Scalable and Cost-Effective Model-Based Software Verification and Testing

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

May 17th, 2013
University of California, Irvine

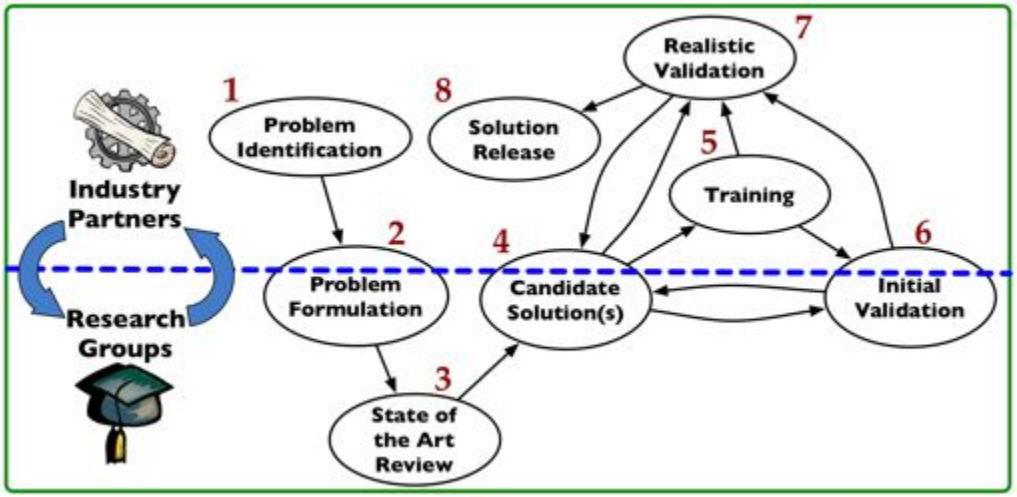
Lionel Briand, IEEE Fellow
FNR PEARL Chair

Luxembourg

- Small country and population
- One of the wealthiest in the world
- Young university (2003) and Ph.D. programs (2007)
- ICT security and reliability, a national research priority
- Priorities implemented as interdisciplinary centres
- International
- Three official languages: English, French, German

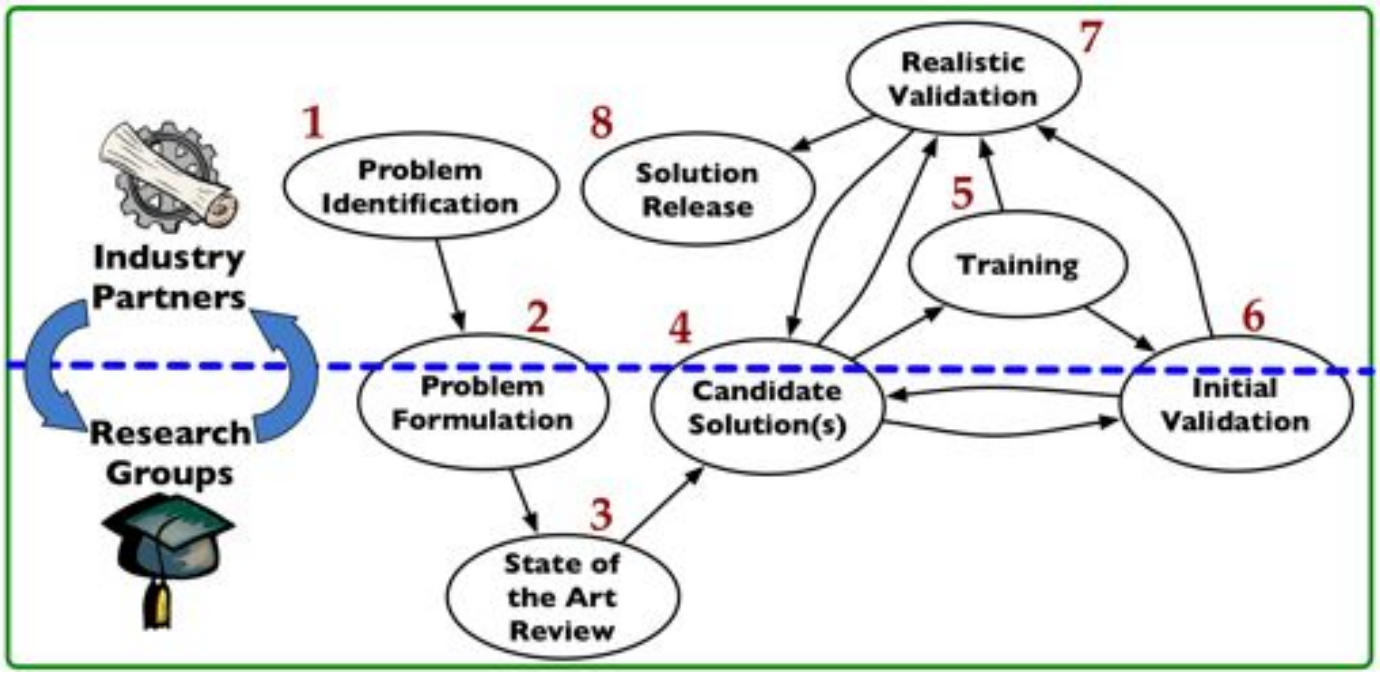
SnT Software Verification and Validation Lab

- SnT centre, Est. 2009: Interdisciplinary, ICT security-reliability-trust
- 180 scientists and Ph.D. candidates, 20 industry partners
- SVV Lab: Established January 2012, www.svv.lu
- 15 scientists (Research scientists, associates, and PhD candidates)
- Industry-relevant research on system dependability: security, safety, reliability
- Four partners: Cetrel, CTIE, Delphi, SES, …



Research Paradigm

- Research informed by practice
- Well-defined problems in context
- Realistic evaluation
- Long term industrial collaborations

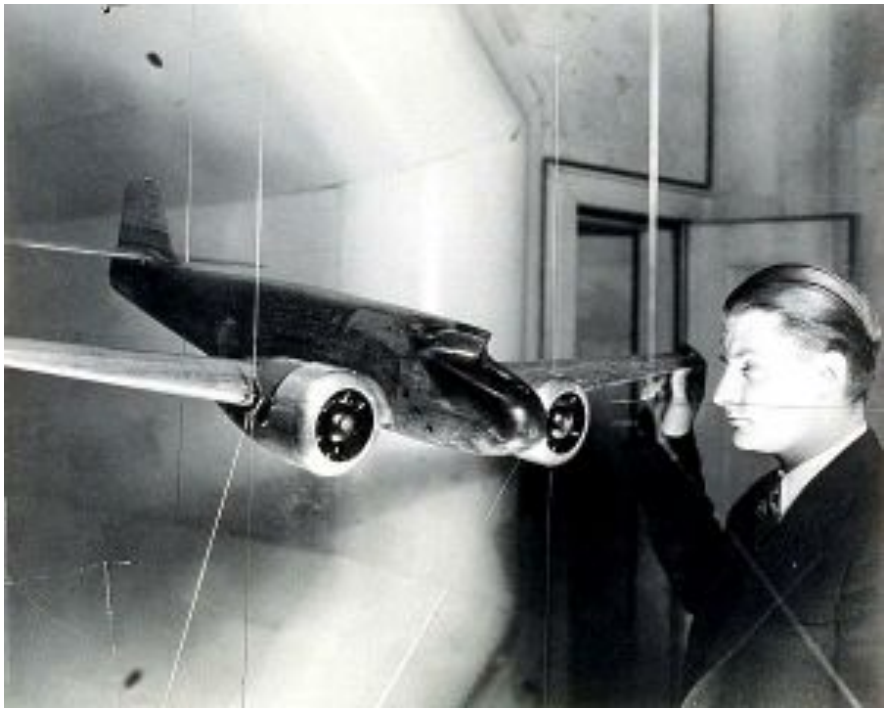


Acknowledgements

- Shiva Nejati
- Mehrdad Sabetzadeh
- Yvan Labiche
- Andrea Arcuri
- Stefano Di Alesio
- Reza Matinnejad
- Zohaib Iqbal
- Shaukat Ali
- Hadi Hemmati
- Marwa Shousha
- …

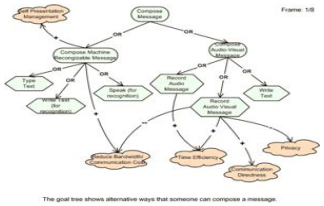
“Model-based”?

- All engineering disciplines rely on abstraction and therefore models
- In most cases, it is the only way to effectively automate testing or verification
- Models have many other purposes: Communication, support requirements and design
- There are many ways to model systems and their environment
- In a given context, this choice is driven by the application domain, standards and practices, objectives, and skills

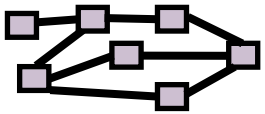


Models in Software Engineering

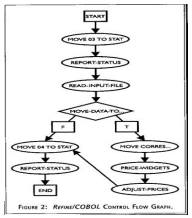
- **Model:** An abstract and analyzable description of software artifacts, created for a purpose



Requirements models



Architecture models



Behavioural models

Test models

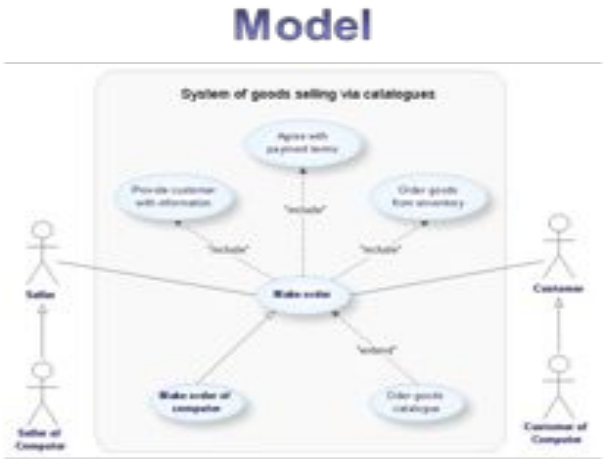
- **Abstract:** Details are omitted. Partial representation. Much smaller and simpler than the artifact being modeled.

- **Analyzable:** Leads to task automation

Talk Objectives

- Overview of several years of research
- Examples, at various levels of details
- Follows a research paradigm that is uncommon in software engineering research
- Conducted in collaboration with industry partners in many application domains: Automotive, energy, telecom …
- Lessons learned regarding scalability and cost-effectiveness

Research Pattern: Models and Search Heuristics



Objective Function

Search Space

Search Technique

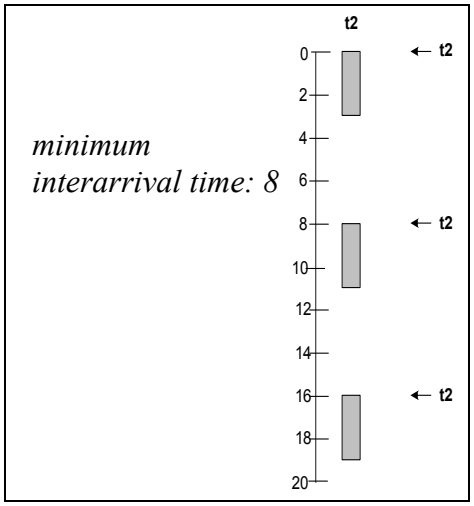
- Search to optimize objective function: Complete or not, deterministic or partly random (stochastic)
- Metaheuristics, constraint solvers
- Scalability: A small part of the search space is traversed
- Model: Guidance to worst case, high risk scenarios across space
- Heuristics: Extensive empirical studies are required

Early Work: Search-Based Schedulability Analysis

L. Briand, Y. Labiche, and M. Shousha, 2003-2006

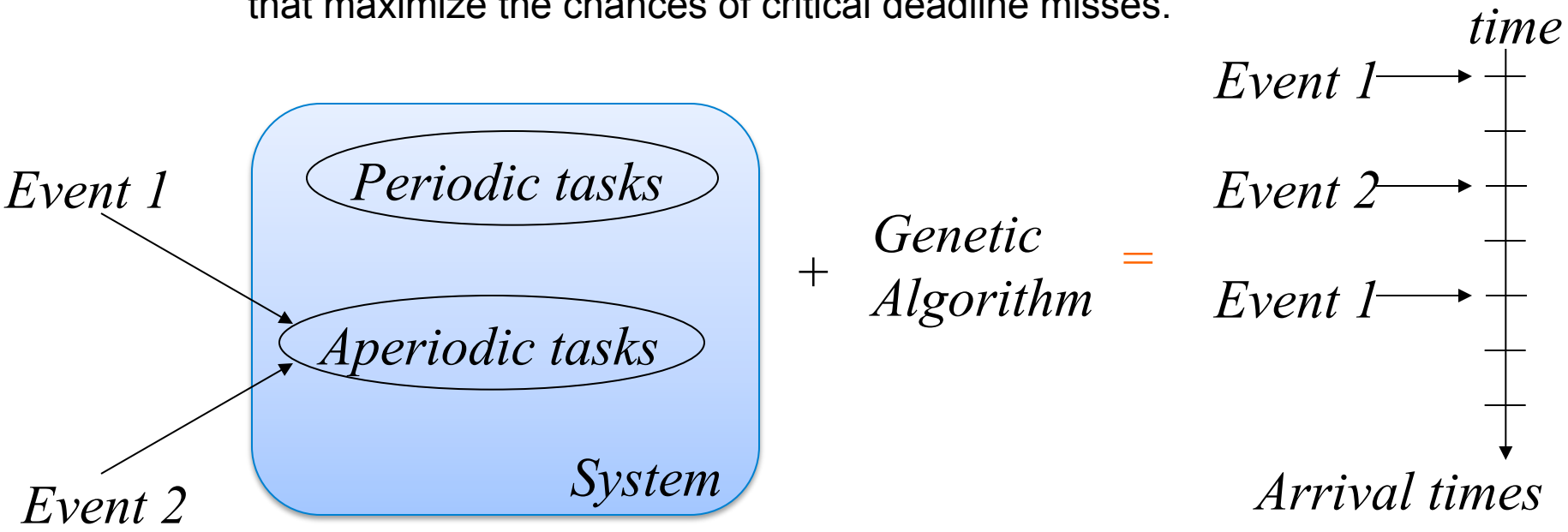
Schedulability Theory

- 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 ready to start at time zero
- Execution times are estimates

