Making Model-Driven Verification Practical and Scalable - Experiences and Lessons Learned

Lionel Briand

Interdisciplinary Centre for ICT Security, Reliability, and Trust (SnT)
University of Luxembourg, Luxembourg

MODELSWARD, Rome, February 20, 2016
Acknowledgements

PhD. Students:
- Vahid Garousi
- Marwa Shousha
- Zohaib Iqbal
- Reza Matinnejad
- Stefano Di Alesio
- Raja Ben Abdessalem

Others:
- Shiva Nejati
- Andrea Arcuri
- Yvan Labiche
The term “verification” is used in its wider sense: Defect detection.

Testing is, in practice, the most common verification technique.

Testing is about systematically, and preferably automatically, exercise a system such as to maximize chances of uncovering (important) latent faults within time constraints.

Other forms of verifications are important too (e.g., design time, run-time), but much less present in practice.

Decades of research have not yet significantly and widely impacted engineering practice.
Cyber-Physical Systems: Challenges

- Increasingly complex and critical systems
- Complex environment
- Hybrid discrete and continuous behavior
- Combinatorial and state explosion
- Complex requirements, e.g., temporal, timing, resource usage
- Uncertainty, e.g., about the environment
Scalable? Practical?

- **Scalable**: Can a technique be applied on large artifacts (e.g., models, data sets, input spaces) and still provide useful support within reasonable effort, CPU and memory resources?

- **Practical**: Can a technique be efficiently and effectively applied by engineers in realistic conditions?
  - realistic ≠ universal
Focus

• **Formal Verification (Wikipedia):** In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.

• **Our focus:** How can we, in a practical, effective and efficient manner, uncover as many (critical) faults as possible in software systems, within time constraints, while scaling to artifacts of realistic size.
Metaheuristics

- **Heuristic search (Metaheuristics):** Hill climbing, Tabu search, Simulated Annealing, Genetic algorithms, Ant colony optimisation ….

- **Stochastic optimization:** General class of algorithms and techniques which employ some degree of randomness to find optimal (or as optimal as possible) solutions to hard problems

- Many verification and testing problems can be re-expressed as (hard) optimization problems
Talk Outline

• Selected project examples, with industry collaborations

• Similarities and patterns

• Lessons learned
Testing Software Controllers

References:

• R. Matinnejad et al., “Effective Test Suites for Mixed Discrete-Continuous Stateflow Controllers”, ACM ESEC/FSE 2015 (Distinguished paper award)
• R. Matinnejad et al., “MiL Testing of Highly Configurable Continuous Controllers: Scalable Search Using Surrogate Models”, IEEE/ACM ASE 2014 (Distinguished paper award)
Dynamic Continuous Controllers
Electronic Control Units (ECUs)

Comfort and variety

More functions

Safety and reliability

Faster time-to-market

Less fuel consumption

Greenhouse gas emission laws
A Taxonomy of Automotive Functions

Different testing strategies are required for different types of functions
Development Process

Model-in-the-Loop Stage
- Simulink Modeling
- MiL Testing

Software-in-the-Loop Stage
- Code Generation and Integration
- SiL Testing

Hardware-in-the-Loop Stage
- Software Running on ECU
- HiL Testing

Generic Functional Model
MATLAB/Simulink model

Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, …
Controller Input and Output at MIL

Diagram:
- Desired value → Error
- Controller (SUT) → Plant Model → System output
- Actual value

Test Input
- Initial Desired Value
- Final Desired Value
- Time: $T/2$, $T$

Test Output
- Desired Value
- Actual Value
- Time: $T/2$, $T$
controllers at mil

inputs: time-dependent variables

plant model

configuration parameters
Requirements and Test Objectives

Initial Desired (ID)
Desired Value (input)
Actual Value (output)

Final Desired (FD)

Responsiveness
Smoothness
Stability

T/2
T

time
Test Strategy: A Search-Based Approach

- Continuous behavior
- Controller’s behavior can be complex
- Meta-heuristic search in (large) input space: Finding worst case inputs
- Possible because of automated oracle (feedback loop)
- Different worst cases for different requirements
- Worst cases may or may not violate requirements

Worst Case(s)?

Initial Desired (ID)

Final Desired (FD)
Smoothness Objective Functions: $O_{\text{Smoothness}}$

We want to find test scenarios which maximize $O_{\text{Smoothness}}$.
Search Elements

- **Search Space:**
  - Initial and desired values, configuration parameters

- **Search Technique:**
  - (1+1) EA, variants of hill climbing, GAs …

- **Search Objective:**
  - Objective/fitness function for each requirement

- **Evaluation of Solutions**
  - Simulation of Simulink model => fitness computation

- **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)
Solution Overview (Simplified Version)

