorithm (RMA)
    • Aperiodic or dependent tasks: Generalized Completion Time Theorem (GCTT).

• GCTT assumes
  – aperiodic tasks equivalent to periodic tasks
    • periods = minimum inter-arrival times
    • aperiodic tasks start at time zero
  • Execution times are estimates
A Search-based Solution

- Goal: Solution making no assumptions and finding near deadline misses as well
- Population-based metaheuristic: Genetic Algorithm
- To automate, based on the system task architecture (UML SPT, MARTE), the derivation of arrival times for task triggering events that maximize the chances of critical deadline misses.
Model as Input

**UML-MARTE Model**

- Estimated execution time, Minimum inter-arrival time, ...
- Task priorities ...

**GA**

- Chromosome
- Fitness evaluation

**Scheduler**

- Start times, Pre-emption
- Arrival/ seeding times
Objective Function

- Focus on one target task at a time
- Goal: Guide the search towards arrival times causing the greatest delays in the executions of the target task
- Properties:
  - Handle deadline misses
  - Consider all task executions, not just worst case execution
  - Reward task executions so that many good executions do not wind up overshadowing one bad execution
Objective Function II

\[ f(Ch) = \sum_{j=1}^{k_t} 2^{e_{t,j} - d_{t,j}} \]

- \( t \): target task
- \( k_t \): maximum number of executions of \( t \)
- \( e \): estimated end time of execution \( j \) of target task as determined by scheduler
- \( d \): deadline of execution \( j \) of target task
Case Study

- Software Engineering Institute (SEI), Naval Weapons Center and IBM’s Federal Sector Division
- Hard real-time, realistic avionics application model similar to existing U.S. Navy and Marine aircrafts
- Eight highest priority tasks deemed schedulable
- Our findings suggest three of eight tasks produce systematic deadline misses
## Results

<table>
<thead>
<tr>
<th>Task</th>
<th>Number of Misses</th>
<th>Value of Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weapon Release</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>Weapon Release Subtask</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>Radar Tracking Filter</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>RWR Contact Management</td>
<td>2</td>
<td>3, 9</td>
</tr>
<tr>
<td>Data Bus Poll Device</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>Weapon Aiming</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>Radar Target Update</td>
<td>4</td>
<td>17, 16, 10, 9</td>
</tr>
<tr>
<td>Navigation Update</td>
<td>7</td>
<td>1, 29, 23, 2, 28, 27, 32</td>
</tr>
</tbody>
</table>
Conclusions

• We devised a method to generate event seeding times for aperiodic tasks so as detecting deadline misses based on design information
• Near deadline misses as well! (stress testing)
• Standard modeling notation (UML/SPT/MARTE)
• No dedicated, additional modeling compared to what is expected when defining a task architecture
• Scalability: GA runs lasted a few minutes
• Default GA parameters, as recommended in literature, work well
• Large empirical studies to evaluate the approach (heuristics)

• Similar work with concurrency analysis: Deadlocks, data races, etc. (Shousha, Briand, Labiche, 2008-2012)
Using Constraint Solving

System: fire/gas detection and emergency shutdown

Drivers
(Software-Hardware Interface)

Alarm Devices
(Hardware)

Real Time Operating System

Multicore Archt.
Safety Drivers May Overload the CPU

• Drivers need to bridge the timing gaps between SW and HW
• SIL 3
• Drivers have flexible design
  • parallel threads communicating in an asynchronous way
• Drivers are subject to real-time constraints to make sure they do not overuse the CPU time, e.g., “The processor spare-time should not be less than 80% at any time”
Safety standards

IEC 61508 is a Safety Standard including guidelines for Performance Testing

To achieve SIL levels 3-4, Stress Testing is “Highly Recommended”
Small Delay \[\rightarrow\] T2 consumes a lot of CPU time \[\rightarrow\] CPU overload

Large Delay \[\rightarrow\] T2 may block T1 and/or T3 \[\rightarrow\] Dead line misses
General Approach: Modeling and Optimization

1. **System Design and Platform Model (UML/MARTE)**
   - **INPUT**

2. **Constraint Program**
   - **INPUT**: Performance Requirements (objective functions)
   - **OUTPUT**
     - Stress Test Cases (Delay values leading to worst case CPU time usage)

3. **Modeling**
   - **INPUT**

4. **Constraint Programming**
   - **CP Engine (COMET)**
Information Requirements

- **Scheduler**
  - preemptive : bool
  - min duration \((\text{min}_d)\) : int
  - max duration \((\text{max}_d)\) : int
  - delay : int

- **Processing Unit**
  - number of cores : int

- **Global Clock**
  - time : int

- **Thread**
  - priority : int
  - period \((p)\) : int
  - min inter-arrival time \((\text{min}_\text{ia})\) : int
  - max inter-arrival time \((\text{max}_\text{ia})\) : int
  - Start()
  - Finish()
  - Wait()
  - Sleep()
  - Resume()
  - Trigger()

- **Data dependency \((\leq_d)\)**

- **Synch**

- **Asynch**

- **Buffer**
  - size : int
  - access()
Some abstractions are design choices: delays, priorities…

Some others depend on the environment: arrival times…
Platform and Design Properties modeled in UML are provided as input in our Constraint Program

COMET input language

Design properties include: threads, priorities, activities, durations...

