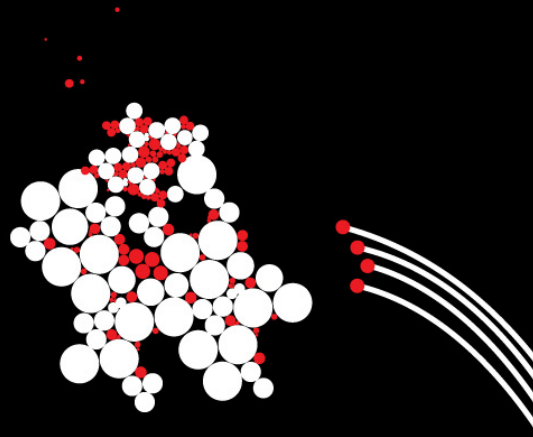
UNIVERSITY OF TWENTE.

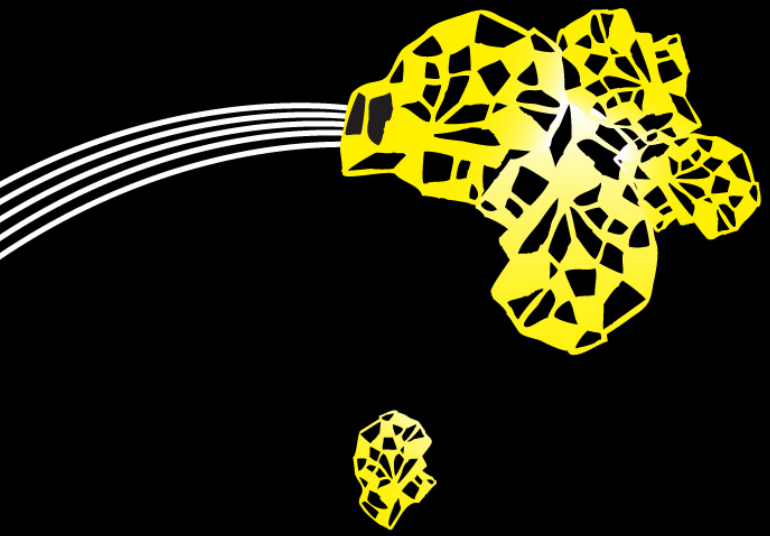


Part 1: Efficient Domain-Specific Tool Development for UPPAAL via Model-Driven Engineering

Stefano Schivo, Buğra Yildiz, **Enno Ruijters**, Christopher Gerking, Rajesh Kumar, Stefan Dziwok, Arend Rensink, and Mariëlle Stoelinga

University of Twente

26 September 2017

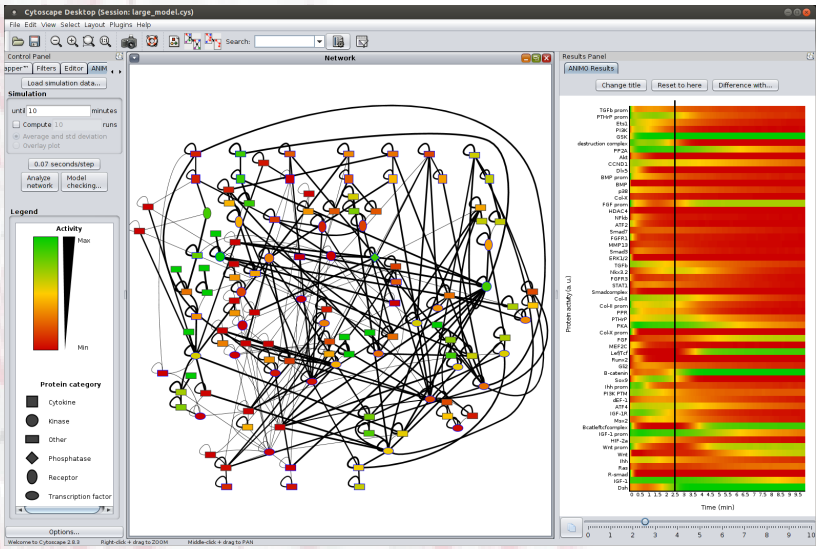


Our contribution in a nutshell

- UPPAAL is a popular back-end model-checker.
  - Used for network protocols, biology, battery-aware scheduling, etc.
- Front-end tools and translations are often developed ad-hoc.
  - Difficult to debug, reuse, extend, etc.
- Model-driven engineering provides more structure for translations.
  - Metamodels for different domains and UPPAAL.
  - Specialized languages for transformations.
- Application of MDE improves development of front-end tools:
  - Better interoperability
  - Faster development
  - Easier to maintain

Bridging tools

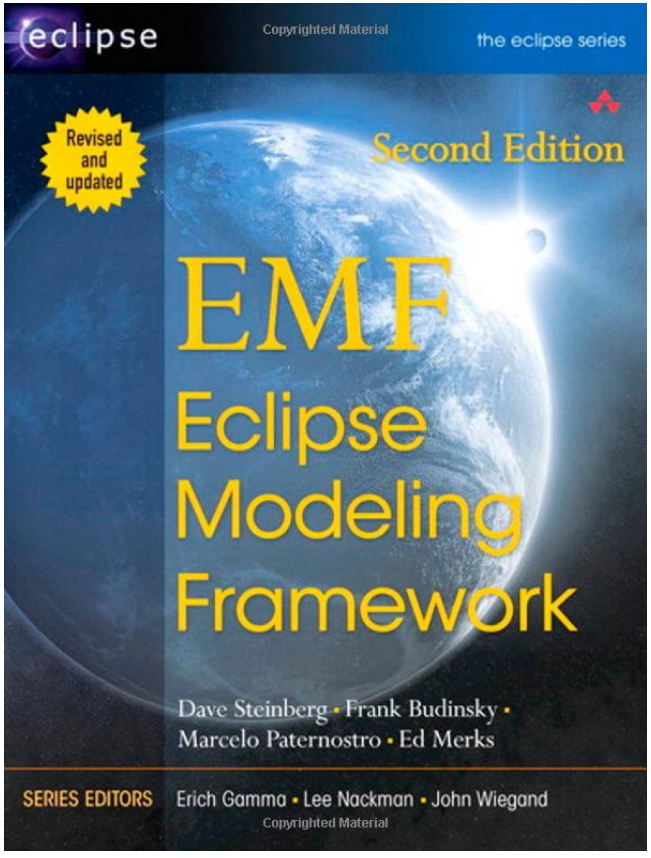
- Typical process:
  - Input domain-specific language (DSL)
  - Translate to general-purpose model (e.g. UPPAAL timed automata)
  - Run general-purpose analysis.
  - Translate result back to domain.



