Practical Model-based Testing With Papyrus and RT-Tester

Jan Peleska and Wen-ling Huang
University of Bremen
Verified Systems International GmbH

Fourth Halmstad Summer School on Testing,
2014-06-11

Universität Bremen

COMPASS

Verified

Acknowledgements. *This presentation has been elaborated in the context of the EU FP7 COMPASS project under grant agreement no.287829.*

Overview

- Model-based testing
- Test Modelling With Papyrus
- Model-based Testing With RT-Tester
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises

Overview

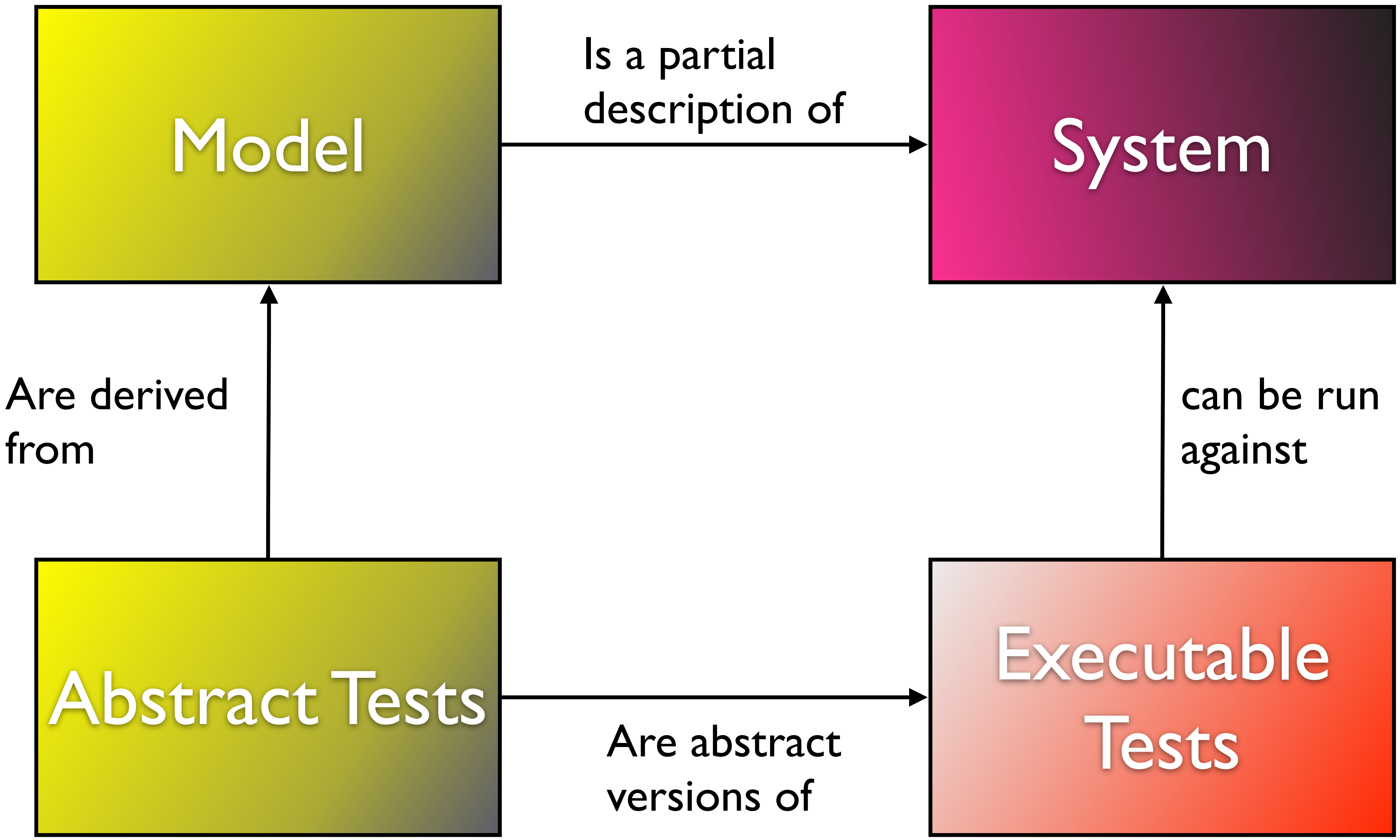
- Model-based testing
- Test Modelling With Papyrus
- Model-based Testing With RT-Tester
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises

Our MBT Approach

Instead of writing test procedures,

- develop a **test model** specifying expected behaviour of SUT ➔ the first MBT variant
- use **generator** to identify "relevant" test cases from the model and calculate concrete test data
- generate **test procedures** fully automatic
- perform **tracing** *requirements* ↔ *test cases* in a fully automatic way

MBT-Paradigm



Overview

- Model-based testing
- Test Modelling With Papyrus
- Model-based Testing With RT-Tester
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises

Papyrus

- Modelling with EMF-based formalisms
- EMF – Eclipse Modelling Framework
- Papyrus provides UML, SysML, DSL support
- Open source – free to use
- http://www.eclipse.org/papyrus/

SysML

- Block definition diagrams
- Internal block diagrams
- Item flows
- State machines with timers
- Operations
- Requirements
- <<satisfy>> relationship between requirements and model elements

Case Studies With SysML

- Simplified version of the turn indication and emergency flashing function in Daimler vehicles
- Full model available under

`http://www.mbt-benchmarks.org`

➔ `Benchmarks`

➔ `Turn Indicator Model Rev. 1.4`

Case Studies With SysML

- New model available: the Ceiling Speed Monitor of the ETCS (European Train Control System)
- Full model available under

`http://www.mbt-benchmarks.org`

➔ `Benchmarks`

➔ `openETCS/ceiling-speed-monitoring`

Model Introduction With Papyrus

- System interface – block diagram
- Requirements
- System Under Test – internal block diagram
- Further decompositions – internal block diagrams and block references
- Behaviour associated with block leaves – state machines and operations

Overview

- Model-based testing
- Test Modelling With Papyrus
- Model-based Testing With RT-Tester
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises

RT-Tester Internals

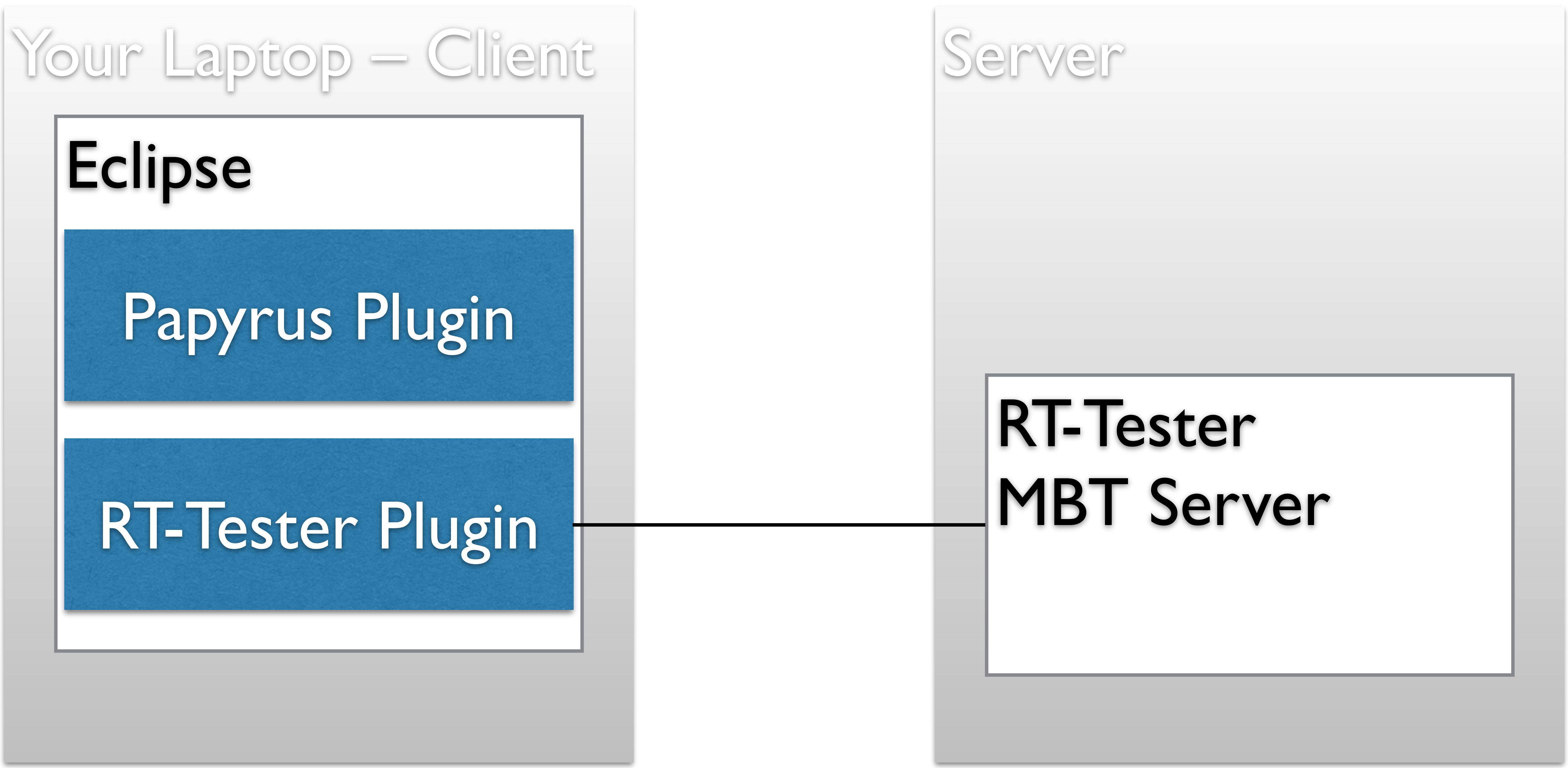
Further reading. Industrial-Strength Model-Based Testing - State of the Art and Current Challenges. In Petrenko, Alexander K. and Schlingloff, Holger (eds.): Proceedings Eighth Workshop on Model-Based Testing, Rome, Italy, 17th March 2013, Electronic Proceedings in Theoretical Computer Science 111, pp. 3-28 (2013). DOI:10.4204/EPTCS.111.1