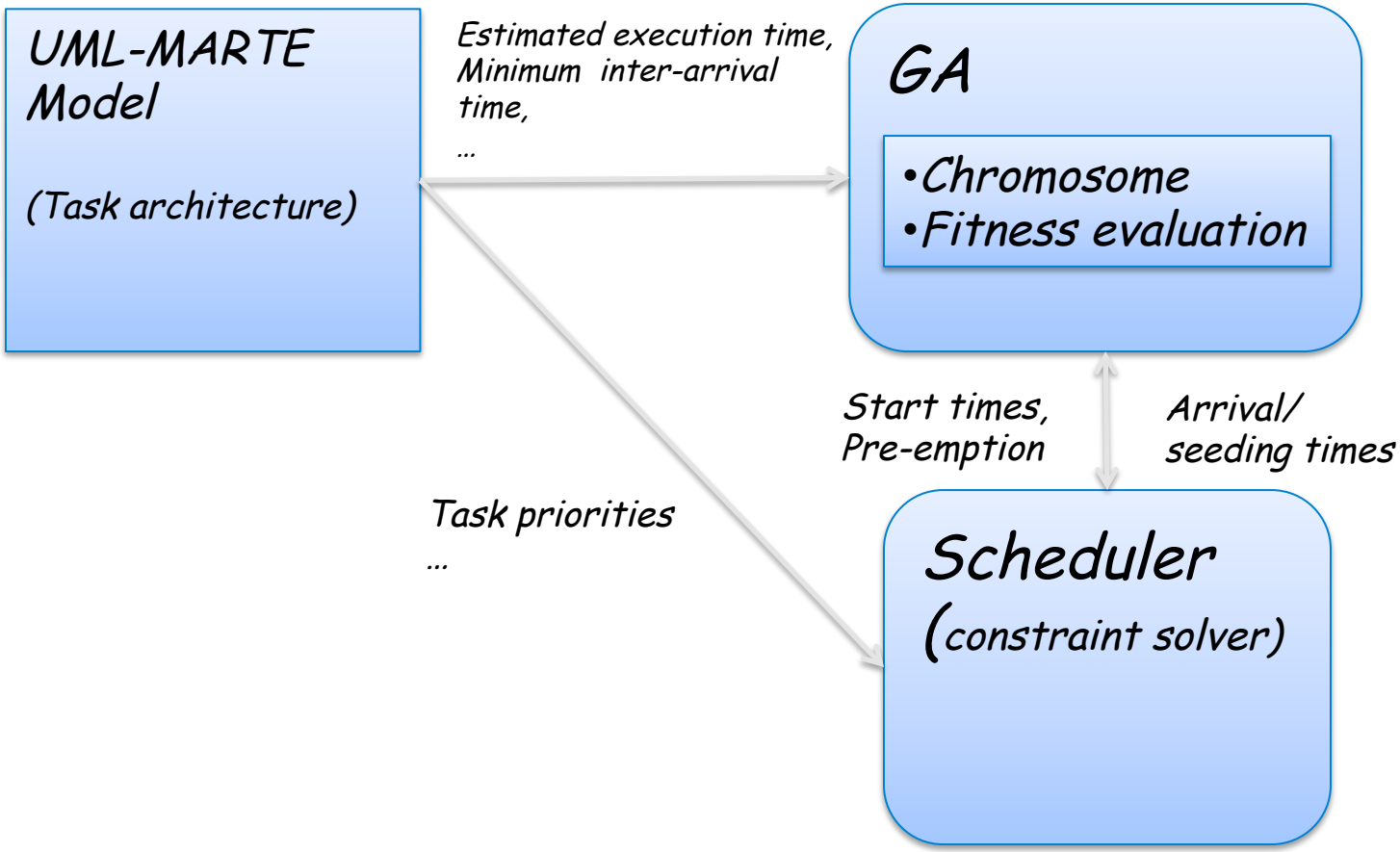


A Search-based Solution

- Goal: Make no assumptions and find near deadline misses as well, identify worst case scenarios
- 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

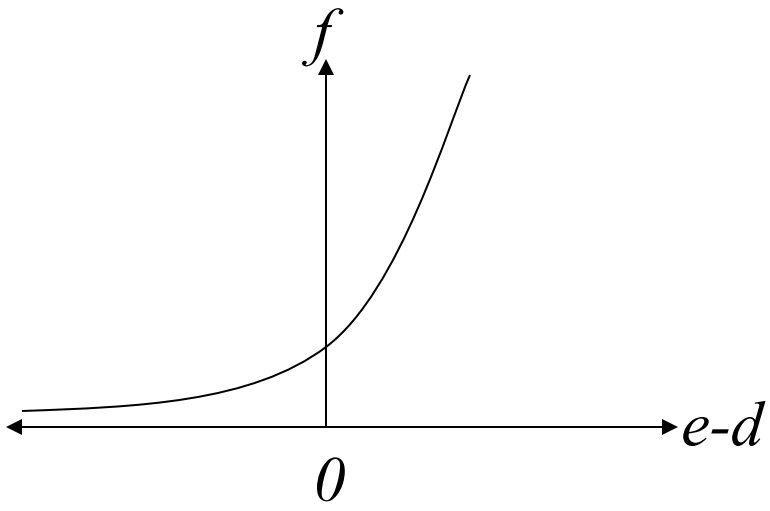


Objective Function

- 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

	Number of Misses	*Value of Misses*
Weapon Release	*0*	*N/A*
Weapon Release Subtask	*0*	*N/A*
Radar Tracking Filter	*0*	*N/A*
RWR Contact Management	*2*	*3, 9*
Data Bus Poll Device	*0*	*N/A*
Weapon Aiming	*0*	*N/A*
Radar Target Update	*4*	*17, 16, 10, 9*
Navigation Update	*7*	*1, 29, 23, 2, 28, 27, 32*

Conclusions

- We devised a method to generate event seeding times for aperiodic tasks so as identifying deadline miss scenarios based on task 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 on regular PC
- 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)

Testing Driven by Environment Modeling

Z. Iqbal, A. Arcuri, L. Briand, 2009-2012

Context

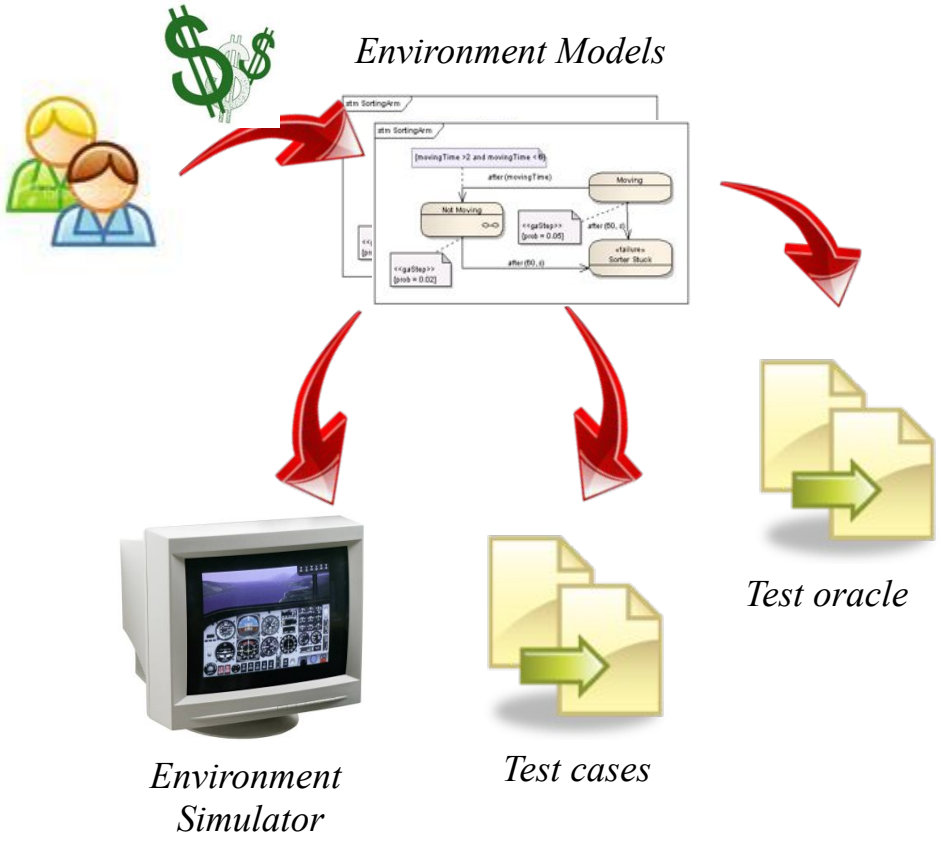
- Three-year project with two industry partners
  - Soft real-time systems: deadlines in order of hundreds of milliseconds
    - Jitter of few milliseconds acceptable
  - *Automation of test cases and oracle generation*, environment simulation

Tomra – Bottle Recycling Machine

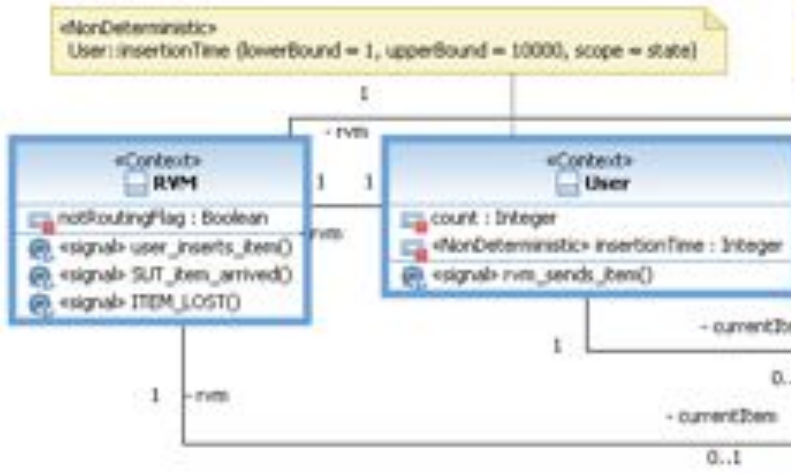
WesternGeco – Marine Seismic Acquisition System

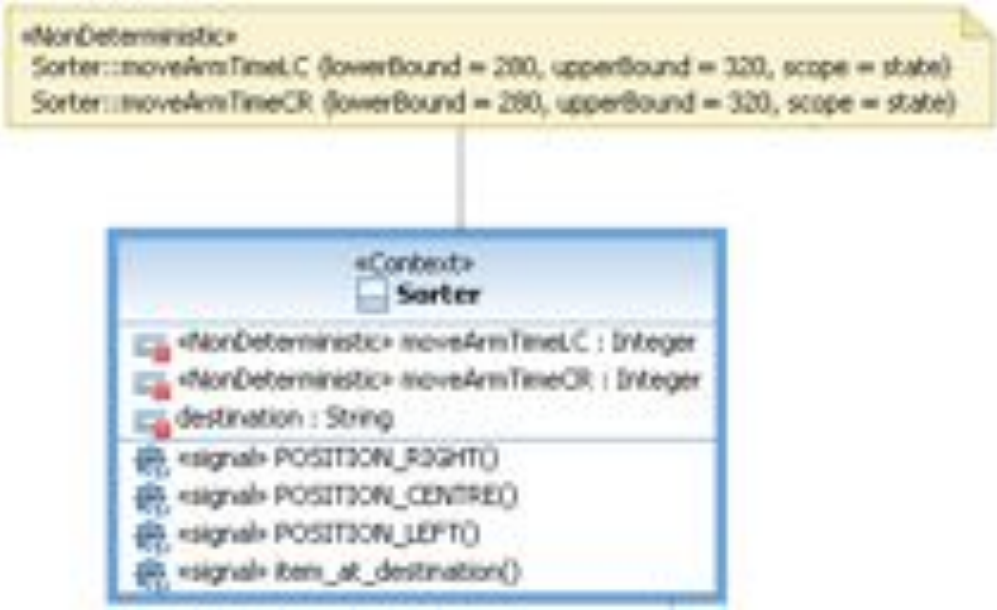
Environment Modeling and Simulation

- 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)