- Disadvantages of ad-hoc translations:
  - General language (Java, etc.)
  - Little/no documented structure of models.
  - Closely tied to specific versions/features.

Benefits of MDE

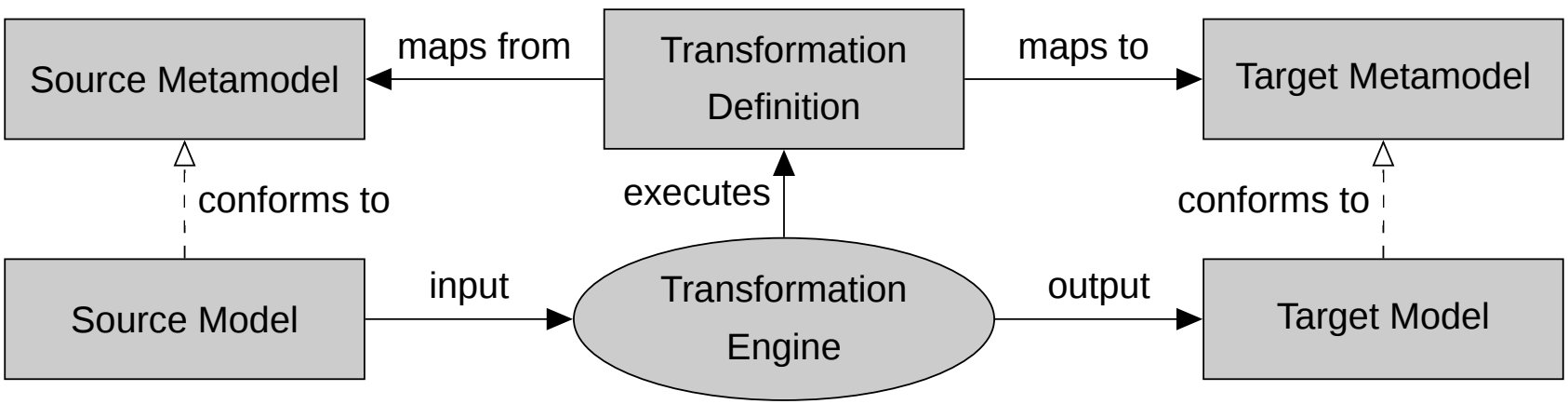
- Interoperability
  - Model transformations facilitate combinations of domains and tools.
- Reusability
  - Metamodels can be reused by different tools in the same domain.
  - Downstream transformations can be reused across domains.
- Faster development
  - Domain experts can focus on the domain.
  - Transformation and metamodeling languages are specific-purpose.



- We present work in UPPAAL, this applies to your tool too :)

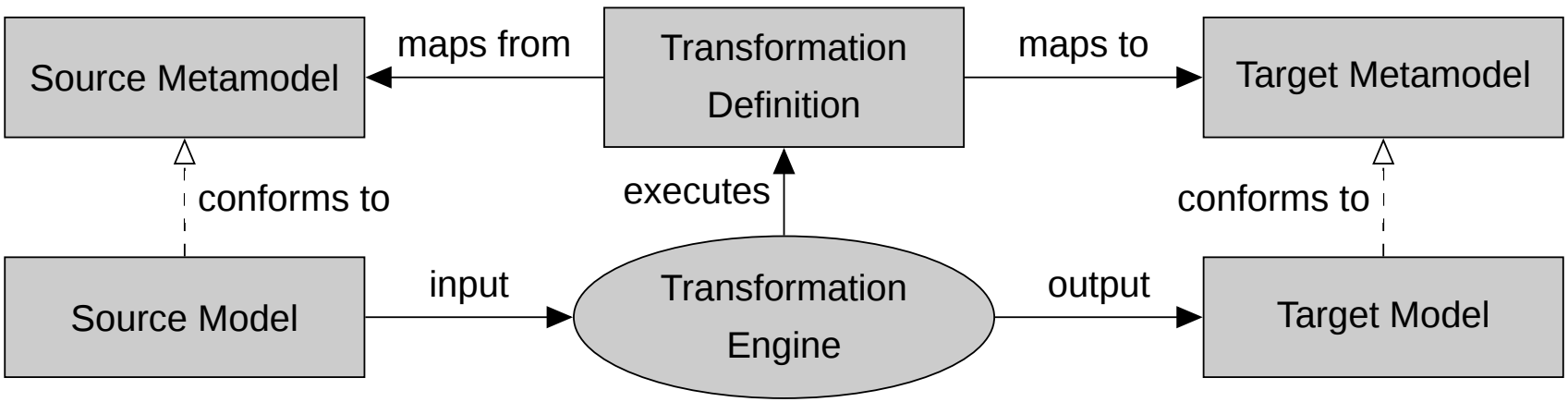
Model-Driven Development (MDE)

- Have models as first-class citizens.
- Metamodels provide syntax and documentation of models.
- Model transformations in purpose-specific languages (Epsilon, etc.)
- Validation/constraints help in debugging and documentation.



