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

Lionel Briand
IEEE Fellow, FNR PEARL Chair

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

STAF, L’Aquila, Italy, 2015
SnT Software Verification and Validation Lab

- SnT centre, Est. 2009: Interdisciplinary, ICT security-reliability-trust
- 230 scientists and Ph.D. candidates, 20 industry partners
- 25 scientists (Research scientists, associates, and PhD candidates)
- Industry-relevant research on system dependability: security, safety, reliability
- Six partners: Cetrel, CTIE, Delphi, SES, IEE, Hitec …
An Effective, Collaborative Model of Research and Innovation

Schneiderman, 2013

- Basic and applied research take place in a rich context
- Basic Research is also driven by problems raised by applied research, which is itself fed by innovation and development
- Publishable research results and focused practical solutions that serve an existing market.
Collaboration in Practice

- Well-defined problems in context
- Realistic evaluation
- Long term industrial collaborations
Motivations

• The term “verification” is used in its wider sense: Defect detection and removal, design or run-time, before or after deployment

• One important application of models is to drive and automate verification

• In practice, despite significant advances, e.g., model-based testing, this is not commonly part of mainstream engineering

• Decades of research have not yet significantly and widely impacted practice
Applicability?
Scalability?
Definitions

• **Applicable:** Can a technology be efficiently and effectively applied by engineers in realistic conditions?
  – realistic ≠ universal

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

- Project examples, with industry collaborations
- Lessons learned and reflections regarding developing applicable and scalable verification solutions
### Some Past Projects (< 5 years)

<table>
<thead>
<tr>
<th>Company</th>
<th>Domain</th>
<th>Objective</th>
<th>Notation</th>
<th>Automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cisco</td>
<td>Video conference</td>
<td>Robustness testing</td>
<td>UML profile</td>
<td>Search, model transformation</td>
</tr>
<tr>
<td>Kongsberg Maritime</td>
<td>Oil &amp; Gas</td>
<td>CPU usage and RT deadlines</td>
<td>UML+MARTE</td>
<td>Constraint Solving</td>
</tr>
<tr>
<td>WesternGeco</td>
<td>Marine seismic acquisition</td>
<td>Functional testing</td>
<td>UML profile + MARTE</td>
<td>Search, constraint solving</td>
</tr>
<tr>
<td>SES</td>
<td>Satellite</td>
<td>Functional and robustness testing, requirements QA</td>
<td>UML profile</td>
<td>Search, Model mutation, NLP</td>
</tr>
<tr>
<td>Delphi</td>
<td>Automotive control systems</td>
<td>Testing safety + performance</td>
<td>Matlab/Simulink</td>
<td>Search, machine learning, statistics</td>
</tr>
<tr>
<td>CTIE</td>
<td>Legal &amp; financial</td>
<td>Legal Requirements testing &amp; simulation</td>
<td>UML Profile</td>
<td>Model transformation, constraint checking</td>
</tr>
<tr>
<td>HITEC</td>
<td>Crisis Support systems</td>
<td>Testing security, Access Control</td>
<td>UML Profile</td>
<td>Constraint verification, machine learning, Search</td>
</tr>
<tr>
<td>CTIE</td>
<td>eGovernment</td>
<td>Offline and run-time verification</td>
<td>UML Profile, BPMN, OCL extension</td>
<td>Domain specific language, Constraint checking</td>
</tr>
<tr>
<td>IEE</td>
<td>Automotive sensor systems</td>
<td>Requirements-driven testing, traceability</td>
<td>UML profile, Use Case Modeling extension, Matlab/Simulink</td>
<td>NLP, Constraint solving</td>
</tr>
</tbody>
</table>
Testing Software Controllers

References:

Dynamic Continuous Controllers
Electronic Control Units (ECUs)

Comfort and variety

More functions

Safety and reliability

Faster time-to-market

Greenhouse gas emission laws

Less fuel consumption
A Taxonomy of Automotive Functions

Computation
- Transforming: unit convertors
- Calculating: calculating positions, duty cycles, etc

Controlling
- State-Based: State machine controllers
- Continuous: Closed-loop controllers (PID)