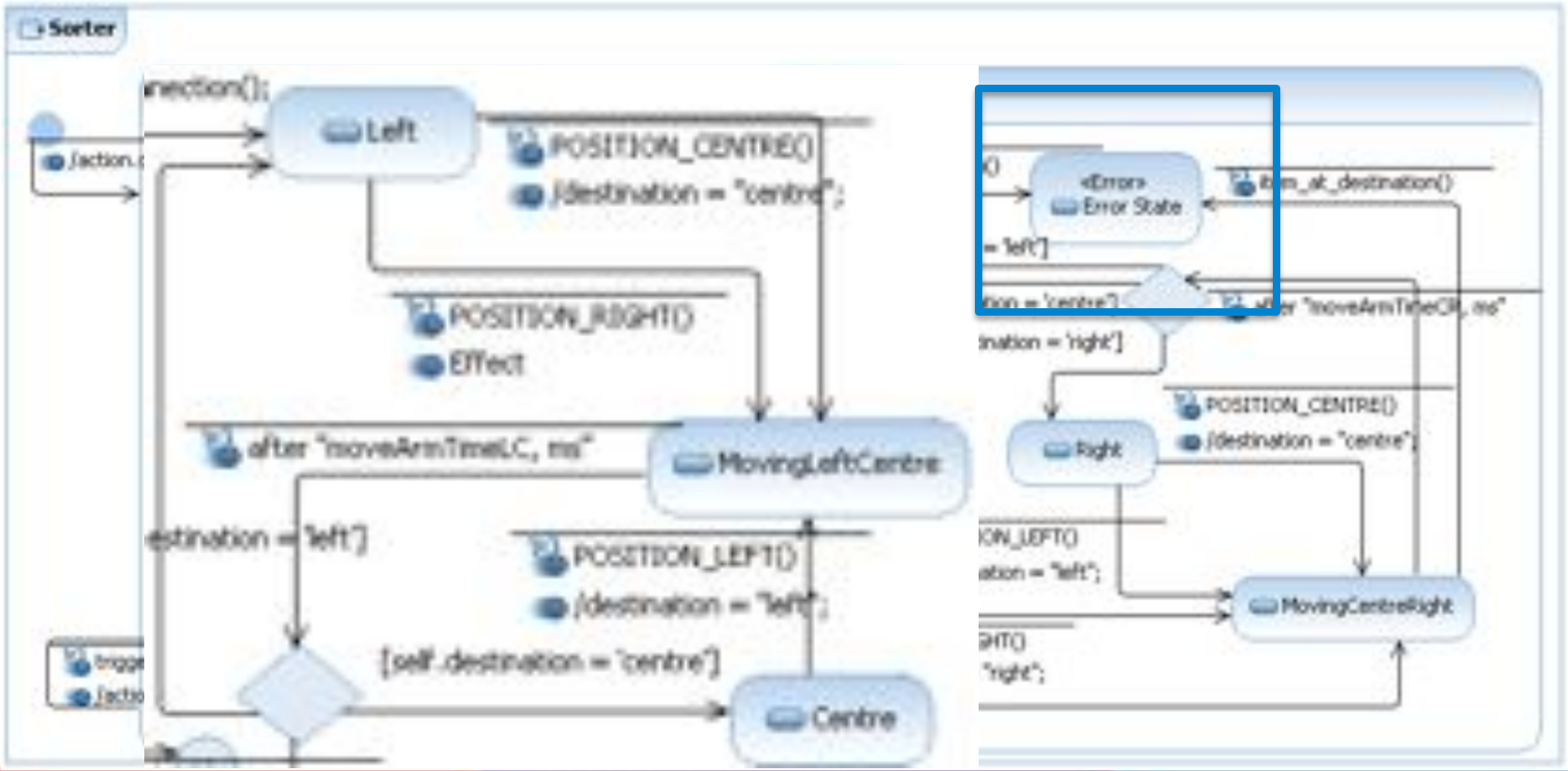


Domain Model





Behavior Model



Test Cases

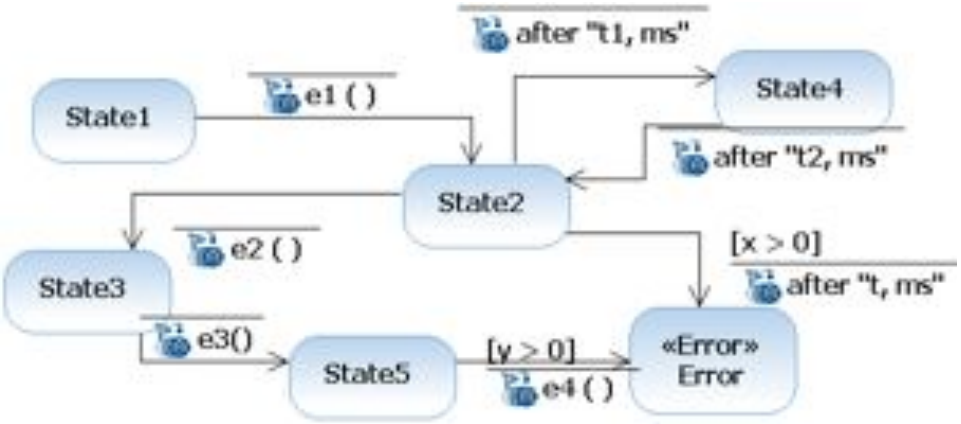
- Test cases are defined by
  - Simulation configuration
  - Environment configuration
- Environment Configuration
  - Number of instances to be created for each component in the domain model (e.g., the number of sensors)
- Simulator Configuration
  - Setting of non-deterministic attribute values
- Test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

Search Objectives and Heuristics

- Bring the system state to an error state by searching 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
- Different metaheuristics: Genetic algorithm, (1+1) EA
- Defining the fitness function based on model information was highly complex: OCL constraints, combination of many heuristics
- Industrial case study and artificial examples showed the heuristic was effective
  - (1+1) EA better than GA

Basic Ideas about the Fitness Function

- Evaluates how “good” the simulator configurations are
- Can only be decided after the execution of a test case
- Decided based on heuristics: How close was the test case to …
  - Approach Level
    - reach an error state?
  - Branch Distance
    - *s*olve the guard on a branch leading to an error state?
    - *d*efined search heuristics for OCL expressions*
  - Time Distance
    - *t*ake a time transition that leads to an error state?

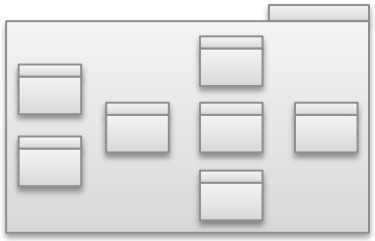


** S. Ali, M.Z. Iqbal, A. Arcuri, L. Briand, "Generating Test Data from OCL Constraints with Search Techniques", forthcoming in IEEE Transactions on Software Engineering*

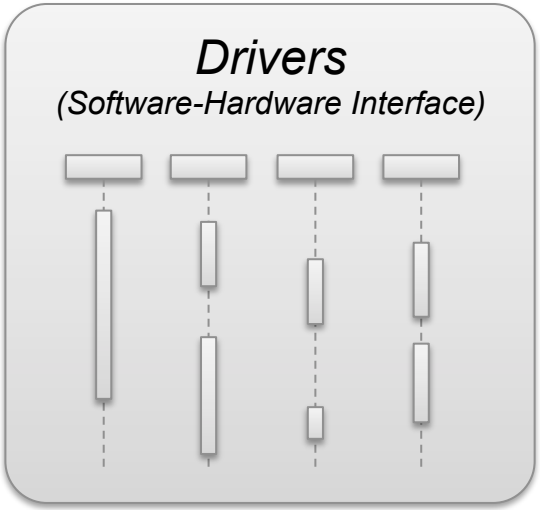
Constraint Optimization to Verify CPU Usage

S. Nejati, S. Di Alesio, M. Sabetzadeh, L. Briand, 2012

System: fire/gas detection and emergency shutdown



Control Modules



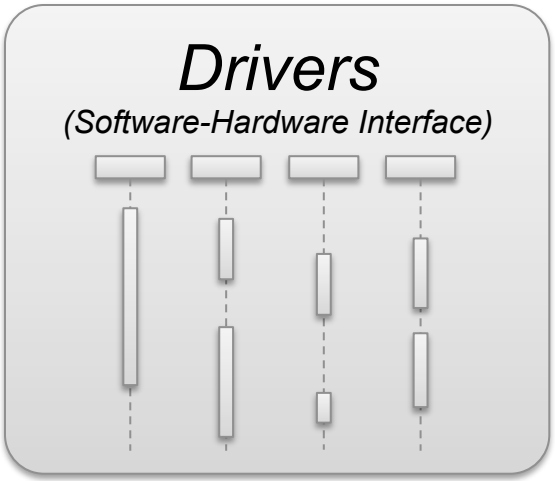
Alarm Devices (Hardware)

Real Time Operating System

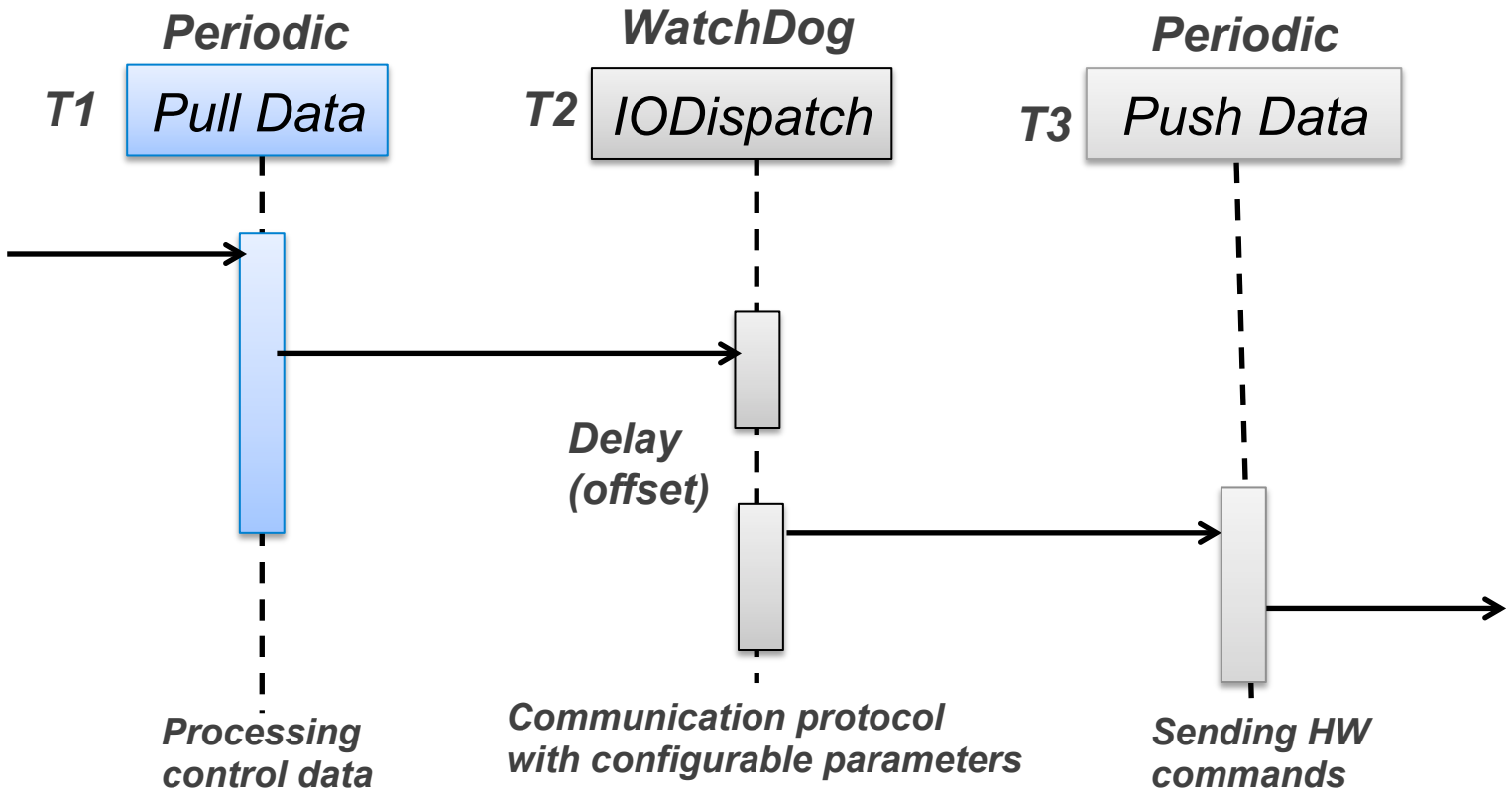
Multicore Archt.

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
  - Period and watchdog threads
- 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”
- Determine how many driver instances to deploy on a CPU



I/O Driver



Small Delay – – ▸ T2 consumes a lot of CPU time – – ▸ CPU overload

Large Delay – – ▸ T2 may blockT1 and/or T3 – → Deadline misses

Safety standards

IEC 61508 is a Safety Standard including guidelines for Performance Testing

Table B.6 – Performance testing
(referenced by tables A.5 and A.6)

	Technique/Measure*	Ref	SIL1	SIL2	SIL3	SIL4
1	Avalanche/stress testing	C.5.21	R	R	HR	HR
2	Response timings and memory constraints	C.5.22	HR	HR	HR	HR
3	Performance requirements	C.5.19	HR	HR	HR	HR
* Appropriate techniques/measures shall be selected according to the safety integrity level.						

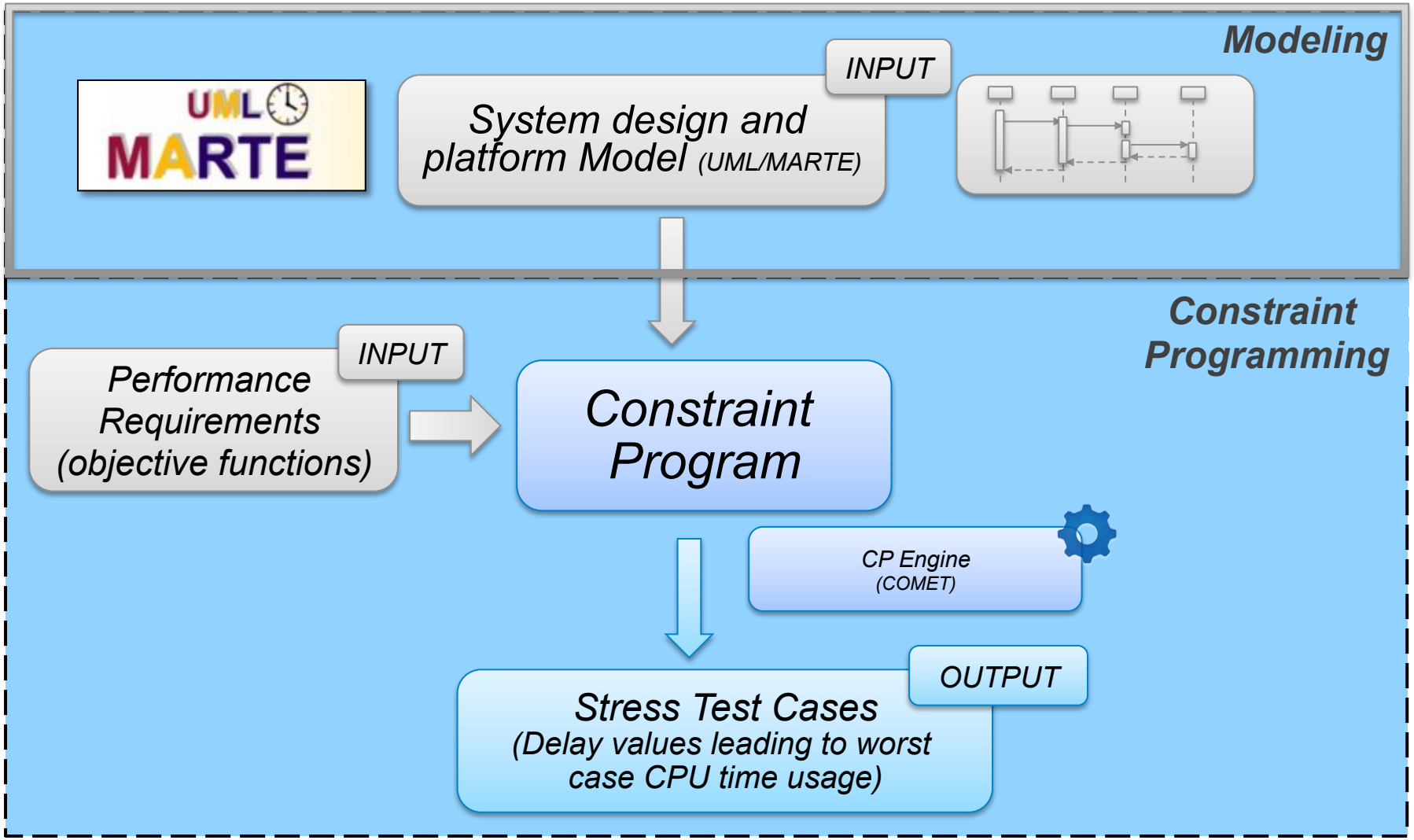
iec.ch

To achieve SIL levels 3-4, Stress Testing is "Highly Recommended"