Reference Tool RT-Tester

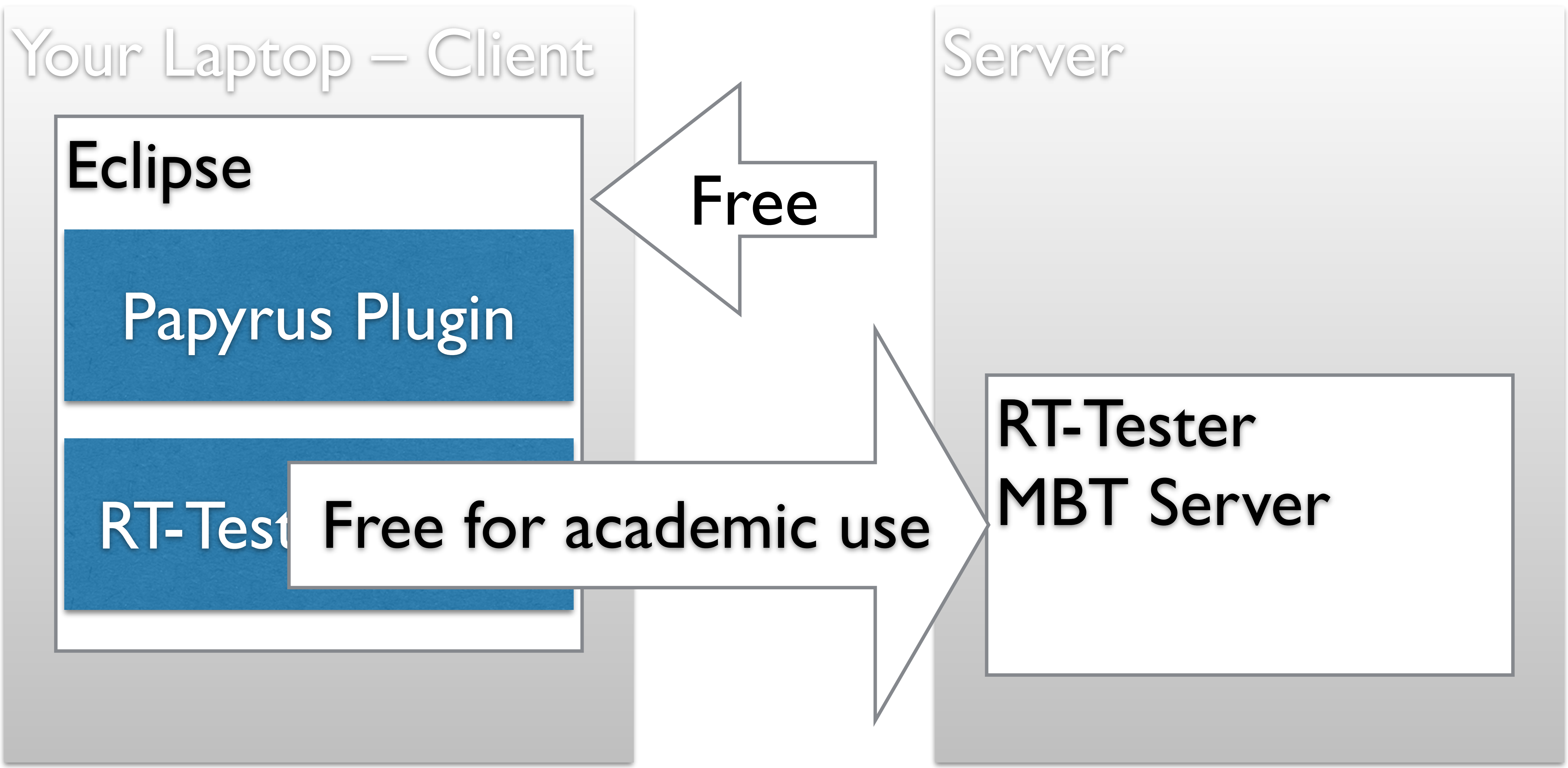
- Supports all test levels – from unit to system integration testing
- Software tests and hardware-in-the-loop tests
- Test projects may combine hand-written test procedures with automatically generated procedures