1. Exploration
- List of Critical Regions
  - Domain Expert
  - Worst-Case Scenarios

Graph Builder

HeatMap Diagram

Objective Functions based on Requirements + Controller-plant model

Graph: Final vs. Initial

Smoothness:
- 0.100
- 0.150
- 0.200
- 0.250
- 0.300

Desired Value
- Initial Desired
- Final Desired

Actual Value
Automotive Example

- **Supercharger bypass flap controller**
  - Flap position is bounded within [0..1]
  - Implemented in MATLAB/Simulink
  - 34 sub-components decomposed into 6 abstraction levels
  - The simulation time $T=2$ seconds

Flap position = 0 (open)  Flap position = 1 (closed)
Finding Seeded Faults

Inject Fault
Analysis – Fitness increase over iterations
Analysis II – Search over different regions

Average

(1+1) EA Distribution

Random Search Distribution

Number of Iterations
Conclusions

- We found much worse scenarios during MiL testing than our partner had found so far, and much worse than random search (baseline).
- These scenarios are also run at the HiL level, where testing is much more expensive: MiL results -> test selection for HiL.
- But further research was needed:
  - Simulations are expensive
  - Configuration parameters
  - Dynamically adjust search algorithms in different subregions (exploratory <-> exploitative)
Testing in the Configuration Space

- MIL testing for all feasible configurations
- The search space is much larger
- The search is much slower (Simulations of Simulink models are expensive)
- Results are harder to visualize
- Not all configuration parameters matter for all objective functions
Modified Process and Technology

**Objective Functions + Controller Model (Simulink)**

1. Exploration with Dimensionality Reduction
   - Regression Tree
   - Domain Expert
   - List of Critical Partitions

2. Search with Surrogate Modeling
   - Visualization of the 8-dimension space using regression trees
   - Surrogate modeling to predict the objective function and speed up the search (Machine learning)

Dimensionality reduction to identify the significant variables (Elementary Effect Analysis)

Worst-Case Scenarios
Dimensionality Reduction

- Sensitivity Analysis: Elementary Effect Analysis (EEA)
- Identify non-influential inputs in computationally costly mathematical models
- Requires less data points than other techniques
- Observations are simulations generated during the Exploration step
- Compute sample mean and standard deviation for each dimension of the distribution of elementary effects
Imagine function $F$ with 2 inputs, $x$ and $y$:

Elementary Effects Analysis Method

- **Elementary Effects for $X$**
  - $F(A_1) - F(A)$
  - $F(B_1) - F(B)$
  - $F(C_1) - F(C)$
  - ...

- **Elementary Effects for $Y$**
  - $F(A_2) - F(A)$
  - $F(B_2) - F(B)$
  - $F(C_2) - F(C)$
  - ...

Diagram showing the changes in $x$ and $y$ for each elementary effect.
Visualization in Inputs & Configuration Space

All Points
- Count: 1000
- Mean: 0.007822
- Std Dev: 0.0049497

FD>=0.43306
- Count: 574
- Mean: 0.0059513
- Std Dev: 0.0040003

FD<0.43306
- Count: 426
- Mean: 0.0103425
- Std Dev: 0.0049919

ID<0.64679
- Count: 373
- Mean: 0.0047594
- Std Dev: 0.0034346

ID>=0.64679
- Count: 201
- Mean: 0.0081631
- Std Dev: 0.0040422

Cal5>=0.020847
- Count: 244
- Mean: 0.0080206
- Std Dev: 0.0031751

Cal5<0.020847
- Count: 182
- Mean: 0.0134555
- Std Dev: 0.0052883

Regression Tree
Surrogate Modeling (1)

- Goal: To predict the value of the objective functions within a critical partition, given a number of observations, and use that to avoid as many simulations as possible and speed up the search.
Surrogate Modeling (2)

- Any supervised learning or statistical technique providing fitness predictions with confidence intervals

1. Predict higher fitness with high confidence: Move to new position, no simulation
2. Predict lower fitness with high confidence: Do not move to new position, no simulation
3. Low confidence in prediction: Simulation
Experiments Results (RQ1)

- The best regression technique to build Surrogate models for all of our three objective functions is Polynomial Regression with \( n = 3 \)
- Other supervised learning techniques, such as SVM

Mean of \( R^2/\text{MRPE} \) values for different surrogate modeling techniques

<table>
<thead>
<tr>
<th></th>
<th>LR</th>
<th>ER</th>
<th>PR(n=2)</th>
<th>PR(n=3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_{sm} )</td>
<td>( 0.66/0.0526 )</td>
<td>( 0.44/0.0791 )</td>
<td>( 0.95/0.0203 )</td>
<td>( 0.98/0.0129 )</td>
</tr>
<tr>
<td>( F_r )</td>
<td>( 0.78/0.0295 )</td>
<td>( 0.49/1.2281 )</td>
<td>( 0.85/0.0247 )</td>
<td>( 0.85/0.0245 )</td>
</tr>
<tr>
<td>( F_{st} )</td>
<td>( 0.26/0.2043 )</td>
<td>( 0.22/1.2519 )</td>
<td>( 0.46/0.1755 )</td>
<td>( 0.54/0.1671 )</td>
</tr>
</tbody>
</table>
Experiments Results (RQ2)

✓ Dimensionality reduction helps generate better surrogate models for Smoothness and Responsiveness requirements

Mean Relative Prediction Errors (MRPE Values)

Smoothness($F_{sm}$)  Responsiveness($F_r$)  Stability($F_{st}$)
Experiments Results (RQ3)

✓ For responsiveness, the search with SM was 8 times faster

✓ For smoothness, the search with SM was much more effective
Our approach is able to identify critical violations of the controller requirements that had neither been found by our earlier work nor by manual testing.