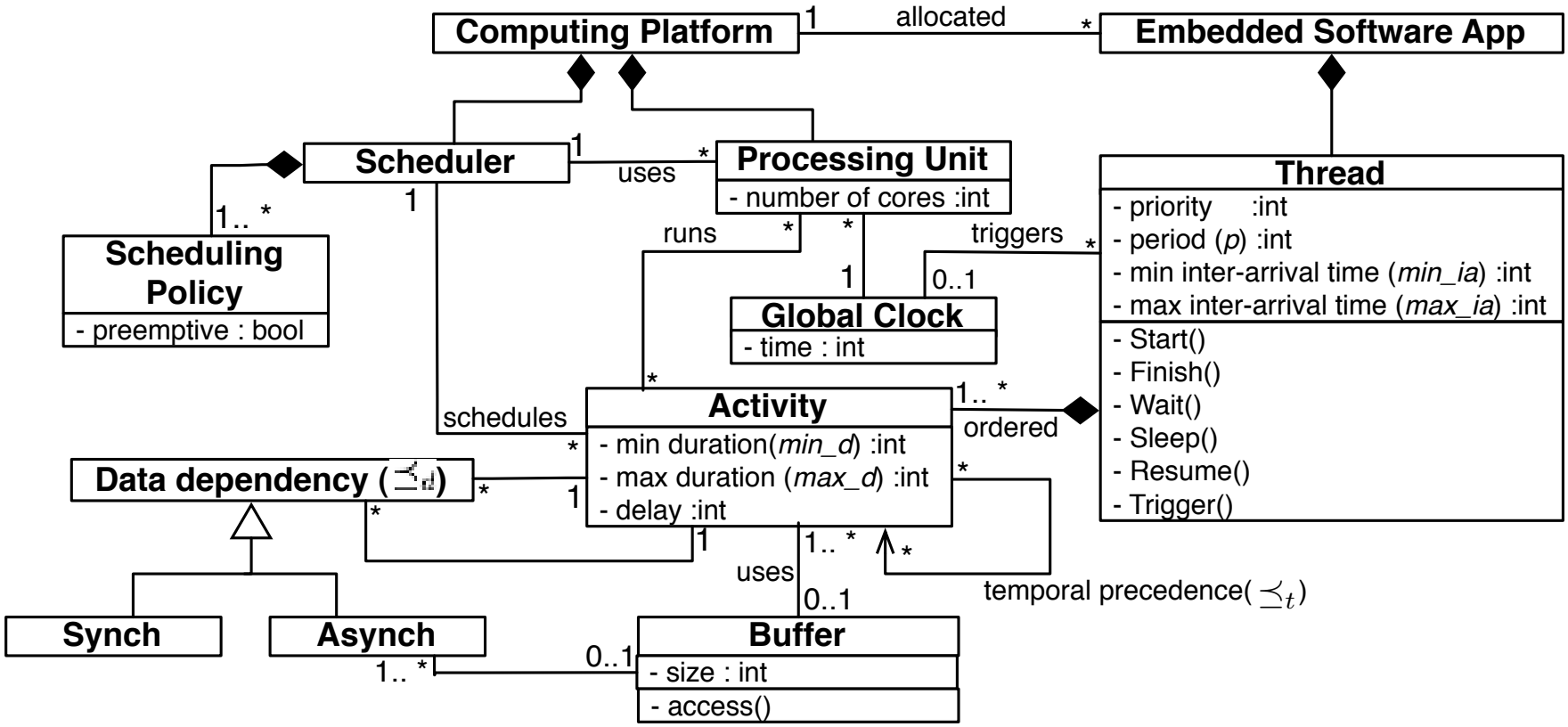
Search Objective

- *Stress test cases:* Values for *environment-dependent parameters* of the embedded software, e.g., the size of time *delays* used in software to synchronize with hardware devices or to receive feedback from the hardware devices.
- *Goal: select delays to* maximize the use of the CPU while satisfying design constraints.

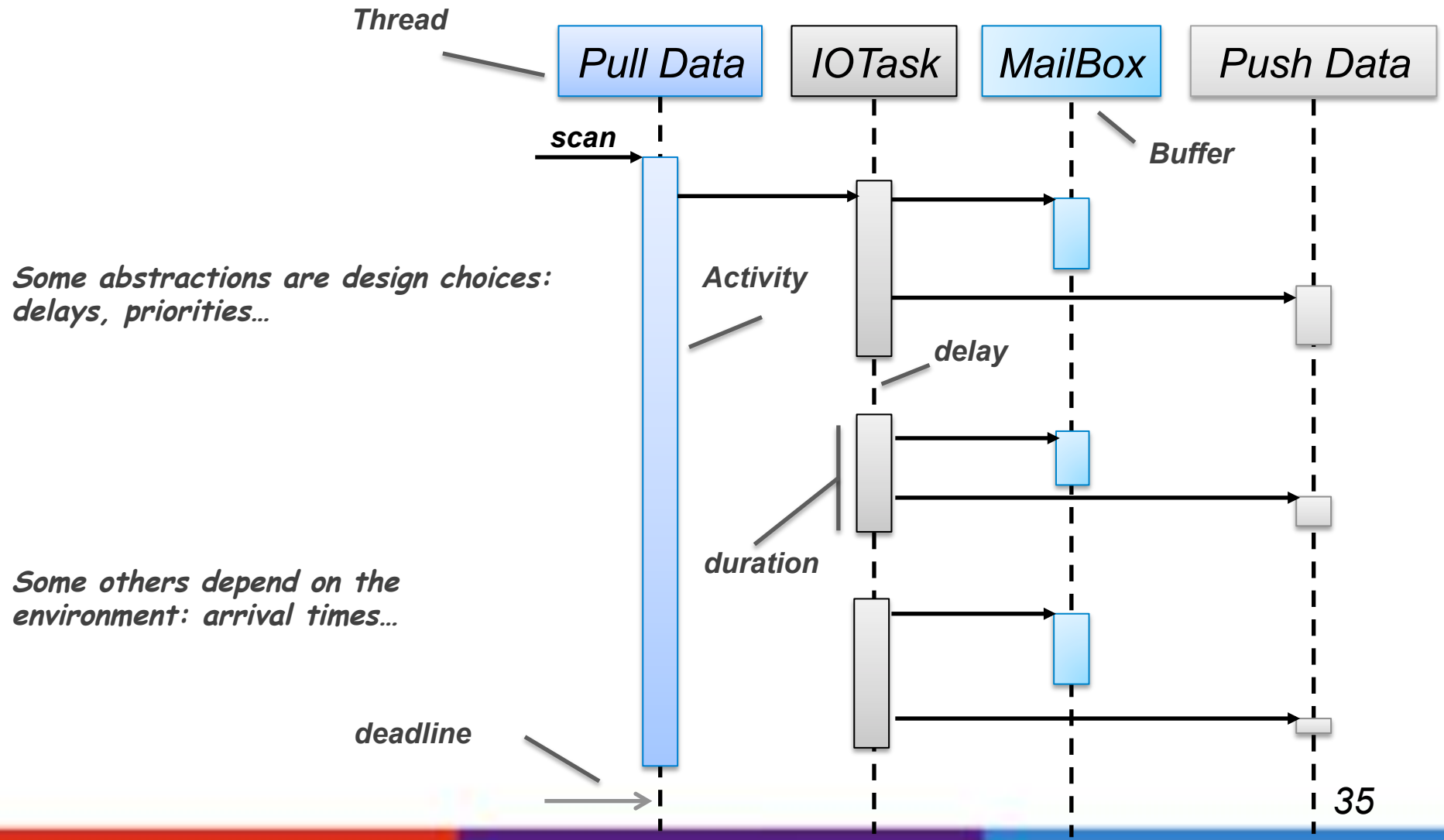
General Approach: Modeling and Optimization



Information Requirements



MARTE: Augmented Sequence Diagrams



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 (e.g., delays), 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 focus on objective functions for CPU Usage here

Each objective function models a specific performance requirement

Testing a different performance requirements only requires to change the objective function (constraints)

```
// 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 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

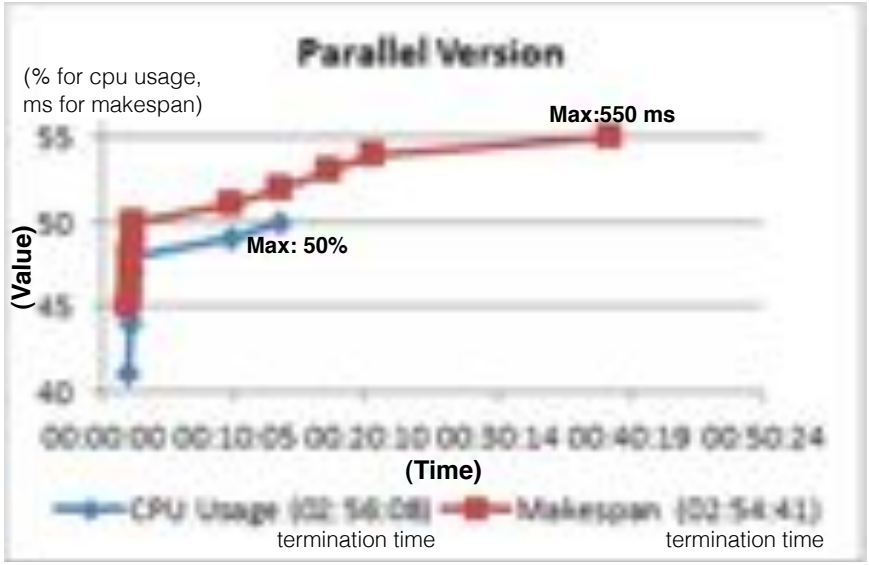
```
// 1) Input: Time and Concurrency information
int c = ...; // #Cores
int n = ...; range J = 0..n-1; // #Threads
int priority[J] = ...; // Priorities
// ...