➜ The tool capabilities are presented here to stimulate benchmarking activities

Eclipse – Papyrus – RT-Tester Integration

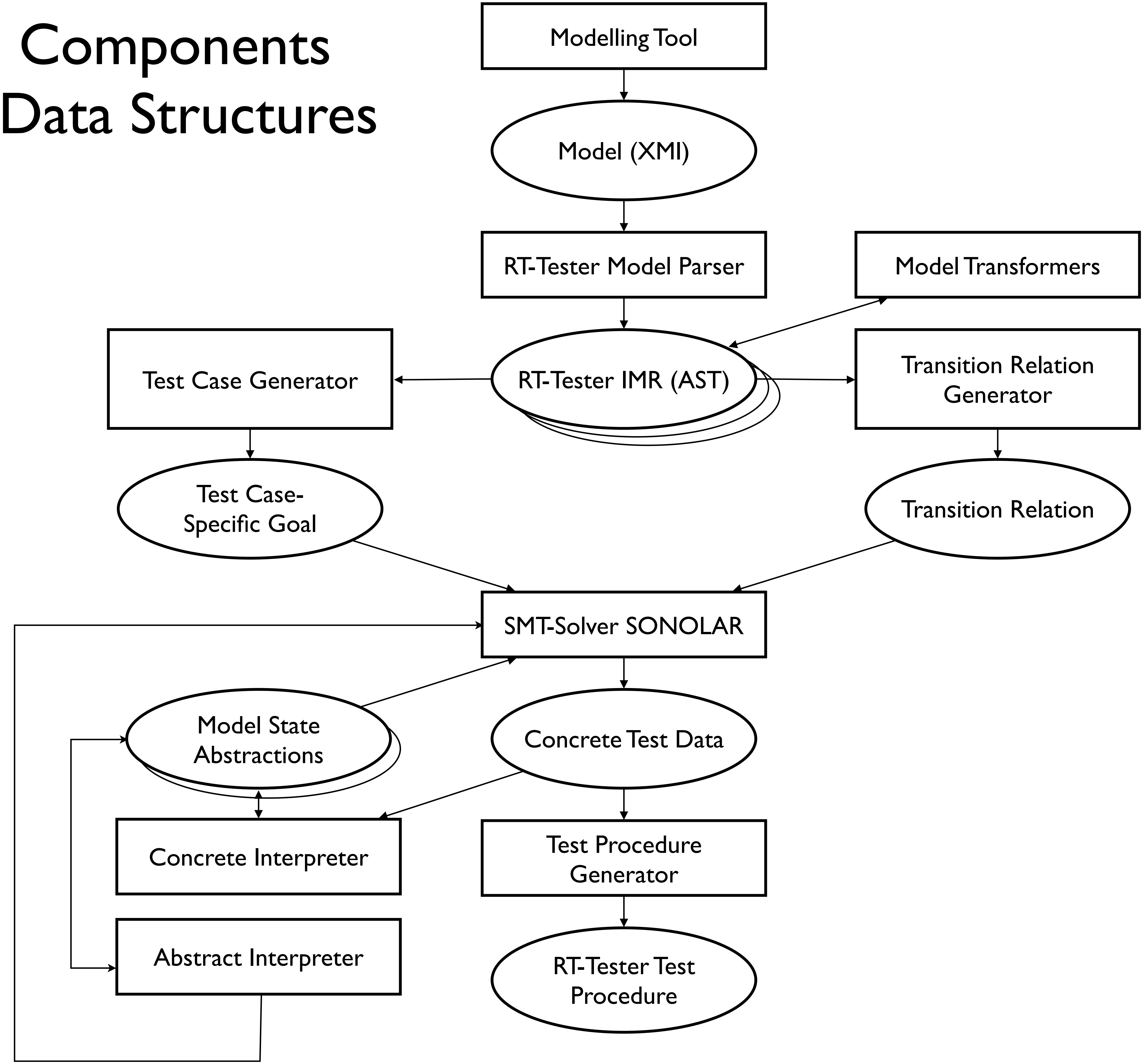