Model transformations

- Transform one (metamodel-described) model to another.
- Transformation languages have specific constructs for model concepts.

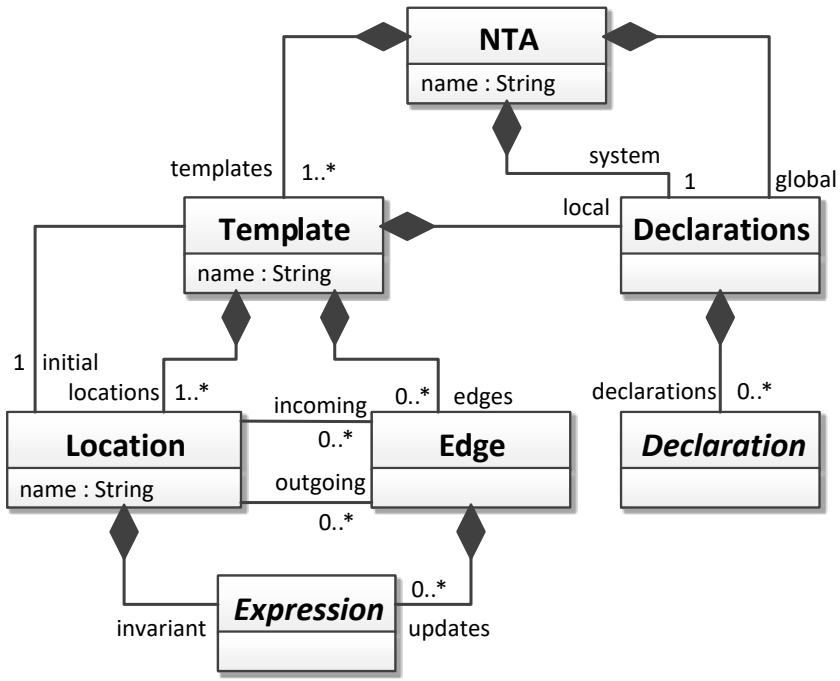


Model transformation: Example

```
rule Base
     transform at : AFT!AttackTree to out : Uppaal!NTA
{
     out.systemDecl = new Uppaal!SystemDeclarations();
     out.systemDecl.system = new Uppaal!System();
     for (node : AFT!Node in at.Nodes) {
          var converted = node.equivalent();
          out.template.add(converted.get(0));
     }
     ...
}
rule andGate transform node : AFT!Node to ret : List
{
     guard : node.nodeType.isKindOf(AFT!AND)
...
```

Metamodels

- Describe syntax of a class of models.
- Object-oriented format (classes, attributes, etc.)
- Shown in UML-like syntax.
- Concrete models are *instances* of such a metamodel.

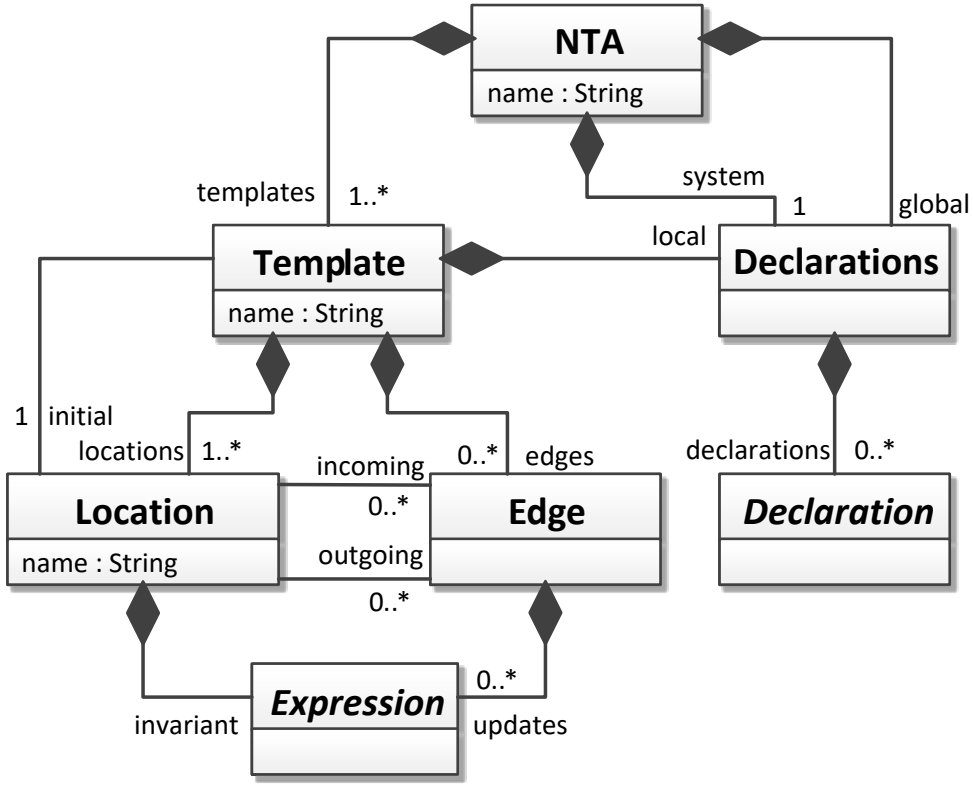


MDE in the steps of a front-end tool

1. Domain expert produces domain-specific model in a metamodel.
2. Domain-specific model is transformed to timed automata using a model transformation.
3. Property of interest is specified in domain-specific language in a metamodel.
4. Property is translated to UPPAAL query using a model transformation.
5. UPPAAL checks query and possibly produces resulting trace.
6. Trace is transformed back to domain-specific representation and presented supported by model transformations and metamodels.