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
MATLAB/Simulink model

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

Desired value → Error → Controller (SUT) → Plant Model → System output

Actual value

Test Input

Test Output

Initial Desired Value

Final Desired Value

T/2
T

T/2
T

Desired Value
Actual Value
Configuration parameters

Inputs: Time-dependent variables

Plant Model

Actual(t)

Desired(t)

Output(t)

Configuration Parameters
Requirements and Test Objectives

- Initial Desired (ID)
- Desired Value (input)
- Actual Value (output)
- Final Desired (FD)

- Smoothness
- Responsiveness
- Stability

Desired Value (input)
Actual Value (output)
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
Search Elements

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

- **Search Technique:**
  - 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 break requirements at MiL level
  - High-risk, stress test cases based on actual hardware (HiL)
Smoothness Objective Functions: $O_{\text{Smoothness}}$

$O_{\text{Smoothness}}(\text{Test Case A}) > O_{\text{Smoothness}}(\text{Test Case B})$

We want to find test scenarios which maximize $O_{\text{Smoothness}}$
Solution Overview (Simplified Version)

Objective Functions based on Requirements + Controller-plant model

1. Exploration
   HeatMap Diagram

Domain Expert
   List of Critical Regions

2. Single-State Search
   Worst-Case Scenarios

Graph Builder

Final vs. Initial

Desired Value
Actual Value

Initial Desired
Final Desired
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

- The higher the fitness, the worse the test scenario
- How efficiently do we converge towards a worst case scenario?
Analysis II – Search over different regions

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)

![Diagram](image)

**Fig. 9.** Diagrams representing the landscape for two representative HeatMap regions: (a) Landscape for the region in Figure 7(b). (b) Landscape for the region in Figure 7(a).

Our observations show that the regions surrounded mostly by dark shaded regions typically have a clear gradient between the initial point of the search and the worst case point (see e.g., Figure 9(a)). However, dark regions located in a generally light shaded area have a noisier shape with several local optimum (see e.g., Figure 9(b)). It is known that for regions like Figure 9(a), exploitative search works best, while for those like Figure 9(b), explorative search is most suitable [10]. This is confirmed in our work where for Figure 9(a), our exploitative search, i.e., (1+1) EA with $t = 0.01$, is faster and more effective than random search, whereas for Figure 9(b), our search is slower than random search. We applied a more explorative version of (1+1) EA where we let $t = 0.03$ to the region in Figure 9(b). The result (Figure 10) shows that the more explorative (1+1) EA is now both faster and more effective than random search. We conjecture that, from the HeatMap diagrams, we can predict which search algorithm to use for the single-state search step. Specifically, for dark regions surrounded by dark shaded areas, we suggest an exploitative (1+1) EA (e.g., $t = 0.01$), while for dark regions located in light shaded areas, we recommend a more explorative (1+1) EA (e.g., $t = 0.03$).
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
   - Worst-Case Scenarios

Visualization of the multi-dimensional space using regression trees

Dimensionality reduction to identify the significant variables

Surrogate modeling to predict the objective function and speed up the search
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

![Sample Standard Deviation vs Sample Mean](image)

- FD
- Cal5
- Cal3
- Cal6
- Cal4
- Cal1, Cal2
- ID

Sample Standard Deviation ($S_{\delta}$)

Sample Mean ($\bar{\delta}_i$)
Elementary Effects Analysis Method

Check mark function \( F \) with 2 inputs, \( x \) and \( y \):

\[
\begin{align*}
F(A1) - F(A) & \\
F(B1) - F(B) & \\
F(C1) - F(C) & \\
& \\
\vdots & \\
F(A2) - F(A) & \\
F(B2) - F(B) & \\
F(C2) - F(C) & \\
& \\
\vdots & 
\end{align*}
\]

Elementary Effects for \( X \) and \( Y \):

- for \( X \):
  - \( F(A1) - F(A) \)
  - \( F(B1) - F(B) \)
  - \( F(C1) - F(C) \)
  - \( \ldots \)

- for \( Y \):
  - \( F(A2) - F(A) \)
  - \( F(B2) - F(B) \)
  - \( F(C2) - F(C) \)
  - \( \ldots \)
Visualization in Inputs & Configuration Space