Preemptions at regular time periods (quanta)

Assume negligible context switching time compared to time quantum

Platform and design properties are constants in our Constraint Program

// 1) Input: Time and Concurrency information
int c = ...; // #Cores
int n = ...; range J = 0..n-1; // #Threads
int priority[J] = ...; // Priorities
// ...

// 2) Output: Scheduling variables
dvar int arrival_time[a in A] in T;
   // Actual arrival times
dvar int start[a in A] in est[a]..lst[a];
   // Actual start times
dvar int end[a in A] in eet[a]..let[a];
   // Actual end times
// ...

// 3) Objective function: Performance Requirement
maximize sum(a in A)(maxl(0, minl(1,
   deadline_miss[a]))); // Deadline misses function
// 4) Constraints: Scheduling policy
subject to {
   forall(a in A) {
      wf4: start[a] <= end[a]; // Threads should end after their start time
   }
// ...
Threads Properties which can be tuned during testing are the output of our Constraint Program.

Tunable Parameters may include design and real time properties.

Tunable Parameters are variables in our Constraint Program.

Some tunable parameters are the basis for the definition to stress test cases, others are results from scheduling.

---

// 1) Input: Time and Concurrency information
int c = ...; // #Cores
int n = ...; range J = 0..n-1; // #Threads
int priority[J] = ...; // Priorities
// ...

// 2) Output: Scheduling variables
dvar int arrival_time[a in A] in T;
  // Actual arrival times
dvar int start[a in A] in est[a]..lst[a];
  // Actual start times
dvar int end[a in A] in eet[a]..let[a];
  // Actual end times
// ...

// 3) Objective function: Performance Requirement
maximize
  sum(a in A) (maxl(0, minl(1, deadline_miss[a]))); // Deadline misses function

// 4) Constraints: Scheduling policy
subject to {
  forall(a in A) {
    wf4: start[a] <= end[a]; // Threads should end after their start time
  }
}
The Performance Requirement is modeled as an objective function to maximize

We focused on objective functions for CPU Usage

Each objective function models a specific performance requirement

Testing a different performance requirements only requires to change the objective function (constraints)
The Platform scheduler and properties are modeled through a set of constraints

Constraints express relationships between constants and variables

Constraints are independent and can be modified to fit different platforms, for example scheduling algorithm

// 1) Input: Time and Concurrency information
int c = ...; // #Cores
int n = ...; range J = 0..n-1; // #Threads
int priority[J] = ...; // Priorities
// ...

// 2) Output: Scheduling variables
dvar int arrival_time[a in A] in T;
    // Actual arrival times
dvar int start[a in A] in est[a]..lst[a];
    // Actual start times
dvar int end[a in A] in eet[a]..let[a];
    // Actual end times
// ...

// 3) Objective function: Performance Requirement
maximize
    sum(a in A)(maxl(0, minl(1,
        deadline_miss[a]))); // Deadline misses function

// 4) Constraints: Scheduling policy
subject to {
    forall(a in A) {
        wf4: start[a] <= end[a]; // Threads should end after their start time
        // ...
    }
}
We run an experiment with real data and two different objective functions. But optimal solutions were found shortly after the search started, even if the search took a much more time to terminate. It took a significant amount of time for the search to terminate.
Conclusions

• We re-express test case generation for CPU usage requirements as a constraint optimization problem

• Approach:
  – A conceptual model for time abstractions
  – Mapping to MARTE
  – A constraint optimization formulation of the problem
  – Application of the approach to a real case study (albeit small)

• Using a constraint solver does not seem to scale to large numbers of threads
Testing Driven by Environment Modeling

Z. Iqbal, A. Arcuri, L. Briand, 2012
Context

- Three-year project with two industry partners
  - Soft-real time systems: deadlines in order of hundreds of milliseconds
    - Jitter of few milliseconds acceptable
  - Automation of test case and oracle generation, environment simulation