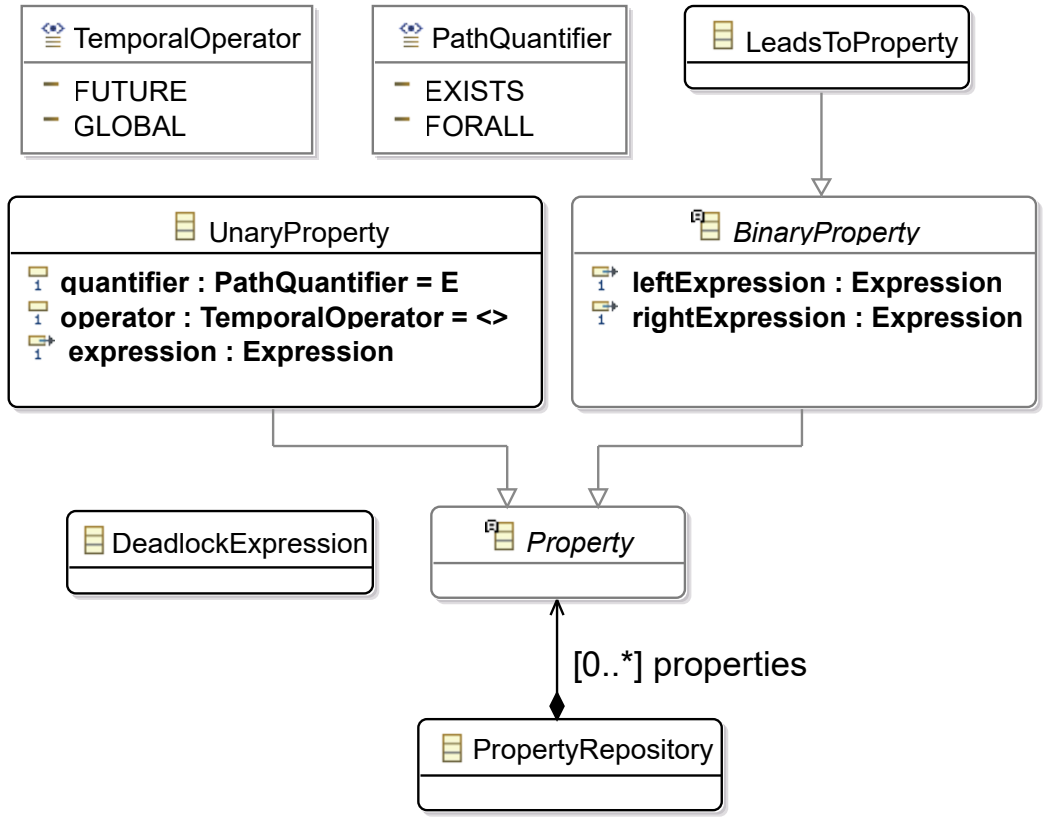
UPPAAL metamodels: Timed automata

- System under analysis.
- Input to UPPAAL.
- Includes automatic transformation to XML files.



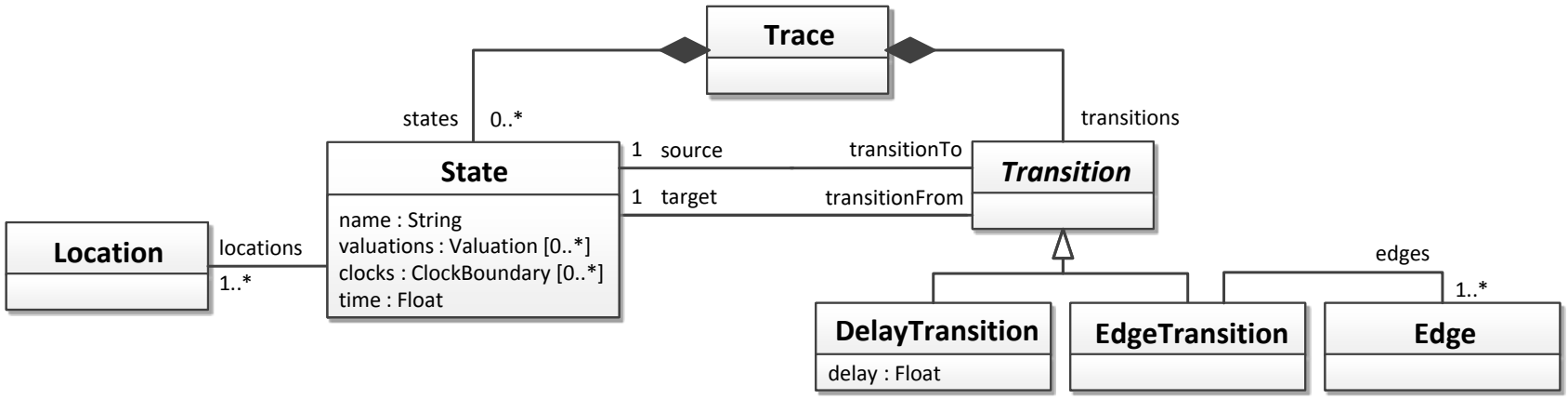
UPPAAL metamodels: Query

- TCTL-like language
- Includes transformation to textual format.
- Linked to NTA for queries on specific models.