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

**Fitness**

![Graph showing real function and surrogate model](Image)
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 ( R^2/\text{MRPE} )</th>
<th>ER ( R^2/\text{MRPE} )</th>
<th>PR(n=2) ( R^2/\text{MRPE} )</th>
<th>PR(n=3) ( R^2/\text{MRPE} )</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

- Open 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

- Adaptive Random Selection
- White-box Structural Coverage
  - State Coverage
  - Transition Coverage
- Output Diversity
- Failure-Based Selection Criteria (search)
  - Domain specific failure patterns
  - Output Stability
  - Output Continuity
Stability of Output Signals
Discontinuities in Output Signals
Automatic Generation of System Test Cases from Use Case Specifications

References:

- Chunhui Wang et al., “UMTG: A Toolset to Automatically Generate System Test Cases from Use Case Specifications”, ESEC/FSE 2015 Tool track
Problem

• Verifying the compliance of a system with its requirements in a cost-effective way
• Traceability between requirements and system test cases – This cannot be manual
• E.g., automotive industry which must comply to ISO 26262
Objectives

- Automatically generate test cases from requirements
- Capture traceability information between test cases and requirements during generation
Context Matters

– Requirements are captured through use cases (common)
– Use cases are used to communicate with customers and the system test team
– Complete and precise behavioral models are not an option: too complex
Strategy

- Analyzable use case specifications (RUCM)
- Automatically extract test model from the use case specifications through Natural Language Processing
- Minimize modeling
- No behavioral modeling
## Costs of Additional Modeling

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Steps</th>
<th>Use Case Flows</th>
<th>OCL Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC1</td>
<td>50</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>UC2</td>
<td>44</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>UC3</td>
<td>35</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>UC4</td>
<td>59</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>UC5</td>
<td>30</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>UC6</td>
<td>25</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>

5 to 10 minutes to write each constraints
Effectiveness: scenarios covered

It is hard for engineers to capture all the possible scenarios involving error conditions.
Modeling and Verifying Legal Requirements

Reference:


Context and Problem

• CTIE: Government computer centre in Luxembourg

• Large government (information) systems

• Implement legal requirements, must comply with the law

• The law usually leaves room for interpretation and changes on a regular basis, many cross-references

• Involves many stakeholders, IT specialists but also legal experts, etc.
Art. 105bis [...]The commuting expenses deduction (FD) is defined as a function over the distance between the principal town of the municipality on whose territory the taxpayer's home is located and the place of taxpayer’s work. The distance is measured in units of distance expressing the kilometric distance between [principal] towns. A ministerial regulation provides these distances.

The amount of the deduction is calculated as follows: If the distance exceeds 4 units but is less than 30 units, the deduction is € 99 per unit of distance. The first 4 units does not trigger any deduction and the deduction for a distance exceeding 30 units is limited to € 2,574.
## Project Objectives

<table>
<thead>
<tr>
<th>Objective</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification of legal requirements</td>
<td>• Make interpretation of the law explicit</td>
</tr>
<tr>
<td>• including rationale and traceability to the text of law</td>
<td>• Improve communication</td>
</tr>
<tr>
<td>• Prerequisite for automation</td>
<td>• Prevent errors in the interpretation of the law to propagate</td>
</tr>
<tr>
<td>Checking consistency of legal requirements</td>
<td></td>
</tr>
<tr>
<td>Automated test strategies for checking system compliance to legal</td>
<td>• Provide effective and scalable ways to verify compliance</td>
</tr>
<tr>
<td>requirements</td>
<td></td>
</tr>
<tr>
<td>Run-time verification mechanisms to check compliance with legal</td>
<td></td>
</tr>
<tr>
<td>requirements</td>
<td></td>
</tr>
<tr>
<td>Analyzing the impact of changes in the law</td>
<td>• Decrease costs and risks associated with change</td>
</tr>
<tr>
<td></td>
<td>• Make change more predictable</td>
</tr>
</tbody>
</table>
Solution Overview

- Test cases
  - Input to Actual software system
  - Generates Analyzable interpretation of the law
  - Traces to Actual result
  - Traces to Simulated result

Actual result
- Results match?
  - Yes
  - No

Simulated result
- Impact of legal changes

Generates
- Impact of legal changes

Traces to
- Analyzable interpretation of the law
Research Steps

1. Conduct grounded theory study
   - What information requirements should we expect?
   - What are the complexity factors?