// 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
                    // ...
```

Case Study (Driver)

We run an experiment with real data and two different objective functions

It took a significant amount of time for the search to terminate



But optimal solutions were found shortly after the search started, even if the search took a much more time 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
- Currently continues this work with Delphi using metaheuristic search: 430 tasks, powertrain systems, AUTOSAR

Testing Closed Loop Controllers

R. Matinnejad, S. Nejati, L. Briand, T. Bruckmann, C. Poull, 2013

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

Comfort and variety

More functions

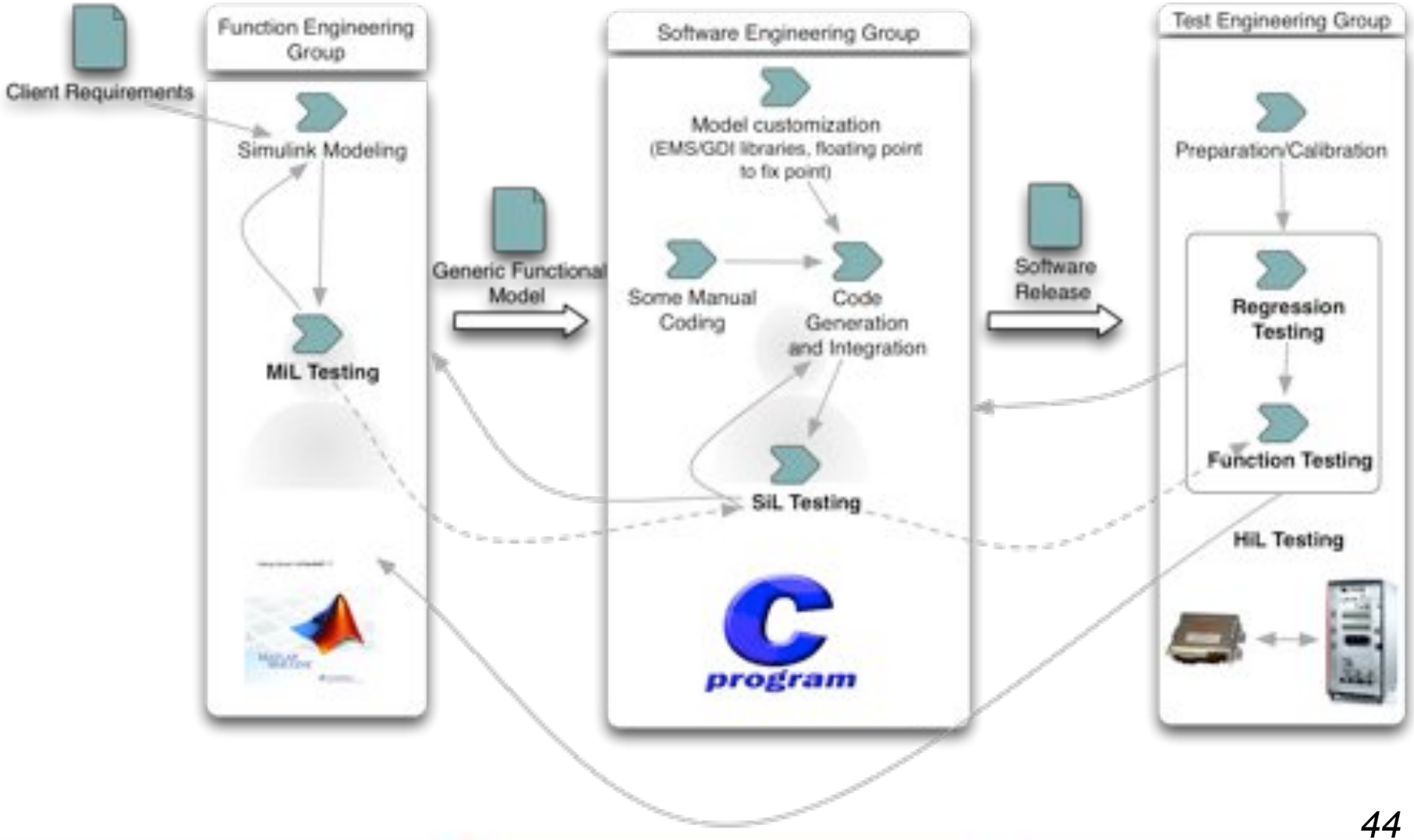
Safety and reliability

Faster time-to-market

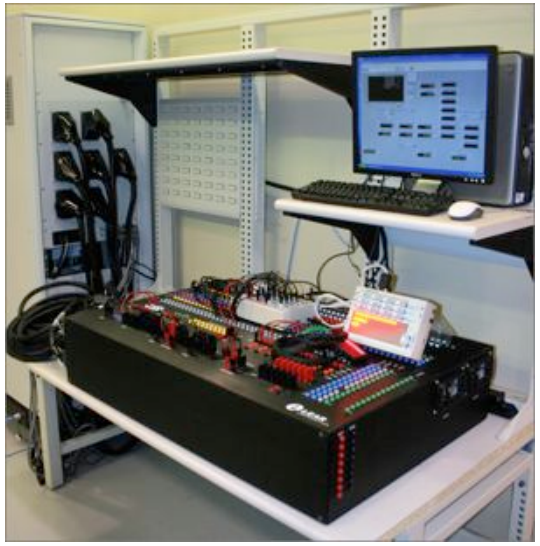
Greenhouse gas emission laws

Less fuel consumption

Three major software development stages in the automotive domain



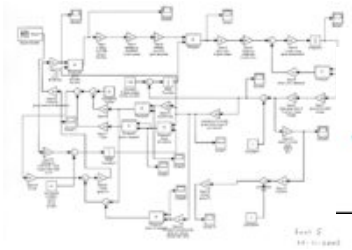
Major Challenges in MiL-SiL-HiL Testing



- 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

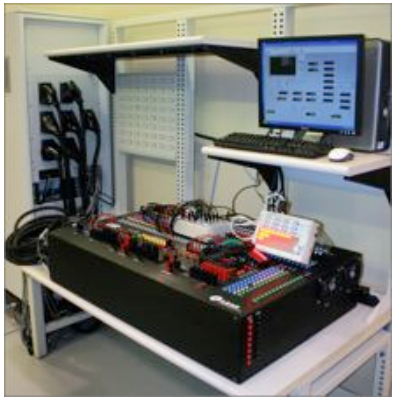
MiL testing

Requirements



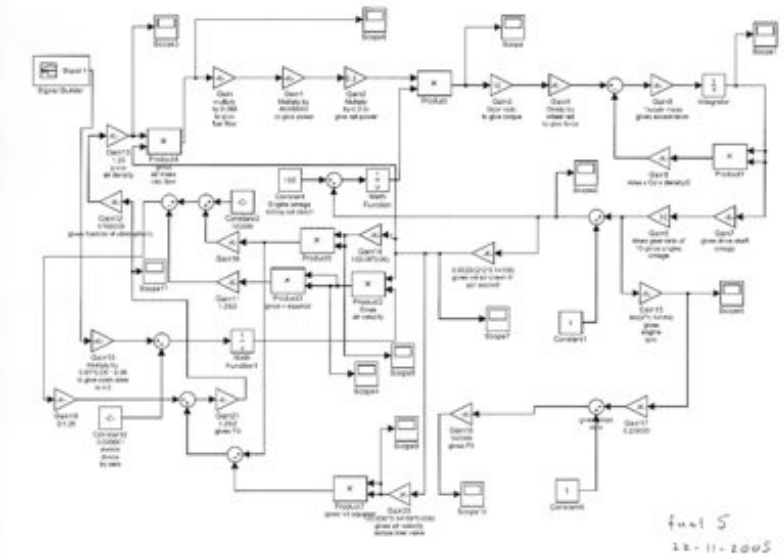
MATLAB
SIMULINK

Individual Functions



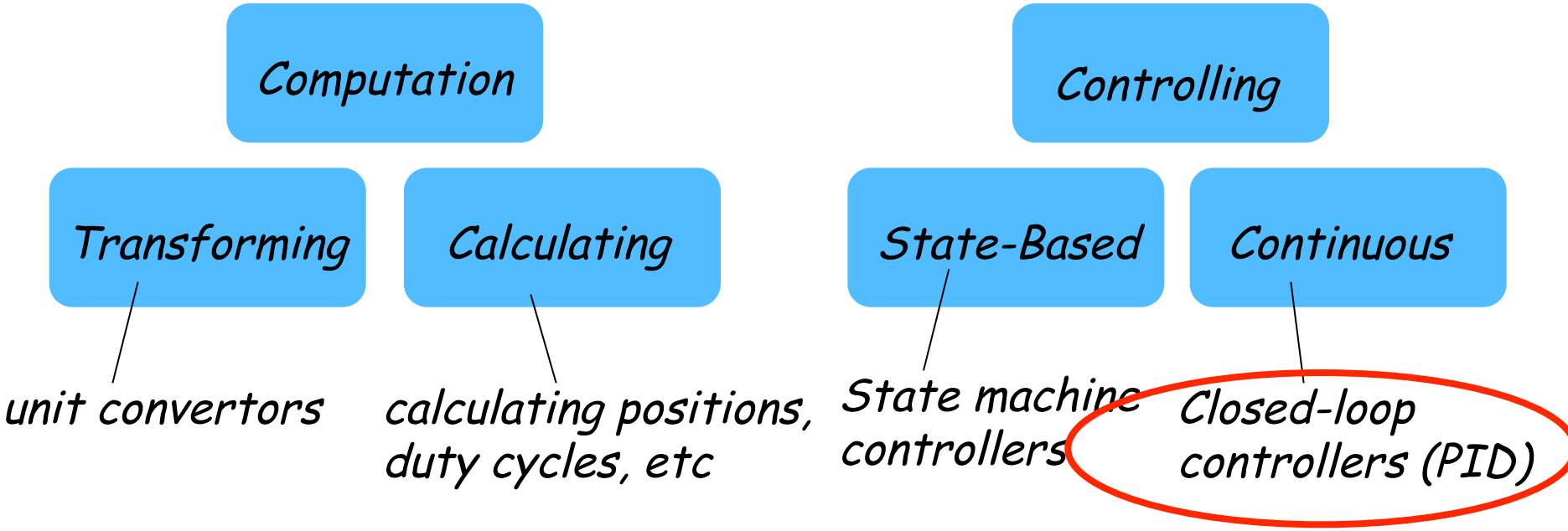
The ultimate goal of MiL testing is to ensure that individual functions behave correctly and timely on any hardware configuration