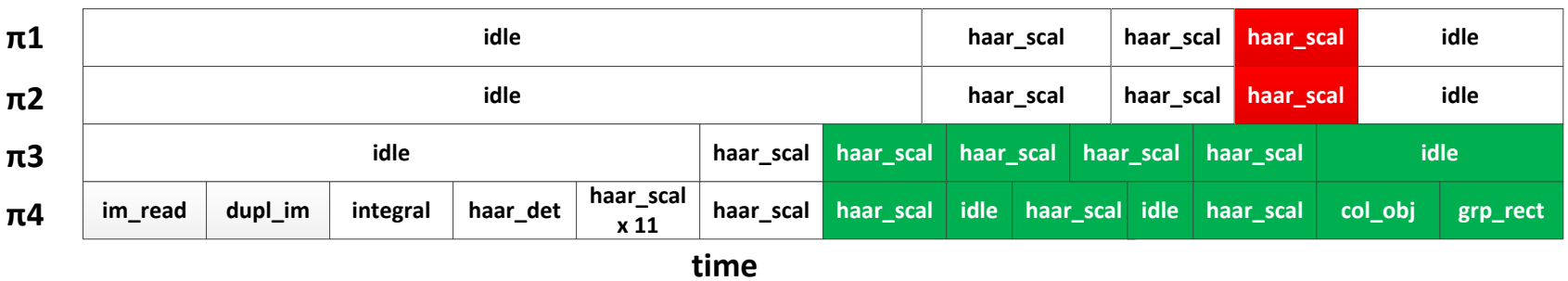
UPPAAL metamodels: Trace

- Counterexamples/witnesses produced by UPPAAL.
- Including parser for textual output.
- Links back to NTA for easy interpretation.



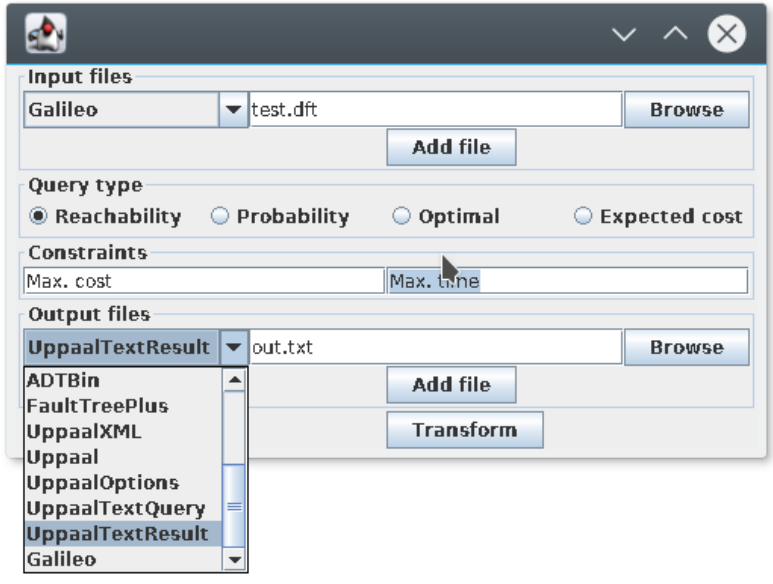
Example tool: Synchrononous dataflow graphs (SDF)

- Hardware-software co-design for streaming applications.
- Inputs: SDF graphs, hardware platform model, allocation model.
- Transform to UPPAAL CORA to obtain cost-optimal trace.
- Transform trace to domain-specific Schedule metamodel.
- Present schedule to user.



Example tool: Attack/Fault Trees

- Converter and analyzer for many variants of fault and attack trees.
- Originally only for ATs.
  - MDE made it easy to extend to FTs and combinations.
- Supports inputs from 4 different tools.
- Outputs to 6 tools, not counting three variants of UPPAAL.

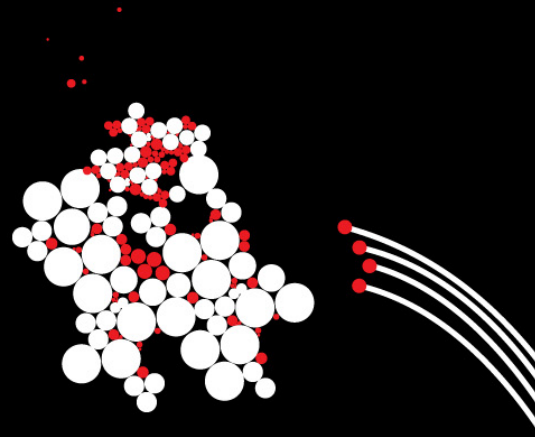


- Key: Single unified metamodel for attack & fault trees.

Conclusions

- Model-driven engineering framework for front-end tools for UPPAAL.
- Metamodels and transformations provide structure.
  - Easier to debug, extend, maintain, etc.
- Promotes reuse and interoperability between domains.
- Formals methods have helped software engineering, now let software engineering help you!
- Metamodels available at `https://github.com/utwente-fmt/uppaal`

UNIVERSITY OF TWENTE.