<table>
<thead>
<tr>
<th></th>
<th>MiL-Testing different configurations</th>
<th>MiL-Testing fixed configurations</th>
<th>Manual MiL-Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability</td>
<td>2.2% deviation</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Smoothness</td>
<td>24% over/undershoot</td>
<td>20% over/undershoot</td>
<td>5% over/undershoot</td>
</tr>
<tr>
<td>Responsiveness</td>
<td>170 ms response time</td>
<td>80 ms response time</td>
<td>50 ms response time</td>
</tr>
</tbody>
</table>
A Taxonomy of Automotive Functions

Different testing strategies are required for different types of functions.
Differences with Close-Loop Controllers

- Mixed discrete-continuous behavior: Simulink stateflows
- Much quicker simulation time
- No feedback loop -> no automated oracle
- The main testing cost is the manual analysis of output signals
- Goal: Minimize test suites
- Challenge: Test selection
- Entirely different approach to testing
Selection Strategies Based on Search

- Input diversity
- White-box Structural Coverage
  - State Coverage
  - Transition Coverage
- Output Diversity
- Failure-Based Selection Criteria
  - Domain specific failure patterns
  - Output Stability
  - Output Continuity
Failure-based Test Generation

- Maximizing the likelihood of presence of specific failure patterns in output signals
- Failure patterns elicited from engineers

**Instability**

**Discontinuity**
Summary of Results

- The test cases resulting from state/transition coverage algorithms cover the faulty parts of the models.
- However, they fail to generate output signals that are sufficiently distinct from the oracle signal, hence yielding a low fault revealing rate.
- Output-based algorithms are more effective.
Automated Testing of Vision Systems Through Simulation

- With Raja Ben Abdessalem, Shiva Nejati
- In collaboration with IEE, Luxembourg
Night Vision (NiVi) System

- The NiVi system is a camera-based assistance system providing improved vision at night
Testing Vision Systems

• Testing vision systems requires complex and comprehensive simulation environments
  – Static objects: roads, weather, etc.
  – Dynamic objects: cars, humans, animals, etc.

• A simulation environment captures the behaviour of dynamic objects as well as constraints and relationships between dynamic and static objects
Overview

Specification Documents
(Simulation Environment and NiVi System)

(1) Development of Requirements and domain models

Domain model ➔ Requirements model

(2) Generation of Test specifications

test case specification

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ranges/values/</td>
<td>[ranges/</td>
</tr>
<tr>
<td>resolution]</td>
<td>resolution]</td>
</tr>
</tbody>
</table>

Traceability
NiVi and Environment Domain Model

- **Object**
  - **Infrastructure**
    - Animated Element
      - Road
        - Buildings
          - Abstract Buildings
            - TrafficSign
  - Environment
    - Underlays
      - Dirt Spot
    - Nature Element
  - Actor
    - Human
    - Animal
      - Car/Motor/Truck/Bus (+NiVi)
    - Speed Profile
    - Trajectory
      - Path
        - Slot
        - Path Segment
      - 1
        - 1..*
        - 1
The NiVi system shall detect any person located in the Acute Warning Area of a vehicle.

\[ \exists t. \text{car.awa.pos}_{x1}[t] < \text{car.awa.humantrajectory.pos}_{x}[t] < \text{car.awa.pos}_{x2}[t] \land \]
\[ \text{car.awa.pos}_{y1}[t] < \text{car.awa.humantrajectory.pos}_{y}[t] < \text{car.awa.pos}_{y2}[t] \]
\[ \Rightarrow \text{car.sensor.warning} == \text{true} \]
MiL Testing via Search

Meta-heuristic Search (multi-objective)

 Simulator + NiVi

Fixed during Search
Environment Settings (Roads, weather, vehicle type, etc.)

Manipulated by Search

Car Simulator (speed)
NiVi

Human Simulator (initial position, speed, orientation)