Main differences between automotive function testing and general software testing



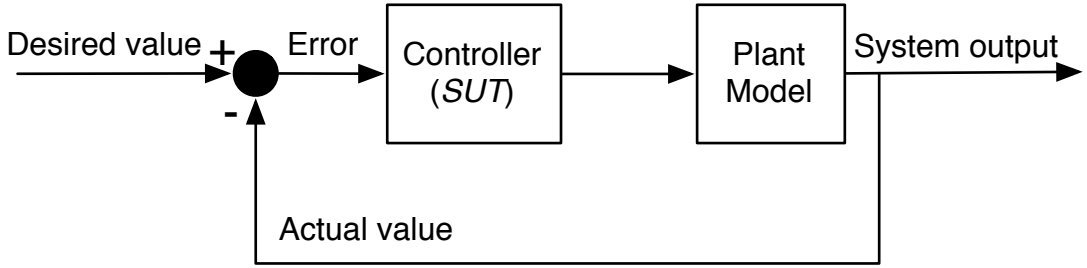
- Continuous behaviour
- Time matters a lot
- Several configurations
  - Huge number of detailed physical measures/ranges/ thresholds captured by calibration values

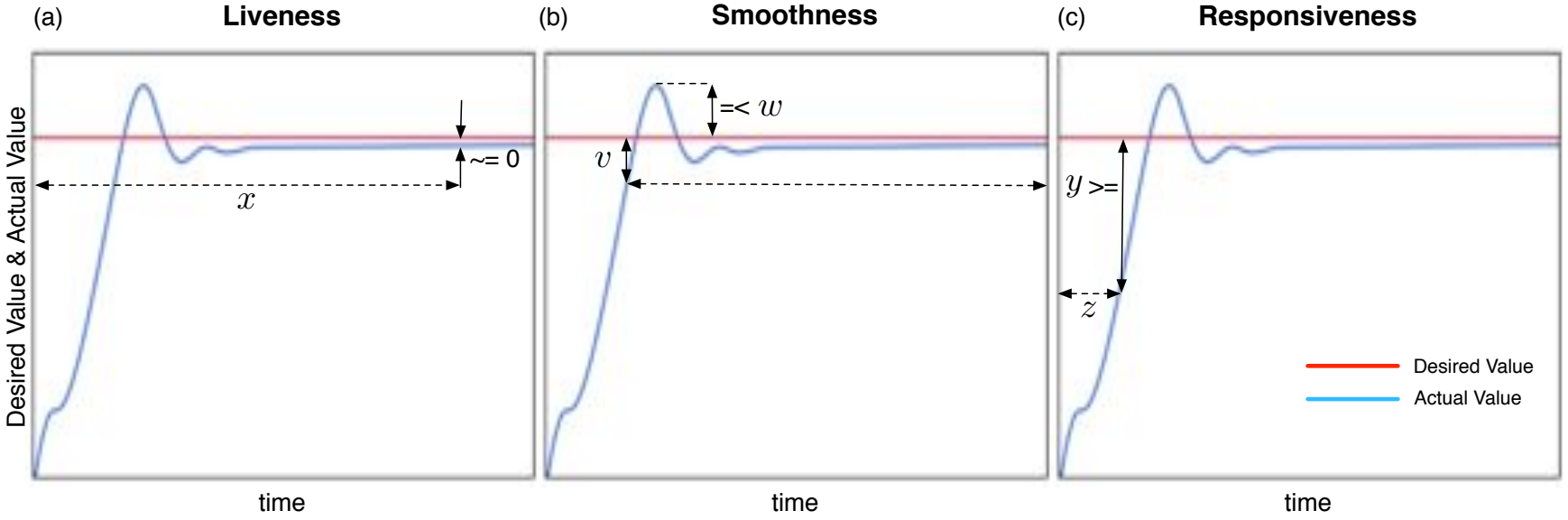
A Taxonomy of Automotive Functions



Different testing strategies are required for different types of functions

Controller Plant Model and its Requirements





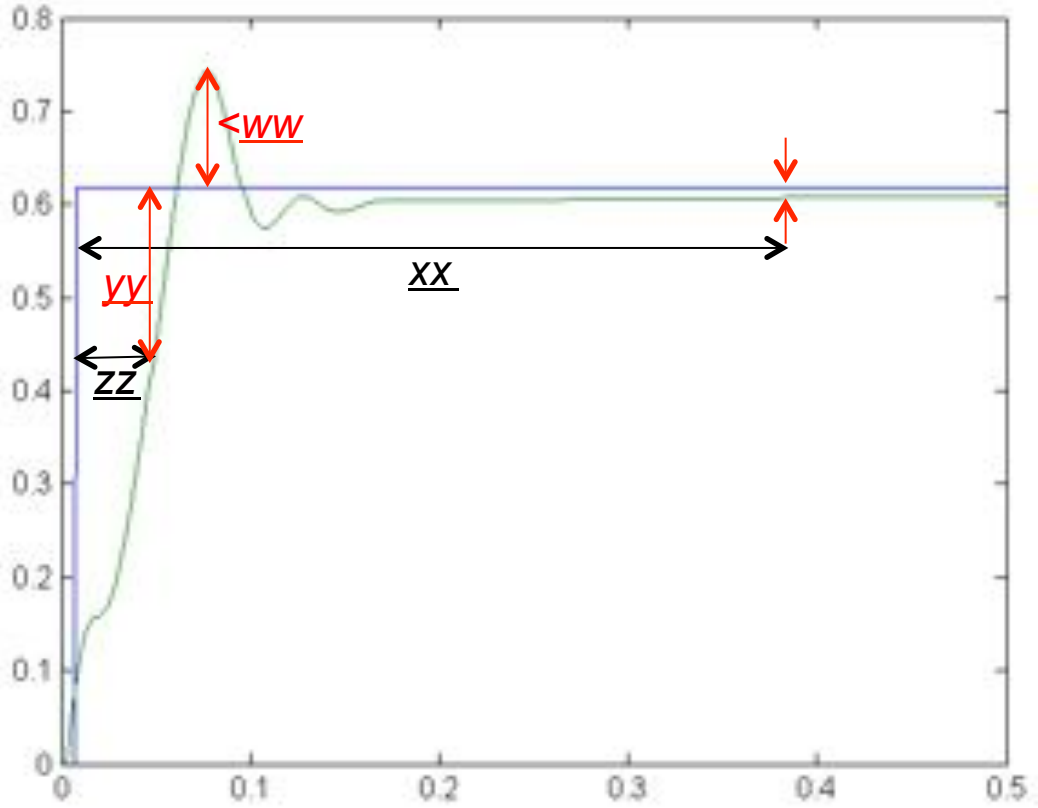
Types of Requirements

(1) functional/liveness

(2) responsiveness/performance

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.

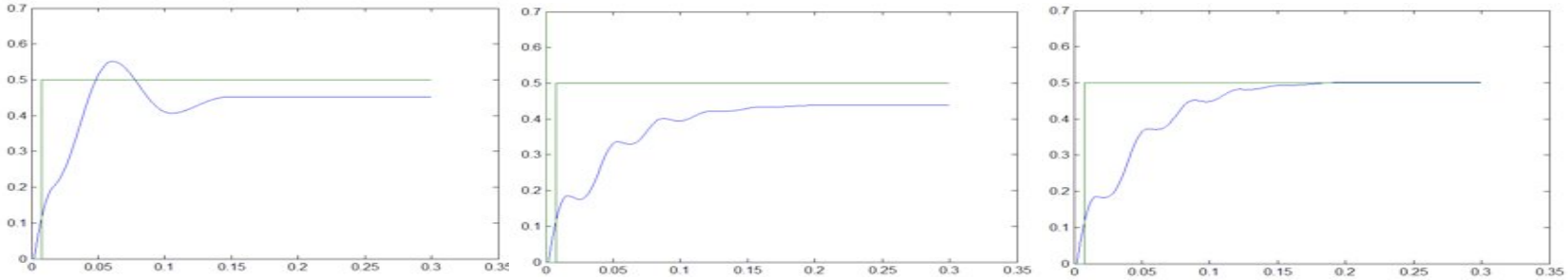
(3) smoothness/safety



Search Elements

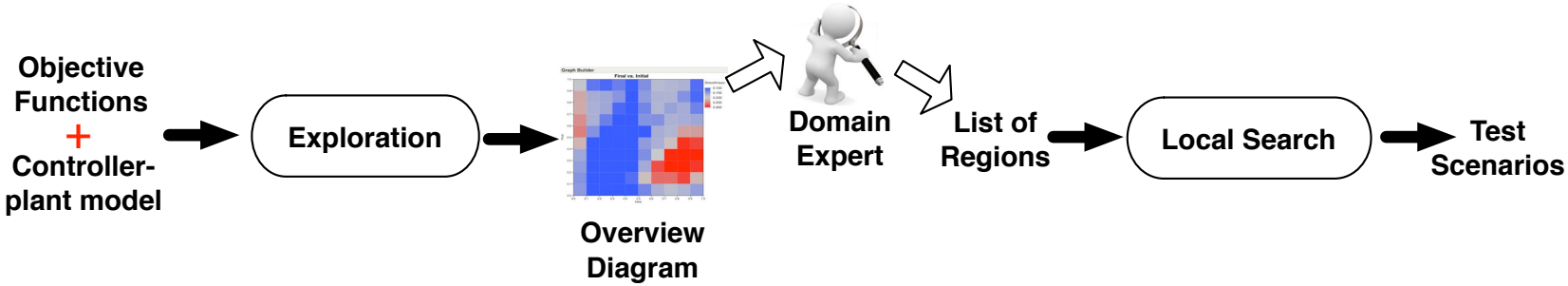
- **Search:**
  - Inputs: Initial and desired values, configuration parameters
  - (1+1) EA
- **Search Objective:**
  - Example requirement that we want to test: liveness
    - ✓ |Desired - Actual(final)|~= 0

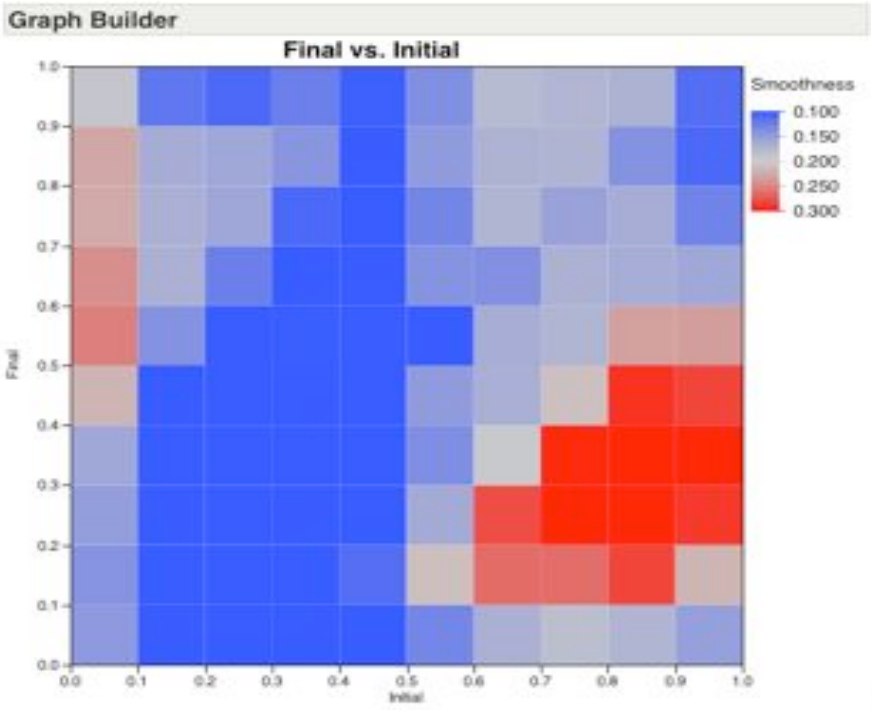
For each set of inputs, we evaluate the objective function over the resulting simulation graphs:

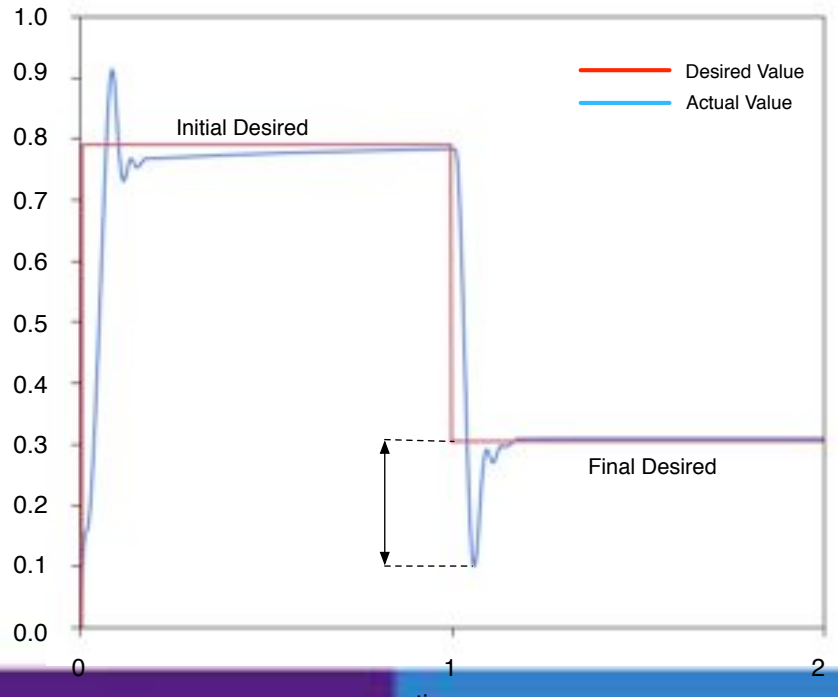


- **Result:**
  - worst case scenarios or values to the input variables that are more likely to break the requirement at MiL level
  - stress test cases based on actual hardware (HiL)

MiL-Testing of Continuous Controllers

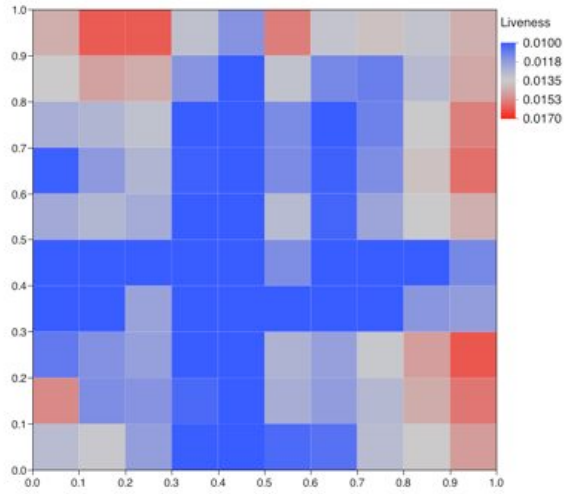