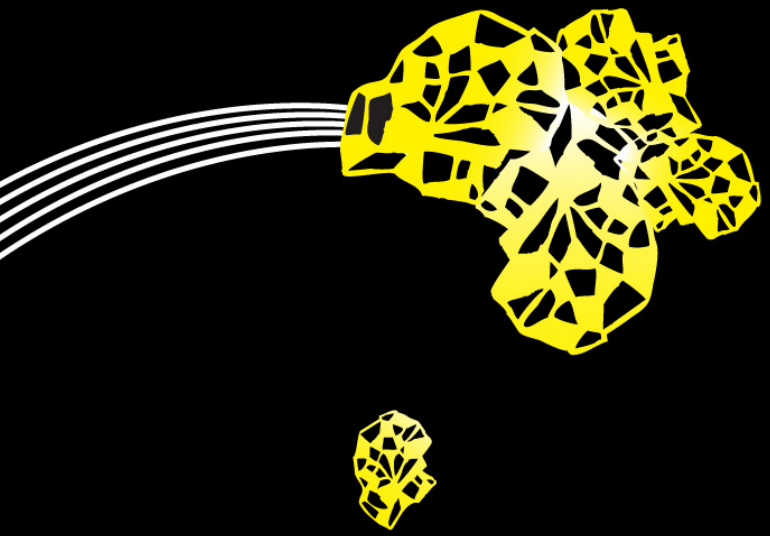


Part 2: Importance sampling for dynamic fault trees

Enno Ruijters, Daniël Reijsbergen,
Pieter-Tjerk de Boer, and Mariëlle Stoelinga

University of Twente

26 September 2017



Our contribution in a nutshell

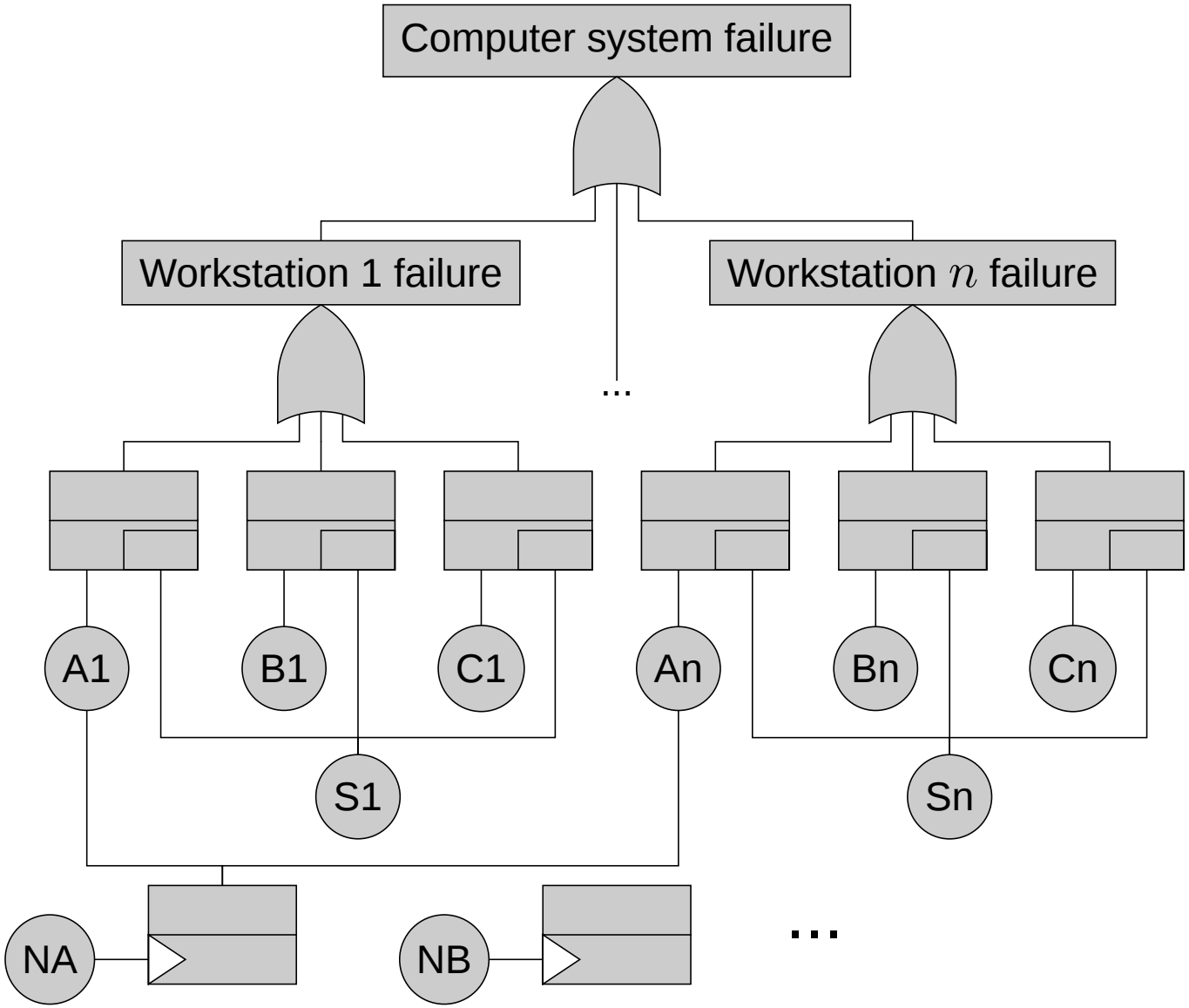
- Many frameworks can provide quantitative dependability analysis.
  - We use dynamic fault trees.
  - Compute system **availability**, reliability, MTTF, etc.
- Complex systems are computationally difficult to analyze:
  - Complex $\rightarrow$ analytic approaches are memory-intensive.
  - Rare failures $\rightarrow$ Monte Carlo simulation requires many samples.
- Our solution: rare event simulation (through importance sampling)
  - Make rare events more likely.
  - Compensate the final result.
  - Automatically.
- **Rare event simulation + dynamic fault trees $\rightarrow$ Faster/more accurate fault tree simulation.**

Comparison of RES techniques

Splitting	Sampling
Requires formalization of distance	Requires specification of ‘rare’ transitions
Changes simulation engine	Changes system under simulation
Good for rare events of many steps	Good for rare event of few steps
Limit case: fewer runs needed	Limit case: only one run needed

We use importance sampling as our system reaches the rare event after only a few, low-probability transitions. Such models provide few points to split the samples.