Generate scenarios

Detection or not?
Collision or not?
<table>
<thead>
<tr>
<th>Situation</th>
<th>Type of Road</th>
<th>Type of vehicle</th>
<th>Type of actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Situation 1</td>
<td>Straight</td>
<td>Car</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 2</td>
<td>Straight</td>
<td>Car</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 3</td>
<td>Straight</td>
<td>Car</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 4</td>
<td>Straight</td>
<td>Truck</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 5</td>
<td>Straight</td>
<td>Truck</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 6</td>
<td>Straight</td>
<td>Truck</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 7</td>
<td>Curved</td>
<td>Car</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 8</td>
<td>Curved</td>
<td>Car</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 9</td>
<td>Curved</td>
<td>Car</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 10</td>
<td>Curved</td>
<td>Truck</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 11</td>
<td>Curved</td>
<td>Truck</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 12</td>
<td>Curved</td>
<td>Track</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 13</td>
<td>Ramp</td>
<td>Car</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 14</td>
<td>Ramp</td>
<td>Car</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 15</td>
<td>Ramp</td>
<td>Car</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 16</td>
<td>Ramp</td>
<td>Truck</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 17</td>
<td>Ramp</td>
<td>Truck</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 18</td>
<td>Ramp</td>
<td>Truck</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 19</td>
<td>Straight</td>
<td>Car+ Cars in parking</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 20</td>
<td>Straight</td>
<td>Car + buildings</td>
<td></td>
</tr>
</tbody>
</table>
Test Case Specification: Dynamic

<table>
<thead>
<tr>
<th>car : Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>PositionX = 10</td>
</tr>
<tr>
<td>Position Y = 50.125</td>
</tr>
<tr>
<td>Position Z = 0.56</td>
</tr>
<tr>
<td>OrientationHeading = 0</td>
</tr>
<tr>
<td>Acceleration = 0</td>
</tr>
<tr>
<td>MaxWalkingSpeed = 100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>trajectoryCar : Trajectory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start locationX = 10</td>
</tr>
<tr>
<td>Start locationY = 50.125</td>
</tr>
<tr>
<td>Start locationZ = 0.56</td>
</tr>
<tr>
<td>Orientation = 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>profilePerson : Speed Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueld</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segmentCar : Path Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length = 100</td>
</tr>
<tr>
<td>Type = Straight</td>
</tr>
<tr>
<td>MaxSpeedLimit = 100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segmentPerson : Path Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length = 60</td>
</tr>
<tr>
<td>Type = Straight</td>
</tr>
<tr>
<td>MaxSpeedLimit = 14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slotCar : Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>startCar : StartState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time = 0</td>
</tr>
<tr>
<td>Speed = 60.66</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pathCar : Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPointX = 10</td>
</tr>
<tr>
<td>StartPointY = 50.125</td>
</tr>
<tr>
<td>StartPointZ = 0.56</td>
</tr>
<tr>
<td>StartAngle = 0</td>
</tr>
<tr>
<td>End Angle = 0</td>
</tr>
<tr>
<td>Length = 100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>profileCar : Speed Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueld</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slotPerson : Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>startPerson : StartState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time = 0</td>
</tr>
<tr>
<td>Speed = 12.59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pathPerson : Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPointX = 74</td>
</tr>
<tr>
<td>StartPointY = 37.72</td>
</tr>
<tr>
<td>StartPointZ = 0</td>
</tr>
<tr>
<td>StartAngle = 93.33</td>
</tr>
<tr>
<td>End Angle = 0</td>
</tr>
<tr>
<td>Length = 60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>profilePerson : Speed Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueld</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segmentPerson : Path Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length = 60</td>
</tr>
<tr>
<td>Type = Straight</td>
</tr>
<tr>
<td>MaxSpeedLimit = 14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slotPerson : Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>startPerson : StartState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time = 0</td>
</tr>
<tr>
<td>Speed = 60.66</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pathPerson : Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPointX = 74</td>
</tr>
<tr>
<td>StartPointY = 37.72</td>
</tr>
<tr>
<td>StartPointZ = 0</td>
</tr>
<tr>
<td>StartAngle = 93.33</td>
</tr>
<tr>
<td>End Angle = 0</td>
</tr>
<tr>
<td>Length = 60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>profilePerson : Speed Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueld</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segmentPerson : Path Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length = 60</td>
</tr>
<tr>
<td>Type = Straight</td>
</tr>
<tr>
<td>MaxSpeedLimit = 14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slotPerson : Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>startPerson : StartState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time = 0</td>
</tr>
<tr>
<td>Speed = 12.59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pathPerson : Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPointX = 74</td>
</tr>
<tr>
<td>StartPointY = 37.72</td>
</tr>
<tr>
<td>StartPointZ = 0</td>
</tr>
<tr>
<td>StartAngle = 93.33</td>
</tr>
<tr>
<td>End Angle = 0</td>
</tr>
<tr>
<td>Length = 60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>profilePerson : Speed Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueld</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segmentPerson : Path Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length = 60</td>
</tr>
<tr>
<td>Type = Straight</td>
</tr>
<tr>
<td>MaxSpeedLimit = 14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slotPerson : Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>startPerson : StartState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time = 0</td>
</tr>
<tr>
<td>Speed = 12.59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pathPerson : Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPointX = 74</td>
</tr>
<tr>
<td>StartPointY = 37.72</td>
</tr>
<tr>
<td>StartPointZ = 0</td>
</tr>
<tr>
<td>StartAngle = 93.33</td>
</tr>
<tr>
<td>End Angle = 0</td>
</tr>
<tr>
<td>Length = 60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>profilePerson : Speed Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueld</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segmentPerson : Path Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length = 60</td>
</tr>
<tr>
<td>Type = Straight</td>
</tr>
<tr>
<td>MaxSpeedLimit = 14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slotPerson : Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>startPerson : StartState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time = 0</td>
</tr>
<tr>
<td>Speed = 12.59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pathPerson : Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPointX = 74</td>
</tr>
<tr>
<td>StartPointY = 37.72</td>
</tr>
<tr>
<td>StartPointZ = 0</td>
</tr>
<tr>
<td>StartAngle = 93.33</td>
</tr>
<tr>
<td>End Angle = 0</td>
</tr>
<tr>
<td>Length = 60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>profilePerson : Speed Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueld</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segmentPerson : Path Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length = 60</td>
</tr>
<tr>
<td>Type = Straight</td>
</tr>
<tr>
<td>MaxSpeedLimit = 14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slotPerson : Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>startPerson : StartState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time = 0</td>
</tr>
<tr>
<td>Speed = 12.59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pathPerson : Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPointX = 74</td>
</tr>
<tr>
<td>StartPointY = 37.72</td>
</tr>
<tr>
<td>StartPointZ = 0</td>
</tr>
<tr>
<td>StartAngle = 93.33</td>
</tr>
<tr>
<td>End Angle = 0</td>
</tr>
<tr>
<td>Length = 60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>profilePerson : Speed Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueld</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segmentPerson : Path Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length = 60</td>
</tr>
<tr>
<td>Type = Straight</td>
</tr>
<tr>
<td>MaxSpeedLimit = 14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slotPerson : Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>startPerson : StartState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time = 0</td>
</tr>
<tr>
<td>Speed = 12.59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pathPerson : Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartPointX = 74</td>
</tr>
<tr>
<td>StartPointY = 37.72</td>
</tr>
<tr>
<td>StartPointZ = 0</td>
</tr>
<tr>
<td>StartAngle = 93.33</td>
</tr>
<tr>
<td>End Angle = 0</td>
</tr>
<tr>
<td>Length = 60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>profilePerson : Speed Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueld</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segmentPerson : Path Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length = 60</td>
</tr>
<tr>
<td>Type = Straight</td>
</tr>
<tr>
<td>MaxSpeedLimit = 14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slotPerson : Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>startPerson : StartState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time = 0</td>
</tr>
<tr>
<td>Speed = 12.59</td>
</tr>
</tbody>
</table>
Multi-Objective Search