2. Build UML profile
   - Explicit means for capturing information requirements
   - Basis for modeling methodology
   - Target: Legal experts and IT specialists

3. Model Transformation to enable V&V
   - Target existing automation techniques
   - Solvers for testing
   - Simulation language or library
Art. 105bis [...] The commuting expenses deduction (FD) is defined as a function over the distance between the principal town of the municipality on whose territory the taxpayer's home is located and the place of taxpayer’s work. The distance is measured in units of distance expressing the kilometric distance between [principal] towns. A ministerial regulation provides these distances.
The amount of the deduction is calculated as follows: If the distance exceeds 4 units but is less than 30 units, the deduction is €99 per unit of distance. The first 4 units does not trigger any deduction and the deduction for a distance exceeding 30 units is limited to €2,574.
Challenges and Results

• Profile must lead to models that are:
  – understandable by both IT specialists and legal experts
  – precise enough to enable model transformation and support our objectives

• Tutorials, many modeling sessions with legal experts

• In theory, though such legal requirements can be captured by OCL constraints alone, this is not applicable

• That is why we resorted to customized activity modeling, carefully combined with a simple subset of OCL

• Many traces to law articles, dependencies among articles: automated detection (NLP) of cross-references
Run-Time Verification of Business Processes

References:

CTIE: Government Computing Centre of Luxembourg

E-government systems mostly implemented as business processes

CTIE models these business processes (MDE methodology)

Business models have temporal properties that must be checked
  - Temporal logics not applicable by analysts
  - Limited tool support (scalability)

Goal: Efficient, scalable, and practical off-line and run-time verification in an MDE context
Solution Overview
Solution Overview

- We identified patterns based on analyzing many properties of real business process models
- How to facilitate their expression?
Solution Overview

- Want to transform the checking of temporal constraints into checking regular constraints on trace conceptual model
- OCL engines (Eclipse) are our target, to rely on mature technology (scalability)
- Defined a more expressive extension of OCL (OCLR), resembling natural language, to facilitate translation
- Optimized translation
Scalability Analysis

- Analyzed 47 properties in Identity Card Management System
- “Once a card request is approved, the applicant is notified within three days; this notification has to occur before the production of the card is started.”
- Scalability: Check time as a function of trace size …
Schedulability Analysis and Stress Testing

References:

• S. Di Alesio et al. “Stress Testing of Task Deadlines: A Constraint Programming Approach”, ISSRE 2013, San Jose, USA
Problem

- 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.
- 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.
- Testing in RTCS is typically expensive, e.g., hardware in the loop.
Arrival Times Determine Deadline Misses

\[ j_0, j_1, j_2 \text{ arrive at } at_0, at_1, at_2 \text{ and must finish before } dl_0, dl_1, dl_2 \]

\[ J_1 \text{ can miss its deadline } dl_1 \text{ depending on when } at_2 \text{ occurs!} \]
Driver Modules

Drivers

(Software-Hardware Interface)

Control Modules

Real-Time Operating System

Multicore Architecture

Alarm Devices

(Hardware)

Monitor gas leaks and fire in oil extraction platforms
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
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

UML Modeling (e.g., MARTE)

Constraint Optimization

System Design

System Platform

Design Model (Time and Concurrency Information)

Deadline Misses Analysis

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

Constraint Programming (CP)

Stress Test Cases

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

INPUT

OUTPUT
Challenges and Solutions (2)

• 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 identifies high risk regions in the input space
  – constraint programming finds provably worst-case schedules within these (limited) regions
Combining CP and GA

A:12

S. Di Alesio et al.

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 $z^\star$.

$\varepsilon$(J) 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 $J^\star(x)$:

$I^\star(x) \text{def} = \bigcup_{j^\star \in J^\star(x)} I_{j^\star}(x)$

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.