Eclipse – Papyrus – RT-Tester Integration

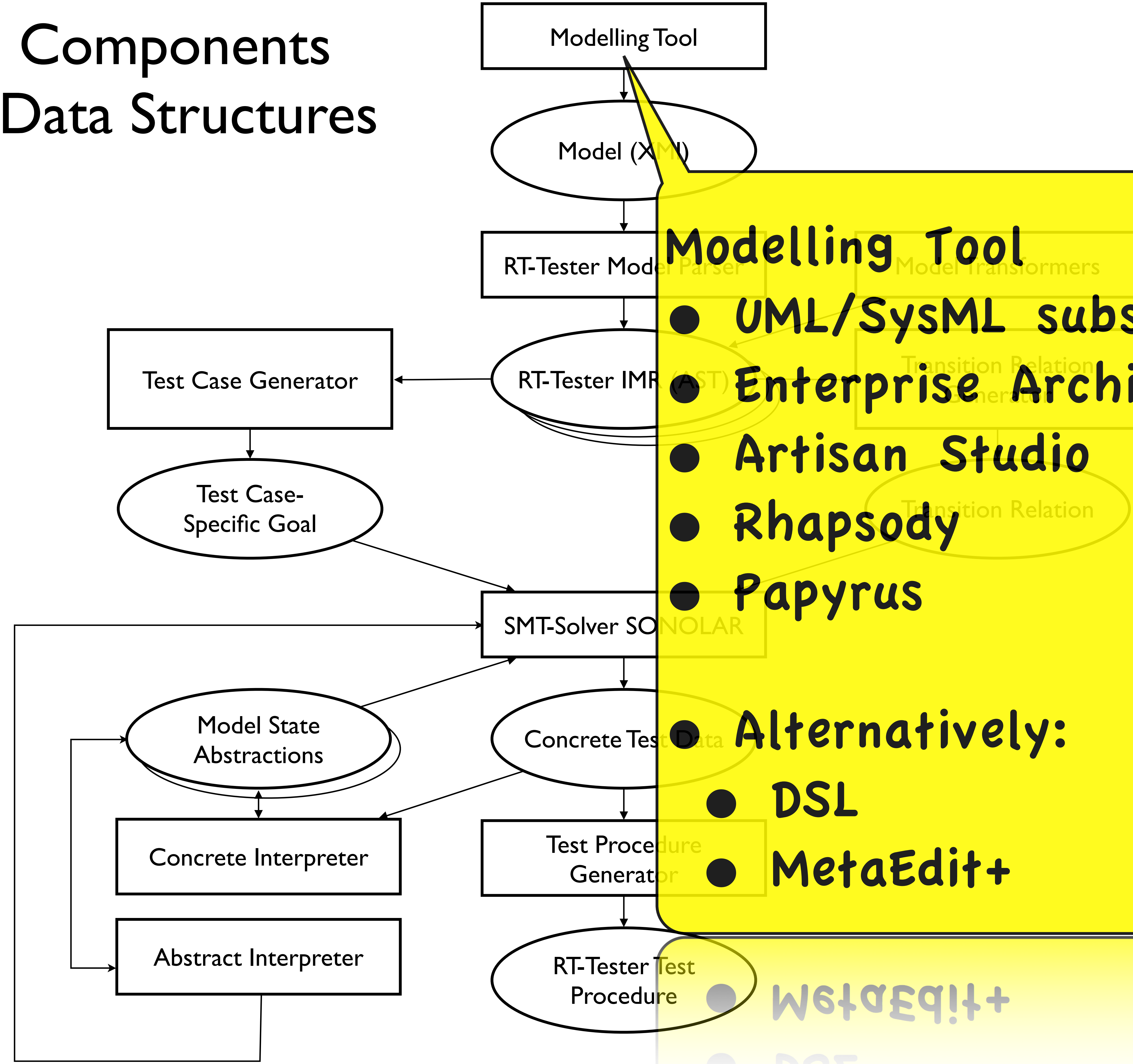


Server located at University of Bremen

Tool Components and Data Structures