- Objective functions:
  - Distance to Car “D(P/Car)”, Time To Collision “TTC”, and Distance to AWA “D(P/AWA)”
- The goal is to identify scenarios that minimize our three objectives at the same times in different environment situations
- Identify automatically most important risky environment situations
  - e.g., ramped roads, curved roads, blocked field of views, and animal as the object to detect
- Challenge: Simulation time => surrogate modeling?
- Found many failures in NiVi
Minimizing CPU Shortage Risks During Integration

References:

Software Integration
Stakeholders

**Car Makers**
- Develop software optimized for their specific hardware
- Provide part suppliers with runnables (exe)

**Part Suppliers**
- Integrate car makers software with their own platform
- Deploy final software on ECUs and send them to car makers
Different Objectives

Car Makers

- Objective: Effective execution and synchronization of runnables
- Some runnables should execute simultaneously or in a certain order

Part Suppliers

- Objective: Effective usage of CPU time
- Max CPU time used by all the runnables should remain as low as possible over time
An overview of an integration process in the automotive domain

Original Equipment Manufacturer (OEM)

AUTOSAR Models

Glue

sw runnables

DELPHI

Automotive Systems
CPU time shortage

• Static cyclic scheduling: predictable, analyzable
• Challenge
  – Many OS tasks and their many runnables run within a limited available CPU time
    • The execution time of the runnables may exceed their time slot
• Goal
  – Reducing the maximum CPU time used per time slot to be able to
    • Minimize the hardware cost
    • Reduce the probability of overloading the CPU in practice
    • Enable addition of new functions incrementally

(a)  
(b)
Using runnable offsets (delay times)

Offsets have to be chosen such that
the maximum CPU usage per time slot is minimized, and further,
the runnables respect their period
the runnables respect their time slot
the runnables satisfy their synchronization constraints
Without optimization

CPU time usage exceeds the size of the slot (5ms)
CPU time usage always remains less than 2.13ms, so more than half of each slot is guaranteed to be free.
Single-objective Search algorithms

Hill Climbing and Tabu Search and their variations

Solution Representation

a vector of offset values: $o_0=0$, $o_1=5$, $o_2=5$, $o_3=0$

Tweak operator

$o_0=0$, $o_1=5$, $o_2=5$, $o_3=0$ $\rightarrow$ $o_0=0$, $o_1=5$, $o_2=10$, $o_3=0$

Synchronization Constraints

offset values are modified to satisfy constraints

Fitness Function

max CPU time usage per time slot
Summary of Problem and Solution

Optimization
while satisfying synchronization/temporal constraints

Explicit Time Model
for real-time embedded systems

Search
meta-heuristic single objective search algorithms

10^27
an industrial case study with a large search space
Search Solution and Results