*Tomra – Bottle Recycling Machine*  
*WesternGeco – Marine Seismic Acquisition System*
Environment Modeling and Simulation

- Independent
  - Black-box
- Behavior driven by environment
  - Environment model
- Software engineers
- No use of Matlab/Simulink
- One model for
  - Environment simulator
  - Test cases and oracles
- UML profile (+ limited use of MARTE)
Domain Model
Behavior Model
Test Cases

• Test cases are defined by
  – Simulation configuration
  – Environment configuration
• Environment Configuration
  – Number of instances to be created for each component in the domain model (e.g., the number of sensors)
• Simulator Configuration
  – Setting of non-deterministic attribute values
• Test oracle: Environment model error states
  – A successful test case is one which leads the environment into an error state
• Bring the system state to an error state by search for appropriate values for non-deterministic environment attributes

• Search heuristics are based on fitness functions assessing how “close” is the current state to an error state

• Defining the fitness function based on model information was tricky

• Used and compared different metaheuristics

• Industrial case study and artificial examples showed the heuristic was effective
Current Work: Testing Closed Loop Controllers

Complexity and amount of software used on vehicles’ Electronic Control Units (ECUs) grow rapidly

**Comfort and variety**

*More functions*  
*Safety and reliability*

**Faster time-to-market**

*Greenhouse gas emission laws*  
*Less fuel consumption*
Three major software development stages in the automotive domain
Major Challenges in MiL-SiL-HiL Testing

- Manual test case generation
- Complex functions at MiL, and large and integrated software/embedded systems at HiL
- Lack of precise requirements and testing Objectives
- Hard to interpret the testing results
The ultimate goal of MiL testing is to ensure that individual functions behave correctly and timely on any hardware configuration.
Main differences between automotive function testing and general software testing

- Continuous behaviour
- Time matters a lot
- Several configurations
  - Huge number of detailed physical measures/ranges/thresholds captured by calibration values
A Taxonomy of Automotive Functions

Different testing strategies are required for different types of functions
Controller Plant Model at MiL-Level and its Requirements

Desired value \[ \pm \] Error \[ \rightarrow \] Controller (SUT) \[ \rightarrow \] Plant Model \[ \rightarrow \] System output

Actual value

Desired Value & Actual Value

(a) Liveness \[ x \]

(b) Smoothness \[ v \]

(c) Responsiveness \[ y \]

Desired Value & Actual Value

Desired Value \[ \rightarrow \] Actual Value

Desired Value \[ \rightarrow \] Actual Value

Desired Value \[ \rightarrow \] Actual Value

Desired Value \[ \rightarrow \] Actual Value

Desired Value \[ \rightarrow \] Actual Value
Types of Requirements

(1) functional/liveness
SBPC function shall guarantee that the flap will move to and will stabilize at its desired position within \( xx \) ms. Further, the flap shall reach within \( yy \)% of its desired position within \( zz \) ms. In addition, after reaching \( vv \)% close to the desired position, the flap shall not jump to a position more than \( ww \)% away from its desired position.

(2) responsiveness/performance

(3) smoothness/safety
Search Elements

• **Inputs of the search:**
  - Initial and desired values, configuration parameters
  - The same as the input to test cases

• **Search Objective:**
  - Example requirement that we want to test: liveness
    ✓ |Desired - Actual(final)|\(\sim\) 0

For each set of inputs, we evaluate the objective function over the resulting simulation graphs:

• **Result:**
  - worst case scenarios or values to the input variables that are more likely to break the requirement at MiL level
  - stress test cases based on actual hardware (HiL)
MiL-Testing of Continuous Controllers

- Objective Functions + Controller-plant model
- Exploration
- Overview Diagram
- Domain Expert
- List of Regions
- Local Search
- Test Scenarios

Graph Builder

Final vs. Initial

Smoothness

Desired Value

Actual Value

Initial Desired

Final Desired

Graph Builder

Initial Desired

Final Desired
Generated Heatmap Diagrams

(a) Liveness

(b) Smoothness

(c) Responsiveness
Random Search vs. (1+1)EA Search in Regions
Example Responsiveness Analysis
Conclusions

• Models
  – Great variety
  – UML, UML profiles (MARTE), Matlab/Simulink
  – Information that is needed to guide the search
  – Test objectives
  – Current modeling practice
  – Reliance on standards

• Search
  – Metaheuristic search: Versatile techniques
  – Most stress testing problems can be re-expressed as a search problem
  – Lower modeling requirements, but no guarantee
  – Many search techniques
  – It is not easy to choose which one to use for a problem
  – Empirical studies
  – Promising results, scalability (incomplete search)
Selected References

- More information: www.svv.lu