Tool Components and Data Structures

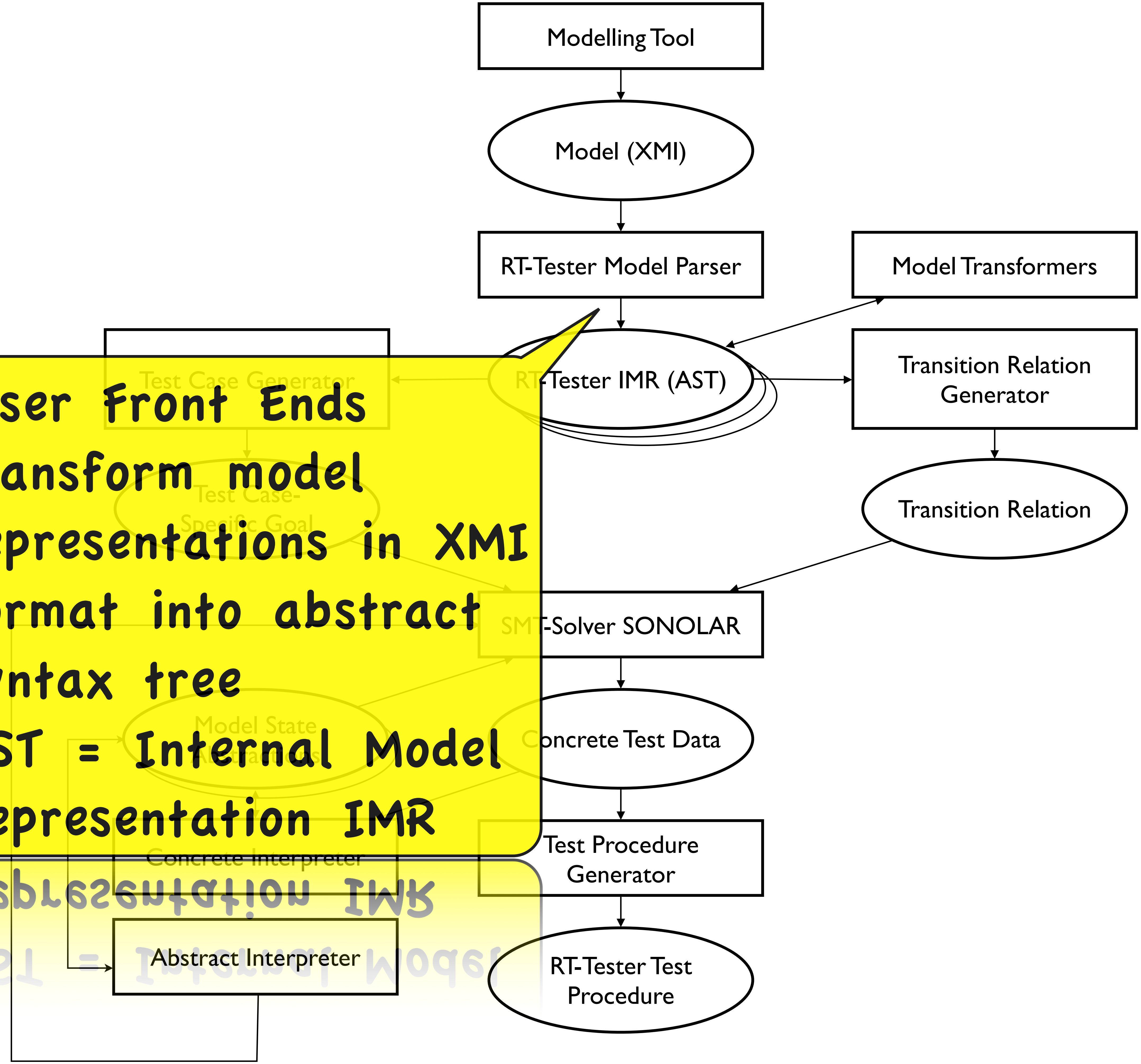


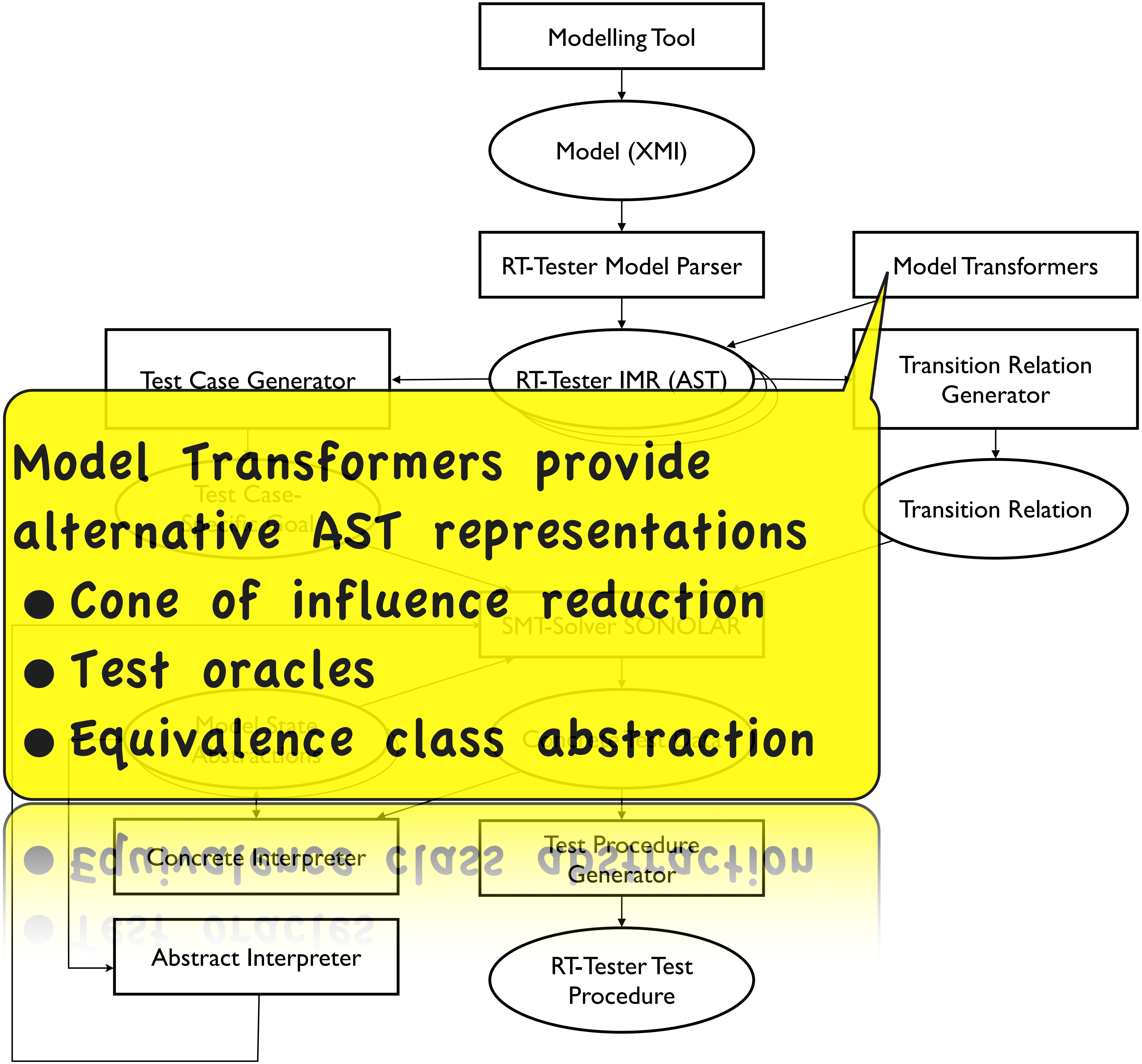
Modelling Tool