- The objective function is the max CPU usage of a 2s-simulation of runnables
- The search modifies one offset at a time, and updates other offsets only if timing constraints are violated
- Single-state search algorithms for discrete spaces (HC, Tabu)

Case Study: an automotive software system with 430 runnables, search space = $10^{27}$

- Running the system without offsets: 5.34 ms
- Optimized offset assignment: 2.13 ms
Comparing different search algorithms

Best CPU usage

Time to find Best CPU usage
Comparing our best search algorithm with random search

Lowest max CPU usage values computed by HC within 70 ms over 100 different runs

Lowest max CPU usage values computed by Random within 70 ms over 100 different runs

Comparing average behavior of Random and HC in computing lowest max CPU usage values within 70 s and over 100 different runs

HC

Random

Average
Trade-off between Objectives

Car Makers \( r_0 \) \( r_1 \) \( r_2 \) \( r_3 \) Part Suppliers

Execute \( r_0 \) to \( r_3 \) close to one another.

Minimize CPU time usage

1 slot

0ms 5ms 10ms 15ms 20ms 25ms 30ms 4ms

2 slots

0ms 5ms 10ms 15ms 20ms 25ms 30ms 3ms

3 slots

0ms 5ms 10ms 15ms 20ms 25ms 30ms 2ms
Trade-off curve

Interesting Solutions

Boundary Trade Offs

# of slots

CPU time usage (ms)
Multi-objective search

- Multi-objective genetic algorithms (NSGA II)
- Pareto optimality
- Supporting decision making and negotiation between stakeholders

Objectives:
- (1) Max CPU time
- (2) Maximum time slots between “dependent” tasks

Graph showing Max CPU Time Usage vs. Total Number of Time Slots for NSGA-II and Random methods.
Trade-Off Analysis Tool

Input.csv:
- runnables
- Periods
- CETs
- Groups
- # of slots per groups

Search

A list of solutions:
- objective 1 (CPU usage)
- objective 2 (# of slots)
- vector of group slots
- vector of offsets

Visualization/Query Analysis

- Visualize solutions
- Retrieve/visualize simulations
- Visualize Pareto Fronts
- Apply queries to the solutions
Conclusions

- Search algorithms to compute offset values that reduce the max CPU time needed
- Generate reasonably good results for a large automotive system and in a small amount of time
- Used multi-objective search tool for establishing trade-off between relaxing synchronization constraints and maximum CPU time usage
Schedulability Analysis and Stress Testing

References:

• S. Di Alesio et al., “Stress Testing of Task Deadlines: A Constraint Programming Approach”, IEEE ISSRE 2013, San Jose, USA
Real-time, concurrent systems (RTCS)

- Real-time, concurrent systems (RTCS) have concurrent interdependent tasks which have to finish before their deadlines.
- Some task properties depend on the environment, some are design choices.
- Tasks can trigger other tasks, and can share computational resources with other tasks.
- How can we determine whether tasks meet their deadlines?
Problem

- **Schedulability analysis** encompasses techniques that try to predict whether all (critical) tasks are schedulable, i.e., meet their deadlines.
- **Stress testing** runs carefully selected test cases that have a high probability of leading to deadline misses.
- Stress testing is **complementary** to schedulability analysis.
- Testing is typically expensive, e.g., hardware in the loop.
- Finding stress test cases is difficult.
Finding Stress Test Cases is Difficult

\[ j_0, j_1, j_2 \] arrive at \( at_0, at_1, at_2 \) and must finish before \( dl_0, dl_1, dl_2 \)

\[ j_0 \quad j_1 \quad j_2 \]

\[
\begin{array}{c|c|c|c}
0 & at_0 & \\
1 & & \\
2 & & \\
3 & & \\
4 & at_1 & \\
5 & dl_0 & \\
6 & & \\
7 & dl_1 & \\
8 & & \\
9 & & \\
\end{array}
\]

\[ j_0 \quad j_1 \quad j_2 \]

\[
\begin{array}{c|c|c|c}
0 & at_0 & \\
1 & & \\
2 & & \\
3 & & \\
4 & at_1 & \\
5 & dl_0 & \\
6 & & \\
7 & dl_1 & \\
8 & & \\
9 & & \\
\end{array}
\]

\[ J_1 \] can miss its deadline \( dl_1 \) depending on when \( at_2 \) occurs!
Challenges and Solutions

• Ranges for arrival times form a very large input space

• Task interdependencies and properties constrain what parts of the space are feasible

• We re-expressed the problem as a constraint optimisation problem

• Constraint programming (e.g., IBM CPLEX)
System monitors gas leaks and fire in oil extraction platforms

Drivers
(Software-Hardware Interface)

Control Modules

Real-Time Operating System

Multicore Architecture

Alarm Devices
(Hardware)
Constraint Optimization

Constraint Optimization Problem

- Static Properties of Tasks (Constants)
- Dynamic Properties of Tasks (Variables)
- OS Scheduler Behaviour (Constraints)
- Performance Requirement (Objective Function)
Process and Technologies