DFT example

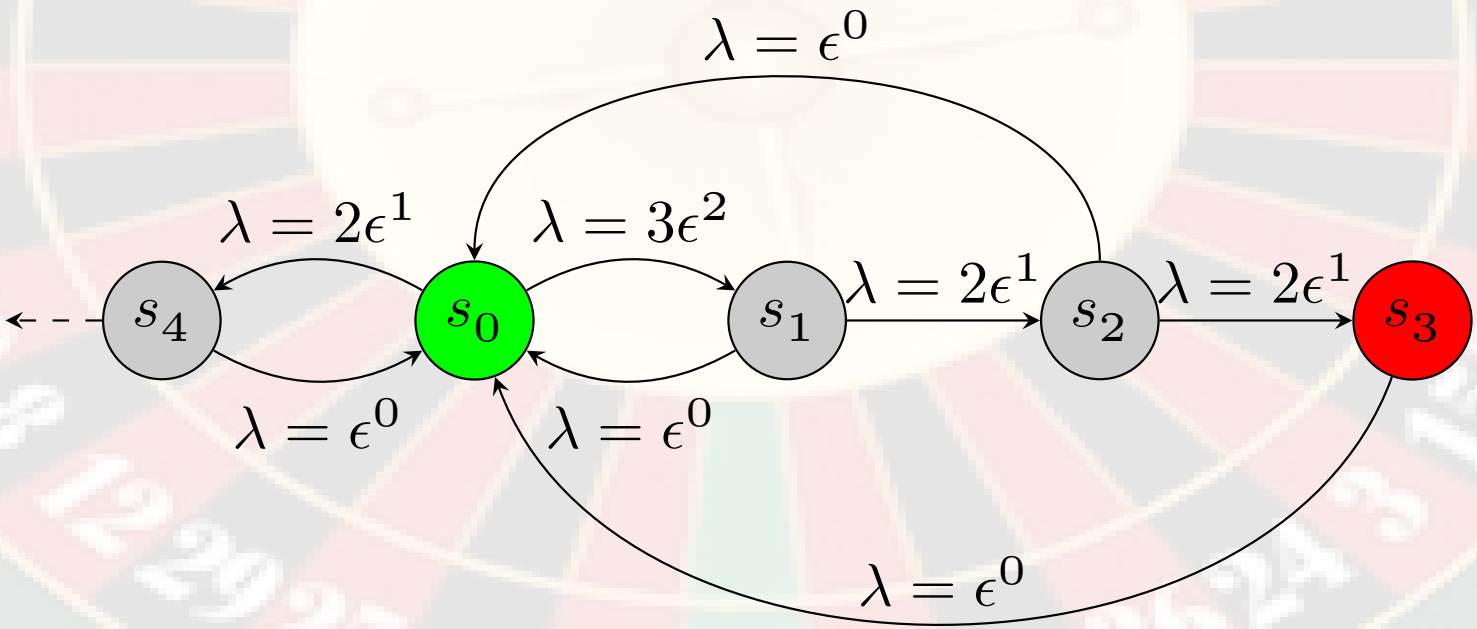

Computer system failure
Workstation 1 failure
Workstation n failure
...
A1
B1
C1
An
Bn
Cn
S1
Sn
NA
NB
...

Path-ZVA algorithm

- Importance sampling algorithm for cyclic Markovian models.
- Divides states into three categories:
  - ‘Perfect’ states reached frequently.
  - ‘Bad’ states reached rarely.
  - ‘Connecting’ states inbetween.
- Estimates:
  - Probability of reaching ‘bad’ states before returning to ‘perfect’ states.
  - Fraction of time spend in ‘bad’ states.
- Transition rates parameterized as $r \cdot \epsilon^n$ with $0 < \epsilon << 1$ to indicate ‘rareness’.

Applying Path-ZVA to DFTs

- Basic idea: Compute state space on-the-fly.
- Path-ZVA stores the subset of states in dominant paths.
- All other states only generated as reached, and not stored.

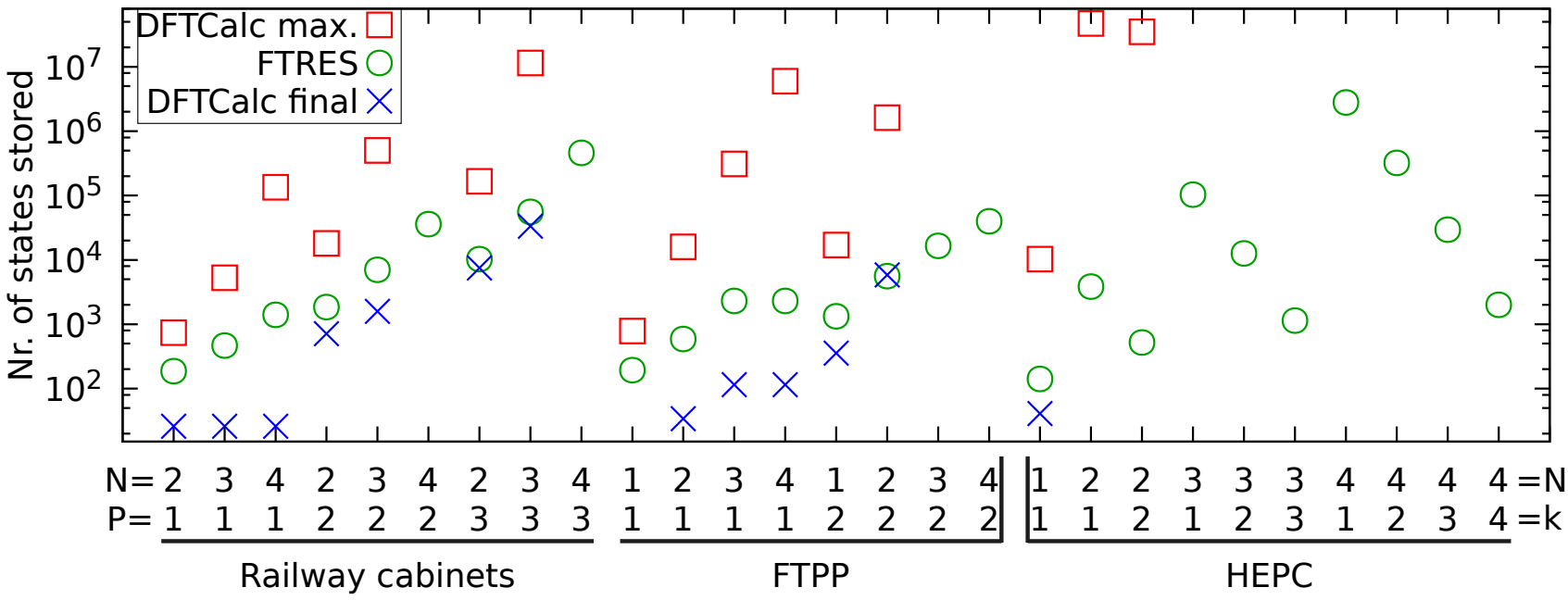


Results: Accuracy

Exact result for DFTCalc, 95% confidence for others:

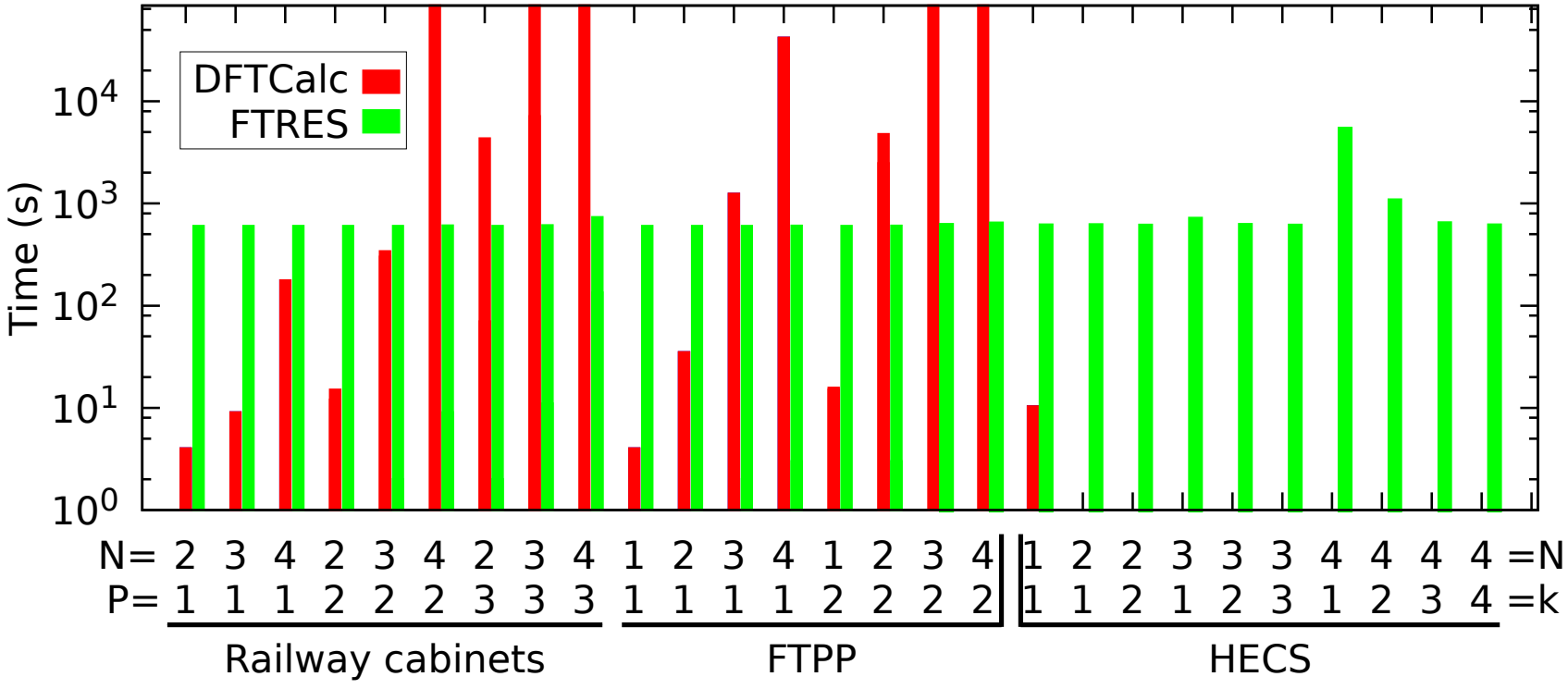
			Unavailability		
	N	**P**	**DFTCalc**	**FTRES**	**MC**
FTPP	1	1	$2.18303 \cdot 10^{-10}$	$[2.182; 2.184] \cdot 10^{-10}$	–
	4	1	$2.22979 \cdot 10^{-10}$	$[2.229; 2.230] \cdot 10^{-10}$	$[0; 2.140] \cdot 10^{-8}$
	1	2	$1.76174 \cdot 10^{-20}$	$[1.761; 1.763] \cdot 10^{-20}$	–
	4	2	–	$[1.257; 2.553] \cdot 10^{-20}$	–
	N	**k**	**DFTCalc**	**FTRES**	**MC**
HECS	1	1	$4.12485 \cdot 10^{-5}$	$[4.118; 4.149] \cdot 10^{-5}$	$[2.615; 10.64] \cdot 10^{-5}$
	2	1	–	$[3.010; 3.061] \cdot 10^{-9}$	–
	2	2	–	$[8.230; 8.359] \cdot 10^{-5}$	$[0; 1.734] \cdot 10^{-4}$
	4	1	–	$[1.328; 8.213] \cdot 10^{-17}$	–
	4	2	–	$[1.145; 1.270] \cdot 10^{-12}$	–
	4	3	–	$[1.744; 1.817] \cdot 10^{-8}$	–
	4	4	–	$[1.609; 1.667] \cdot 10^{-4}$	–

Overall results: State space



- FTRES always below DFTCalc maximal state space size.
- FTRES computes results where DFTCalc does not.

Overall results: Speed



- FTRes and MC spend a constant 5 mins. simulating.
- Simulation time mostly dominates state-space exploration.
- Almost all DFTCalc experiments for HECS ran out of memory.

Thank you for your attention.

Questions?