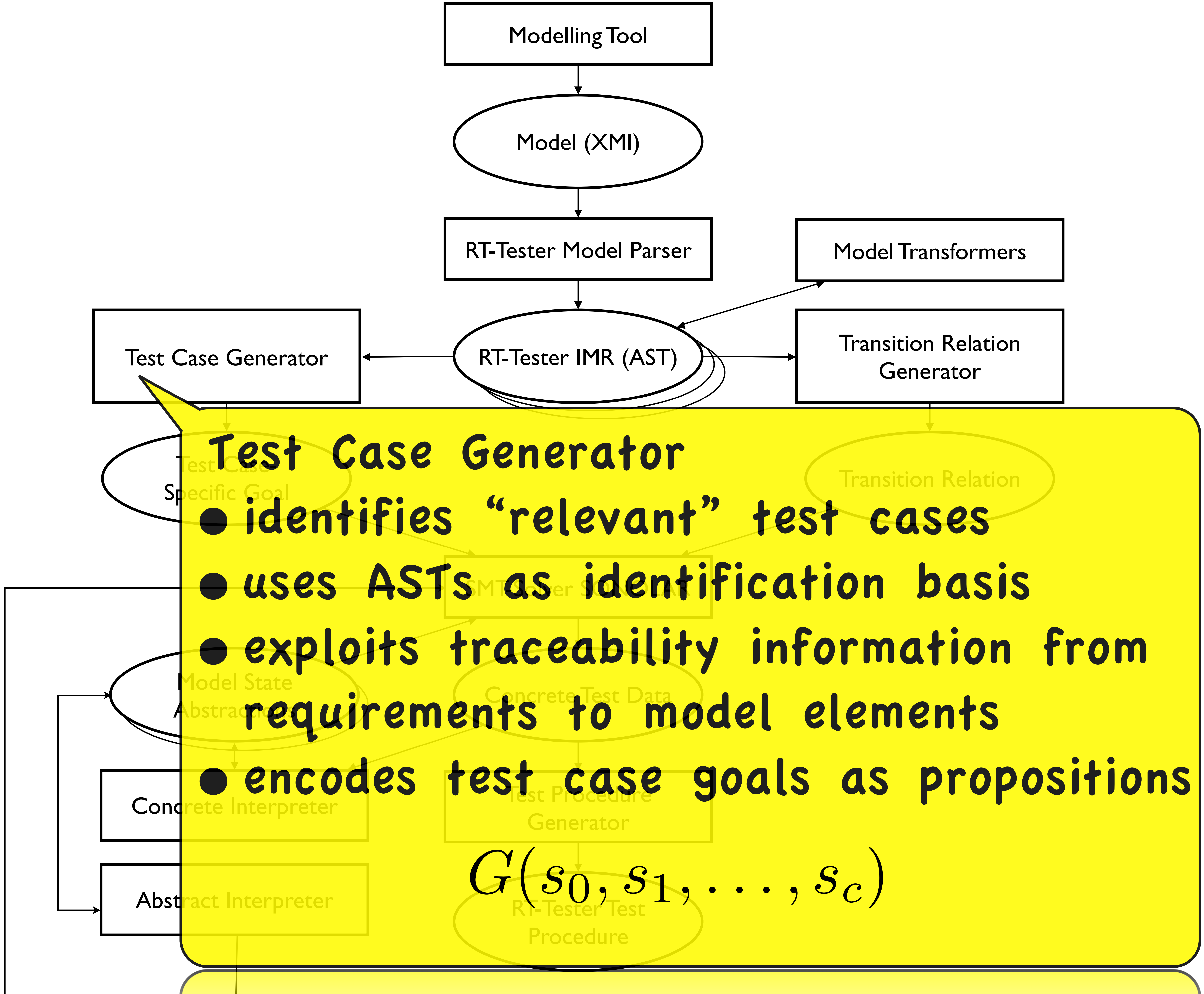
- UML/SysML subset
- Enterprise Architect
- Artisan Studio
- Rhapsody
- Papyrus
- Alternatively:
  - DSL
  - MetaEdit+