1. **System Design**
2. **System Platform**
3. **Deadline Misses Analysis**
4. **Stress Test Cases**

**INPUT**
- Design Model (Time and Concurrency Information)

**Constraint Optimization**
- Optimization Problem (Find arrival times that maximize the chance of deadline misses)
- Constraint Programming (CP)

**OUTPUT**
- Solutions (Task arrival times likely to lead to deadline misses)

**UML Modeling (e.g., MARTE)**

System Design Solutions (Task arrival times likely to lead to deadline misses)
Challenges and Solutions

• **Scalability problem:** Constraint programming (e.g., IBM CPLEX) cannot handle such large input spaces (CPU, memory)

• **Solution:** Combine metaheuristic search and constraint programming
  – metaheuristic search (GA) identifies high risk regions in the input space
  – constraint programming finds provably worst-case schedules within these (limited) regions
  – Achieve (nearly) GA efficiency and CP effectiveness
Combining GA and CP

Fig. 3: Overview of GA+CP: the solutions $x_1$, $y_1$, and $z_1$ in the initial population of GA evolve into $x_6$, $y_6$, and $z_6$, then CP searches in their neighborhood for the optimal solutions $x^\star$, $y^\star$, and $z^\star$.

**Union set $I^\star$ of impacting sets of tasks missing or closest to miss their deadlines.** Let $I^\star(x)$ be the union of the impacting sets of tasks in $J^\star(x)$:

$$I^\star(x) = \bigcup_{j \in J^\star(x)} \bigcup_{k \in K} I_{j,k}(x)$$

By definition, $I^\star(x)$ contains all the tasks that can have an impact over a task that misses a deadline or is closest to a deadline miss.

**Neighborhood $\varepsilon$ of an arrival time and neighborhood size $D$.** Let $\varepsilon(x_{j,k})$ be the interval centered in the arrival time $x_{j,k}$ computed by GA, and let $D$ be its radius:

$$\varepsilon(x_{j,k}) = [x_{j,k} - D, x_{j,k} + D]$$

$\varepsilon$ defines the part of the search space around $x_{j,k}$ where to find arrival times that are likely to break task deadlines. $D$ is a parameter of the search.

**Constraint Model $M$ implementing a Complete Search Strategy.** Let $M$ be the constraint model defined in our previous work [Di Alesio et al. 2014] for the purpose of identifying arrival times for tasks that are likely to lead to deadline misses scenarios. $M$ models the static and dynamic properties of the software system respectively as constants and variables, and the scheduler of the operating system as a set of constraints among such variables. Note that $M$ implements a complete search strategy over the space of arrival times. This means that $M$ searches for arrival times of all aperiodic tasks within the whole interval $T$. Our combined GA+CP strategy consists in the following two steps:
Process and Technologies

**UML Modeling (e.g., MARTE)**

**Constraint Optimization**

**INPUT**

- System Design
- System Platform

**Design Model (Time and Concurrency Information)**

**Optimization Problem**

- (Find arrival times that maximize the chance of deadline misses)

**OUTPUT**

- Genetic Algorithms (GA)
- Constraint Programming (CP)

**Solutions**

- (Task arrival times likely to lead to deadline misses)

**Deadline Misses Analysis**

**System Design**

**System Platform**
Environment-Based Testing of Soft Real-Time Systems

References:

Objectives

- Model-based system testing
  - Independent test team
  - Black-box
  - Environment models
Environment: Domain Model

```
<Context>
  RVM
  - notRoutingFlag : Boolean
  - <signal> user_inserts_item()
  - <signal> SUT_item_arrived()
  - <signal> ITEM_LOST()

<Context>
  User
  - count : Integer
  - <NonDeterministic> insertTime : Integer
  - <signal> rvm_sends_item()

<Context>
  Sorter
  - <NonDeterministic> moveArmTimeLC : Integer
  - <NonDeterministic> moveArmTimeCR : Integer
  - destination : String
    - <signal> POSITION_RIGHT()
    - <signal> POSITION_CENTRE()
    - <signal> POSITION_LEFT()
    - <signal> item_at_destination()
```

```
<NonDeterministic>
  Sorter::moveArmTimeLC {lowerBound = 280, upperBound = 320, scope = state}
  Sorter::moveArmTimeCR {lowerBound = 280, upperBound = 320, scope = state}
```
Environment: Behavioral Model

```
<Context>
  Sorter
  <NonDeterministic> moveArmTimeLC : Integer
  <NonDeterministic> moveArmTimeCR : Integer
  destination : String
  <signal> POSITION_RIGHT()
  <signal> POSITION_CENTRE()
  <signal> POSITION_LEFT()
  <signal> item_at_destination()
```

```
injection();
Left
POSITION_CENTRE()
/destination = "centre";

POSITION_RIGHT()
/Effect

after "moveArmTimeLC, ms"
MovingLeftCentre
estimation = 'left'

POSITION_LEFT()
/destination = "left";

Centre
```

```
MovingCentreRight
```

```
Item_at_destination()
```

```
Error State
```

```
bordered
```

```
/destination = "centre";
```

```
/destination = "left";
```

```
/destination = "right";
```

```
SHT() "right";
```

```
SHT() "left";
```

```
SHT() "right";
```

```
SHT() "left";
```
Test Case Generation