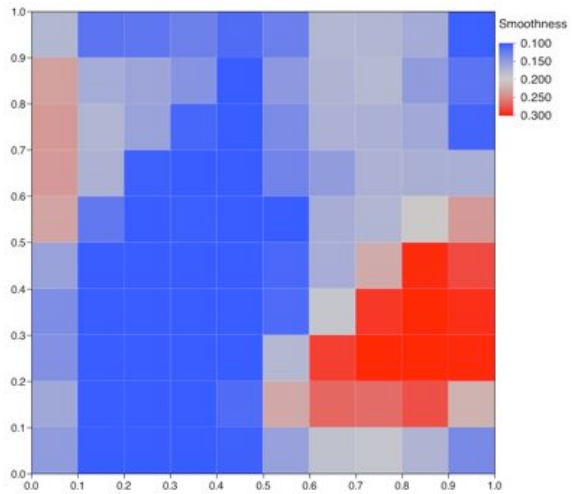


Generated Heatmap Diagrams

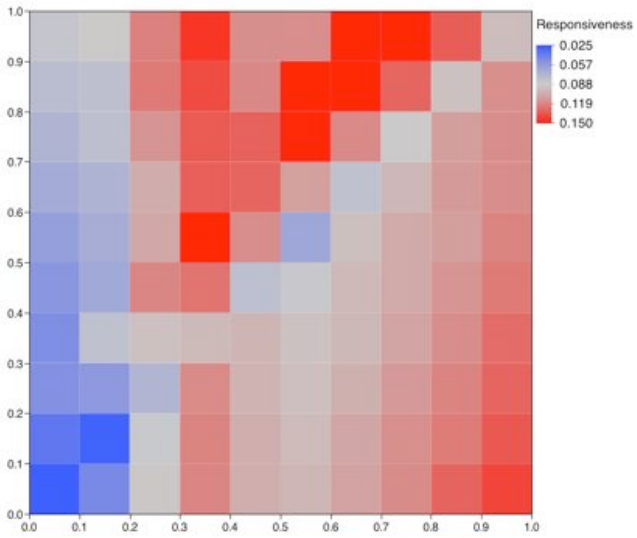
(a) Liveness



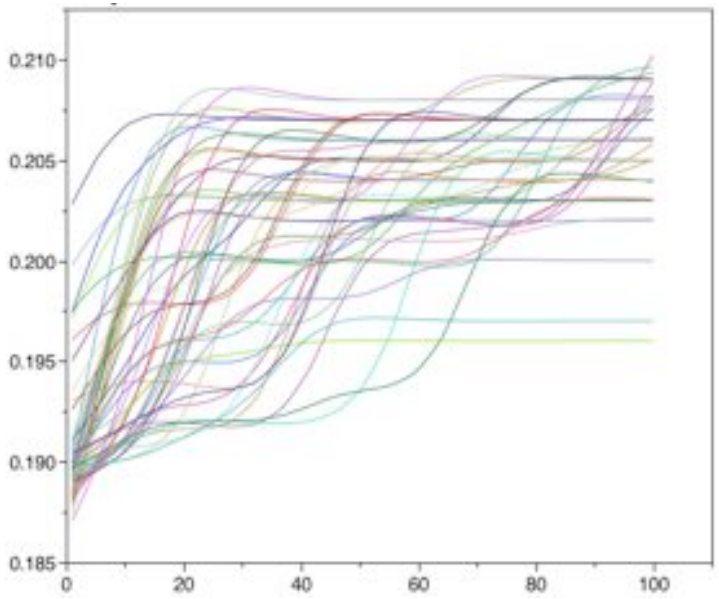
(b) Smoothness



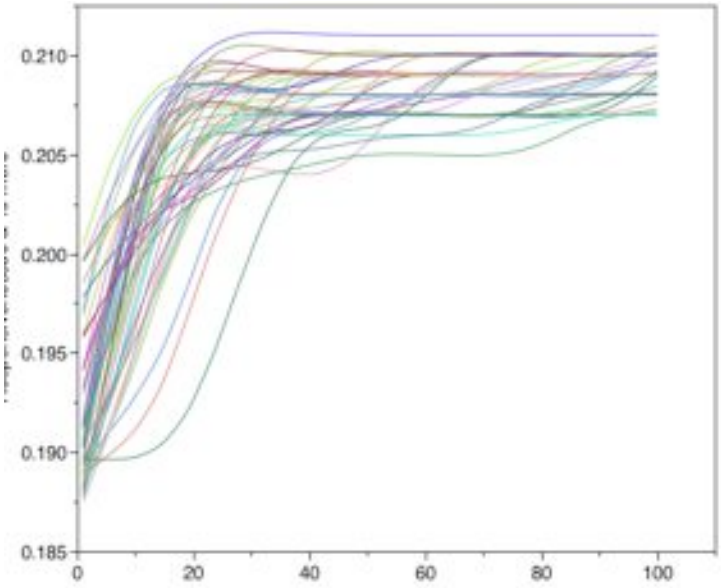
(c) Responsiveness



Random Search vs. (1+1)EA
Example with Responsiveness Analysis



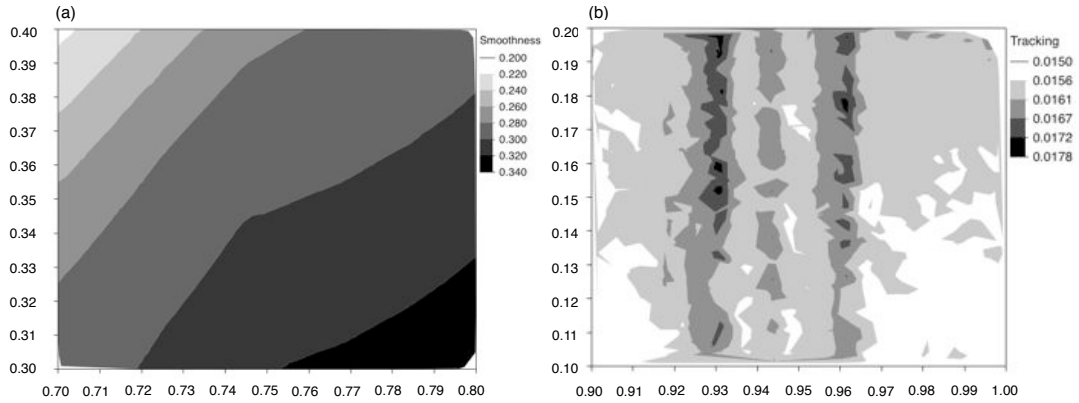
Random



(1+1) EA

Conclusions

- We found much worse scenarios during MiL testing than our partner had found so far
- They are running them at the HiL level, where testing is much more expensive: MiL results -> test selection for HiL
- On average, the results of the single-state search showed significant improvements over the result of the exploration algorithm
- Configuration parameters?
- Need more exploitative or explorative search algorithms in different subregions



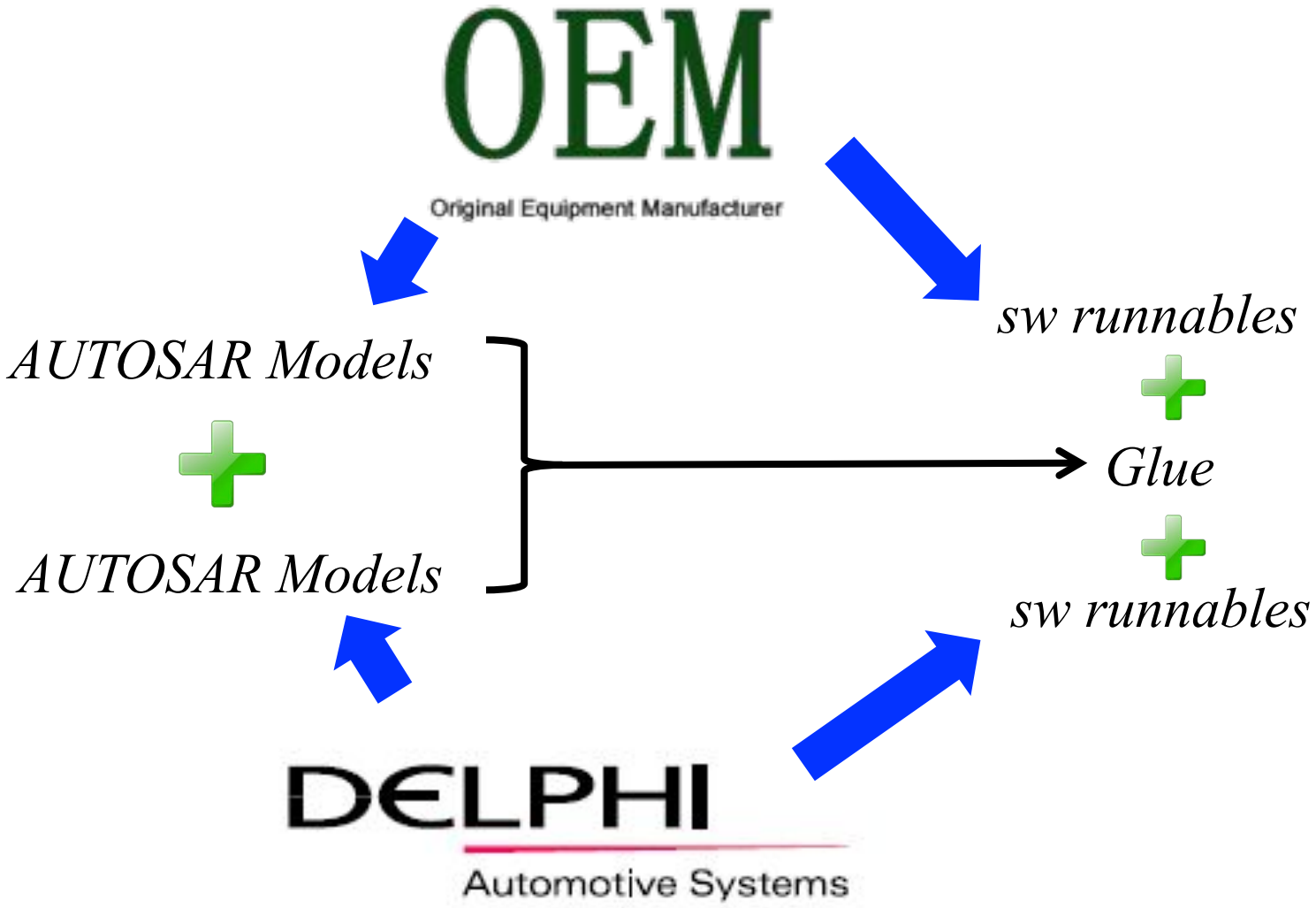
Minimizing CPU Time Shortage Risks in Integrated Embedded Software

S. Nejati, M. Adedjouma
L. Briand, T. Bruckmann, C. Poull, 2013

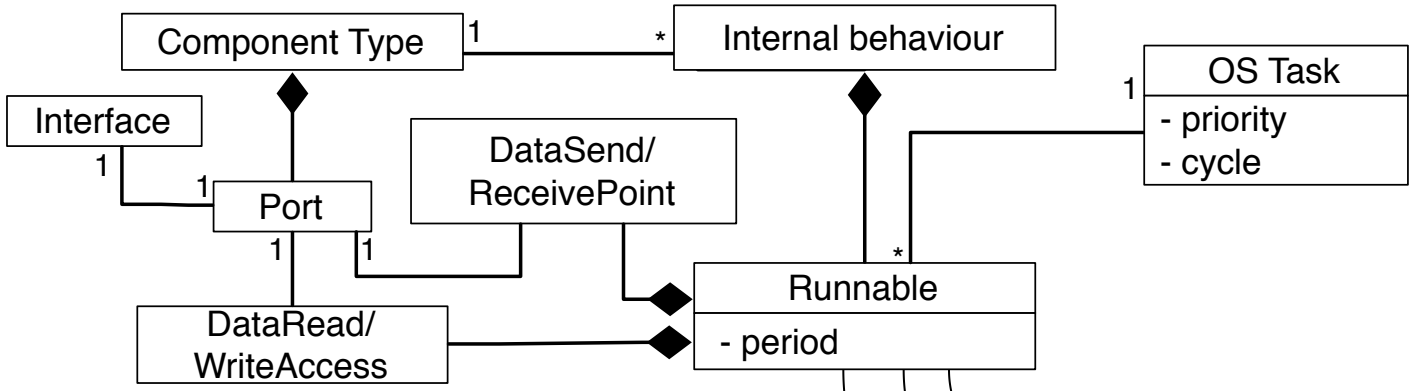
Today's cars rely on integrated systems

- Modular and independent development
- Huge opportunities for division of labor and outsourcing
- Need for reliable and effective integration processes

An overview of an integration process in the automotive domain



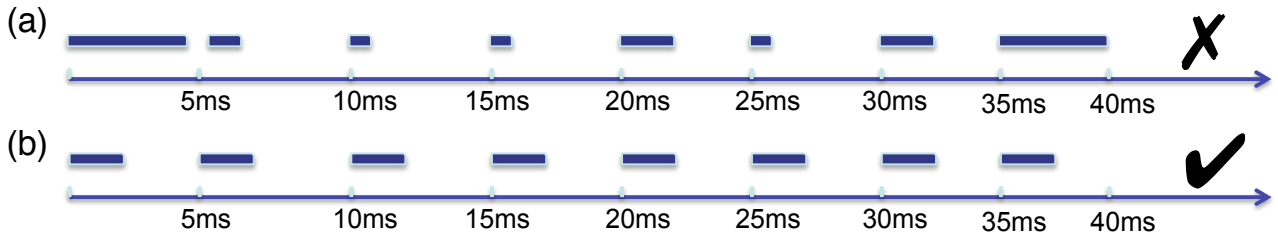
AUTOSAR captures the timing properties of the Runnables and their data dependencies



Runnables Glue Code:

```
Task_o1() { /* cycle_o1 = 5 ms */
  if ( (timer mod r1.period) == 0) do /* timer mod 10 == 0 */
    execute runnable r1
  if ( (timer mod r2.period) == 0) do /* timer mod 20 == 0 */
    execute runnable r2
  if ( (timer mod r3.period) == 0) do /* timer mod 100 == 0 */
    execute runnable r3
  ...
}
Task_o2() {
  ...
```

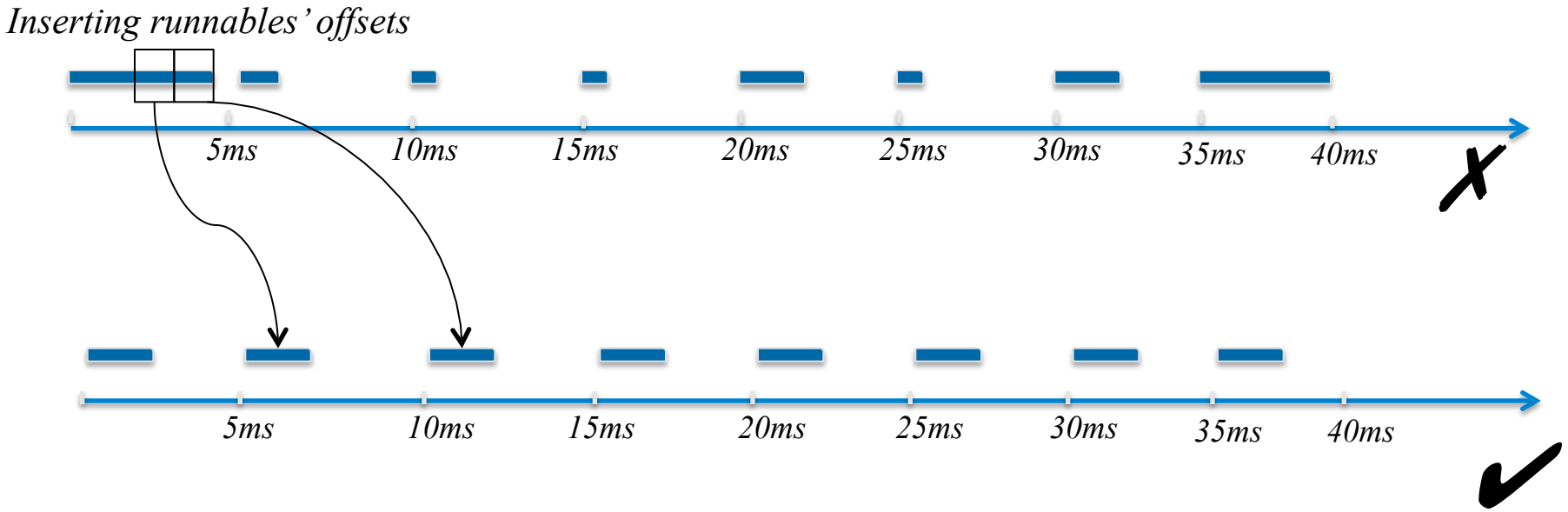
CPU Time Shortage in Integrated Embedded Software



- *Challenge*
  - *Many OS tasks and their many runnables run within a limited available CPU time*
    - *The execution time of the runnables may exceed the OS cycles*

- *Our goal*
  - *Reducing the maximum CPU time used per time slot to be able to*
    - *Minimize the hardware cost*
    - *Enable addition of new functions incrementally*
    - *Reduce the possibility of overloading the CPU in practice*

We minimize the maximum CPU usage using runnables 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 the OS cycles

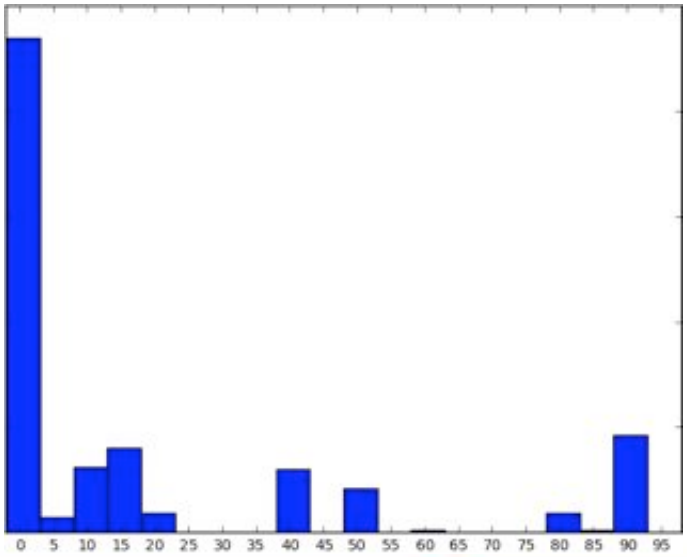
the runnnables satisfy their synchronization constraints

Meta heuristic search algorithms

- 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)
- Used restart option to make them more explorative

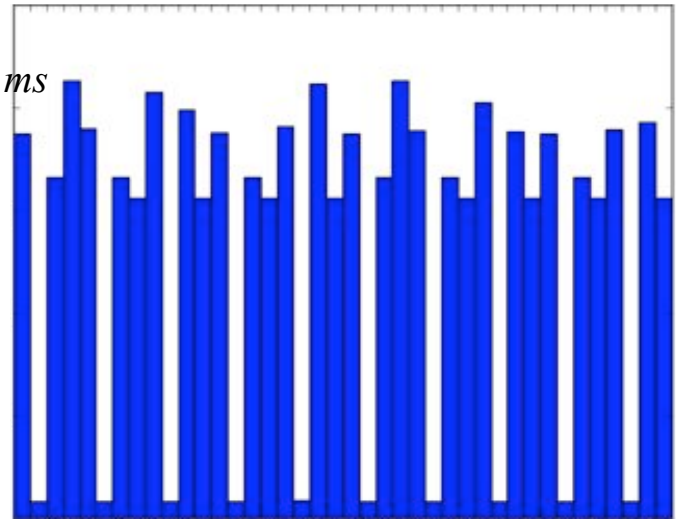
Case Study: an automotive software system with 430 runnables

5.34 ms



Running the system without offsets

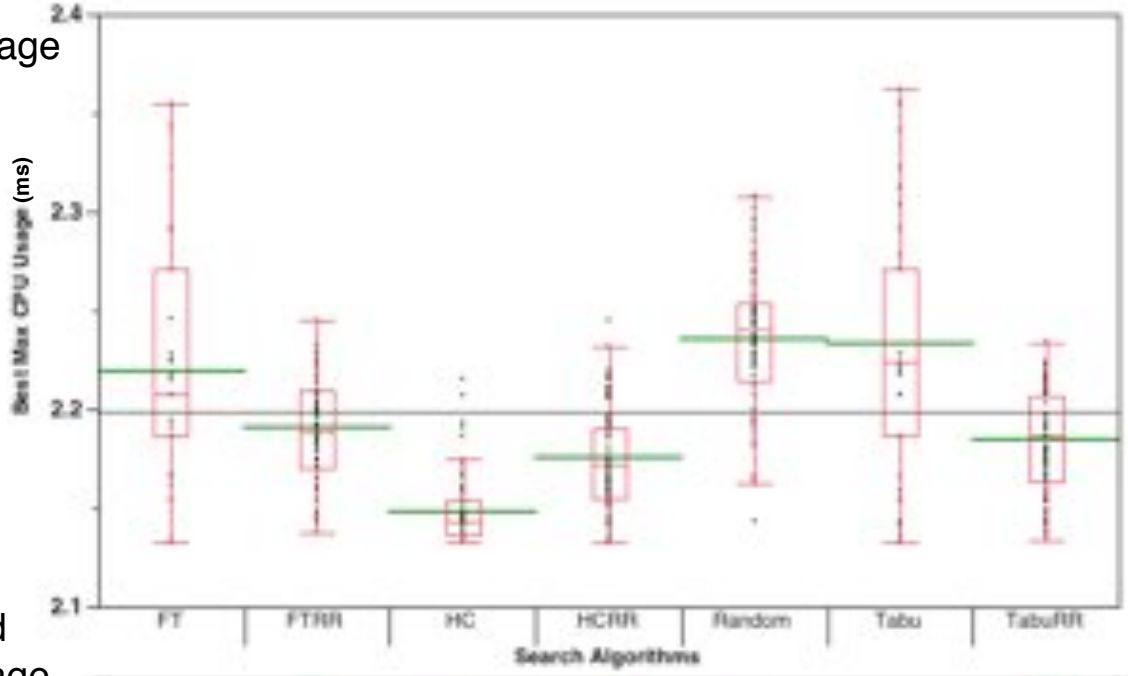
2.13 ms



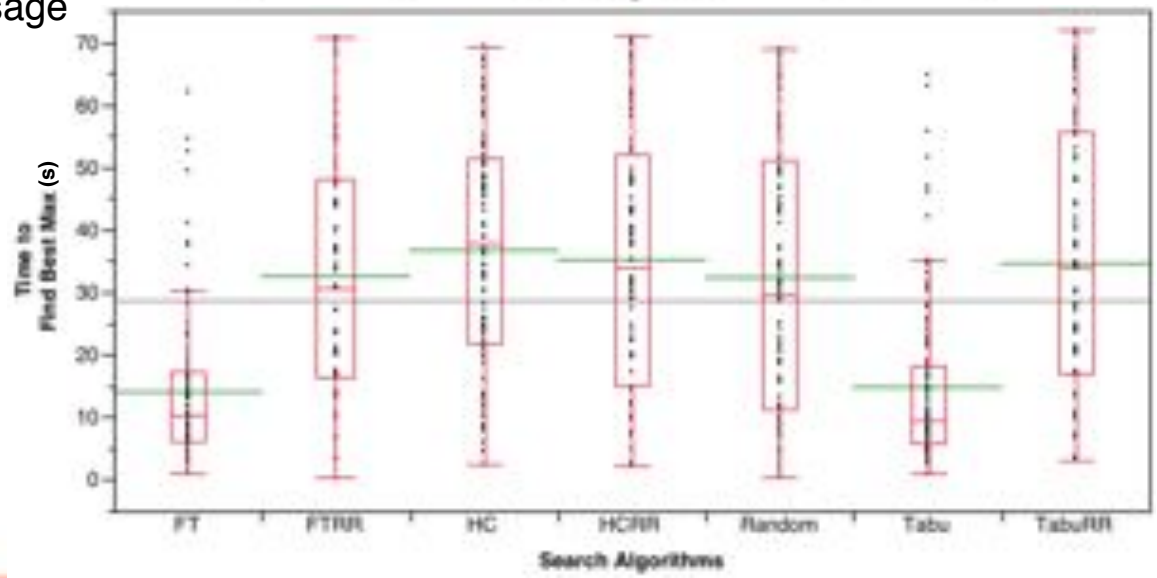
Our optimized offset assignment

Comparing different search algorithms

Best CPU usage

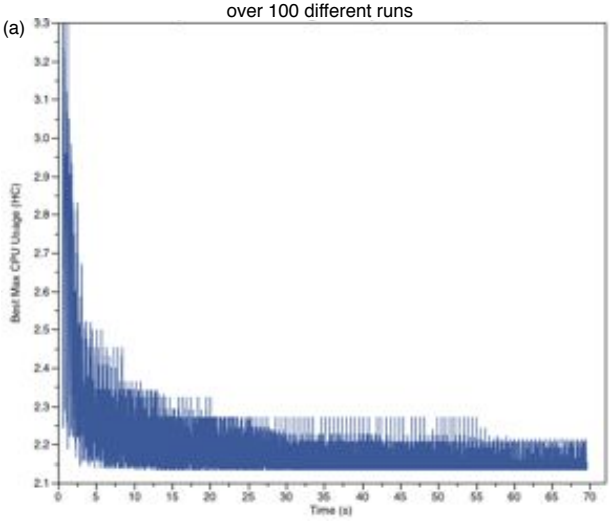


Time to find Best CPU usage

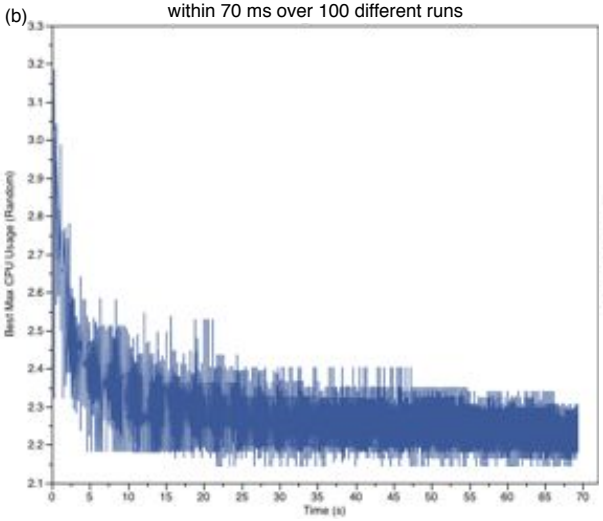


Comparing our best search algorithm with Random search

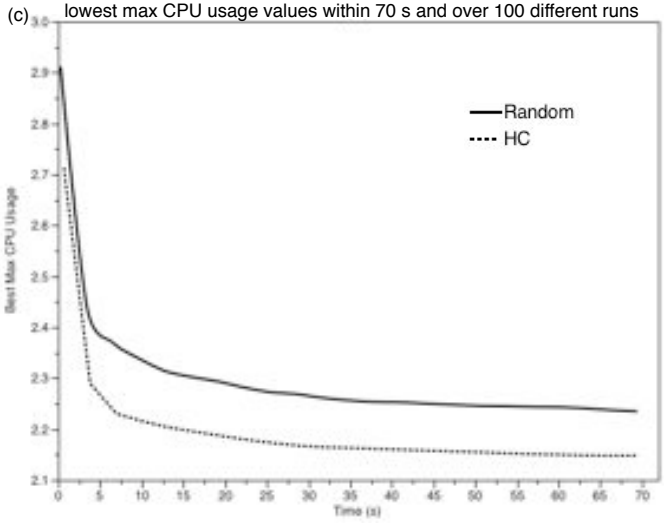
(a) Lowest max CPU usage values computed by HC within 70 ms over 100 different runs



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



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



Conclusions

- We developed a number of search algorithms to compute offset values that reduce the max CPU time needed

- Our evaluation shows that our approach is able to generate reasonably good results for a large automotive system and in a small amount of time

- Due to large number of runnables and the orders of magnitude difference in runnables periods and their execution times, we were not able to use constraint solvers

MDE Projects Overview (< 5 years)

Company	Domain	Objective	Notation	Automation
ABB	Robot controller	Testing	UML	Model traversal for coverage criteria
Cisco	Video conference	Testing (robustness)	UML profile	Metaheuristic
Kongsberg Maritime	Fire and gas safety control system	Certification	SysML + req traceability	Slicing algorithm
Kongsberg Maritime	Oil&gas, safety critical drivers	CPU usage analysis	UML+MARTE	Constraint Solver
FMC	Subsea system	Automated configuration	UML profile	Constraint solver
WesternGeco	Marine seismic acquisition	Testing	UML profile + MARTE	Metaheuristic
DNV	Marine and Energy, certification body	Compliance with safety standards	UML profile	Constraint verification
SES	Satellite operator	Testing	UML profile	Metaheuristic
Delphi	Automotive systems	Testing (safety +performance)	Matlab/Simulink	Metaheuristic
Lux. Tax department	Legal & financial	Legal Req. QA & testing	Under investigation	Under investigation

Conclusions I

- Models
  - UML profiles, MARTE, SysML, Matlab/Simulink, AUTOSAR
  - Never UML only: always some tailoring required
  - Always a specific modeling methodology, i.e., how to use the notation => semantics
  - Driven by
    - Information that is needed to guide automation, e.g., search, optimization
    - Appropriate level of abstraction: scalability, cost-effectiveness
    - Test/Verification objectives
    - Current modeling practices and skills
    - Reliance on standards
  - Modeling requirements must be “reasonable”, achievable, cost-effective for engineers

Conclusions II

- Automation
  - Many testing and verification problems can be re-expressed as a search/optimization problem
  - Search-based software engineering: Enabling automation for hard problems
  - But limited, though changing, applications to model-driven engineering
  - Models are needed to guide search and optimization
  - Many search techniques: Search algorithm, fitness function …
  - It is not easy to choose which one to use for a given problem
  - Empirical studies
  - Promising results, scalability (incomplete search)

Discussions

- 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 be found?
  - Better if problem can match 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
- Model checking
  - Detailed operational models (e.g., state models, CTL),, involving temporal properties (e.g., CTL)
  - Enough details to analyze statically or execute symbolically
  - These modeling requirements are usually not realistic in actual system development
  - Originally designed for checking temporal properties, as opposed to explicit timing properties

Discussions II

- Metaheuristic search
  - Tends to be more versatile, tailorable to a new problem
  - Entail lower modeling requirements
  - Can provide responses at any time, without systematically and deterministically searching the same part of the space at every run
  - “best solution found within time constraints”, not a proof, no certainty
  - But in practice (complex) models are not fully correct, there is no certainty

Selected References

- L. Briand, Y. Labiche, and M. Shousha, “Using genetic algorithms for early schedulability analysis and stress testing in real-time systems”, Genetic Programming and Evolvable Machines, vol. 7 no. 2, pp. 145-170, 2006
- M. Shousha, L. Briand, and Y. Labiche, “UML/MARTE Model Analysis Method for Uncovering Scenarios Leading to Starvation and Deadlocks in Concurrent Systems”, IEEE Transactions on Software Engineering 38(2), 2012.
- Z. Iqbal, A. Arcuri, L. Briand, “Empirical Investigation of Search Algorithms for Environment Model-Based Testing of Real-Time Embedded Software”, ACM ISSTA 2012
- S. Nejati, S. Di Alesio, M. Sabetzadeh, L. Briand, “Modeling and Analysis of CPU Usage in Safety-Critical Embedded Systems to Support Stress Testing”, ACM/IEEE MODELS 2012
- Shiva Nejati, Mehrdad Sabetzadeh, Davide Falessi, Lionel C. Briand, Thierry Coq, “A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies”, Information & Software Technology 54(6): 569-590 (2012)
- Lionel Briand et al., “Traceability and SysML Design Slices to Support Safety Inspections: A Controlled Experiment”, forthcoming in ACM Transactions on Software Engineering and Methodology, 2013

Selected References (cont.)

- Rajwinder Kaur Panesar-Walawege, Mehrdad Sabetzadeh, Lionel C. Briand: Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation. Information & Software Technology 55(5): 836-864 (2013)
- Razieh Behjati, Tao Yue, Lionel C. Briand, Bran Selic: SimPL: A product-line modeling methodology for families of integrated control systems. Information & Software Technology 55(3): 607-629 (2013)
- Hadi Hemmati, Andrea Arcuri, Lionel C. Briand: Achieving scalable model-based testing through test case diversity. ACM Trans. Softw. Eng. Methodol. 22(1): 6 (2013)
- Nina Elisabeth Holt, Richard Torkar, Lionel C. Briand, Kai Hansen: State-Based Testing: Industrial Evaluation of the Cost-Effectiveness of Round-Trip Path and Sneak-Path Strategies. ISSRE 2012: 321-330
- Razieh Behjati, Tao Yue, Lionel C. Briand: A Modeling Approach to Support the Similarity-Based Reuse of Configuration Data. MoDELS 2012: 497-513
- Shaukat Ali, Lionel C. Briand, Andrea Arcuri, Suneth Walawege: An Industrial Application of Robustness Testing Using Aspect-Oriented Modeling, UML/ MARTE, and Search Algorithms. MoDELS 2011: 108-122