$\varepsilon$ 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:
Applicable? Scalable?
Scalability Examples

• This is the most common SE challenge in practice
• Testing software controllers
  – Large input and configuration space
  – Smart search optimization heuristics (e.g., sensitivity analysis, machine learning)
• Requirements-Driven Testing from UCS
  – NLP of restricted use case specifications
  – OCL constraints solving using metaheuristic search
• Schedulability analysis and stress testing
  – Constraint programming cannot scale by itself
  – Carefully combined with genetic algorithms
Scalability Examples (2)

• Verifying legal requirements
  – Many provisions and articles in laws: Traceability is complex
  – Many dependencies within the law
  – Natural Language Processing: Cross references, support for identifying missing modeling concepts
  – Simulation: Generating large number of model instances satisfying constraints and probability distributions

• Run-time Verification of Business Processes
  – Traces can be large and properties complex to verify
  – Optimized transformations of temporal properties into regular OCL properties, defined on a trace conceptual model
  – Incremental verification at regular time intervals
  – Heuristics to identify which subtraces to verify
Scalability: Lessons Learned

- Scalability must be part of the problem definition and solution from the start, not a refinement or an after-thought.
- It often involves heuristics, e.g., meta-heuristic search, NLP, machine learning, statistics.
- Scalability often leads to solutions that offer “best answers” within time constraints, not guarantees.
- Solutions to achieve scalability are multidisciplinary.
- Scalability analysis should be a component of every research project – otherwise it is unlikely to be adopted in practice.
- How many research papers do include even a minimal form of scalability analysis?
Applicability

- Can the target user population efficiently apply it?

- Assumptions: Are working assumptions realistic, e.g., realistic information requirements?

- Integration into the development process, e.g., are required inputs available in the right form and level of precision?
Applicability examples

• Testing software controllers
  – Working assumption: availability of sufficiently precise plant (environment) models
  – Means to visualize relevant properties in the search space (inputs, configuration), to get an overview and focus search on high-risk areas

• Schedulability analysis and stress testing
  – Availability of tasks architecture models
  – Accurate WCET analysis
  – Assess risk based on near-deadline misses as well
Applicability examples (2)

• Requirements-Driven Testing from UCS
  – Precise behavioral scenarios are not an option
  – Number of constraints
  – Effort of writing constraints

• Run-time verification of business process models
  – Temporal logic not usable by analysts
  – Language closer to natural language, directly tied to business process model
  – Easy transition to industry strength constraint checker (OCL)

• Verifying legal requirements
  – Modeling notation must be shared by IT specialists and legal experts
  – One common representation for many applications, with traces to the law to handle changes
  – Multiple model transformation targets: Simulation, testing …
Applicability: Lessons Learned

• Make working assumptions explicit: Determine the context of applicability

• Make sure those working assumptions are at least realistic in some industrial domain and context

• Assumptions don’t need to be universally true – they rarely are anyway. No solution needs to be universally applicable

• Run industrial case studies and usability studies – do it for real!
Conclusions

- In most research endeavors, applicability and scalability are an afterthought, a secondary consideration, when at all considered.
- Implicit assumptions are often made, often unrealistic in any context.
- Problem definition in a vacuum.
- Not adapted to research in an engineering discipline.
- Leads to limited impact.
- Research in model-based V&V is necessarily multi-disciplinary.
- Industrial case studies and user studies are required and far too rare.
- In engineering research, there is no substitute to reality.
Acknowledgements

PhD. Students:
• Reza Matinnejad
• Stefano Di Alesio
• Wei Dou
• Ghanem Soltana
• Chunhui Wang

Scientists:
• Shiva Nejati
• Mehrdad Sabetzadeh
• Domenico Bianculli
• Fabrizio Pastore
• Arnaud Gotlieb
Making Model-Driven Verification Practical and Scalable: Experiences and Lessons Learned

Lionel Briand
IEEE Fellow, FNR PEARL Chair
Interdisciplinary Centre for ICT Security, Reliability, and Trust (SnT)
University of Luxembourg, Luxembourg

STAF, L’Aquila, Italy, 2015

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