- Test objectives: Reach “error” states (critical environment states)
- Test Case: Simulation Configuration
  - Setting non-deterministic properties of the environment, e.g., speed of sorter’s left and right arms
- Oracle: Reaching an “error” state
- Metaheuristics: search for test cases getting to error state
- Fitness functions
  - Distance from error state
  - Distance from satisfying guard conditions
  - Time distance
  - Time in “risky” states
Stress Testing focused on Concurrency Faults

Reference:

Stress Testing of Distributed Systems

Reference:

General Pattern: Using Metaheuristic Search

- Problem = fault model
- Model = system or environment
- Search to optimize objective function(s)
- Metaheuristics, constraint programming
- Scalability: A small part of the search space is traversed
- Model: Guidance to worst case, high risk scenarios across space
- Reasonable modeling effort based on standards or extension
- Heuristics: Extensive empirical studies are required
General Pattern: Using Metaheuristic Search

- Simulation can be time consuming
- Makes the search impractical or ineffective
- Surrogate modeling based on machine learning
- Simulator dedicated to search
Scalability
Project examples

• Scalability is the most common verification challenge in practice

• Testing closed-loop controllers, vision system
  – Large input and configuration space
  – Smart heuristics to avoid simulations (machine learning)

• Schedulability analysis and stress testing
  – Large space of possible arrival times
  – Constraint programming cannot scale by itself
  – CP was carefully combined with genetic algorithms
Scalability: Lessons Learned

- Scalability must be part of the problem definition and solution from the start, not a refinement or an after-thought.
- Meta-heuristic search, by necessity, has been an essential part of the solutions, along with, in some cases, machine learning, statistics, etc.
- Scalability often leads to solutions that offer “best answers” within time constraints, but no guarantees.
- Scalability analysis should be a component of every research project – otherwise it is unlikely to be adopted in practice.
- How many papers research papers do include even a minimal form of scalability analysis?
Practicality
Project examples

• Practicality requires to account for the domain and context

• Testing controllers
  – Relies on Simulink only
  – No additional modeling or complex translation
  – Within domains, differences have huge implications in terms of applicability (open versus closed loop controllers)

• Minimizing risks of CPU shortage
  – Trade-off between effective synchronisation and CPU usage
  – Trade-off achieved through multiple-objective GA search and appropriate decision tool

• Schedulability analysis and stress testing
  – Near deadline misses must also be identified
Practicality: Lessons Learned

- In software engineering, and verification in particular, just understanding the real problems in context is difficult.
- What are the inputs required by the proposed technique?
- How does it fit in development practices?
- Is the output what engineers require to make decisions?
- There is no unique solution to a problem as they tend to be context dependent, but a context is rarely unique and often representative of a domain or type of system.
Discussion

• **Metaheuristic search for verification**
  – Tends to be versatile, tailorable to new problems and contexts
  – Can cope with the verification of continuous behavior
  – Entails acceptable modeling requirements
  – Can provide “best” answers at any time
  – Scalable, practical

**But**

– Not a proof, no certainty
– Effectiveness of search guidance is key and must be experimented and evaluated
– Models are key to provide adequate guidance
– Search must often be combined with other techniques, e.g., machine learning
Discussion II

- **Constraint solvers (e.g., Comet, ILOG CPLEX, SICStus)**
  - Is there an efficient constraint model for the problem at hand?
  - Can effective heuristics be found to order the search?
  - Better if there is a match to a known standard problem, e.g., job shop scheduling
  - Tend to be strongly affected by small changes in the problem, e.g., allowing task pre-emption
  - Often not scalable, e.g., memory

- **Model checking**
  - Detailed operational models (e.g., state models), involving (complex) temporal properties (e.g., CTL)
  - Enough details to analyze statically or execute symbolically
  - These modeling requirements are usually not realistic in actual system development. State explosion problem.
  - Originally designed for checking temporal properties through reachability analysis, as opposed to explicit timing properties
  - Often not scalable
Talk Summary

- Focus: Meta-heuristic Search to enable scalable verification and testing.
- Scalability is the main challenge in practice.
- We drew lessons learned from example projects in collaboration with industry, on real systems and in real verification contexts.
- Results show that meta-heuristic search contributes to mitigate the scalability problem.
- It has also shown to lead to applicable solutions in practice.
- Solutions are very context dependent.
- Solutions tend to be multidisciplinary: system modeling, constraint solving, machine learning, statistics.
Making Model-Driven Verification Practical and Scalable - Experiences and Lessons Learned

Lionel Briand

Interdisciplinary Centre for ICT Security, Reliability, and Trust (SnT)
University of Luxembourg, Luxembourg

MODELSWARD, Rome, February 20, 2016

SVV lab: svv.lu
SnT: www.securityandtrust.lu