Parser Front Ends

- transform model representations in XMI format into abstract syntax tree
- AST = Internal Model Representation IMR




Modelling Tool
Model (XMI)
RT-Tester Model Parser
Model Transformers
Test Case Generator
RT-Tester IMR (AST)
Transition Relation Generator
Transition Relation
Model Transformers provide alternative AST representations
● Cone of influence reduction
● Test oracles
● Equivalence class abstraction
SMT-Solver SONOLAR
Concrete Interpreter
Test Procedure Generator
Abstract Interpreter
RT-Tester Test Procedure



Test Case Generator

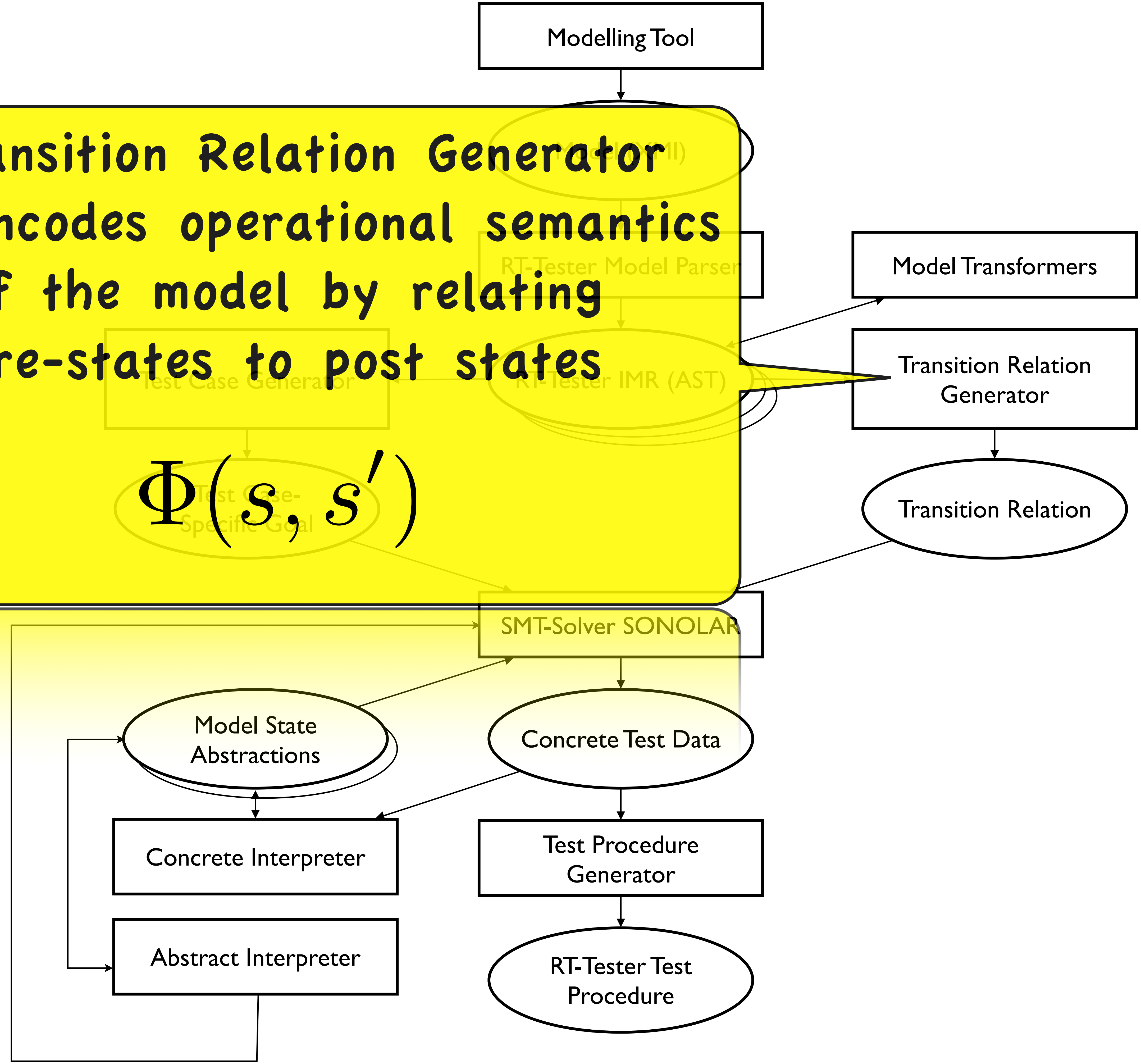
- identifies "relevant" test cases
- uses ASTs as identification basis
- exploits traceability information from requirements to model elements
- encodes test case goals as propositions

$$G(s_0, s_1, \ldots, s_c)$$

Transition Relation Generator

- encodes operational semantics of the model by relating pre-states to post states

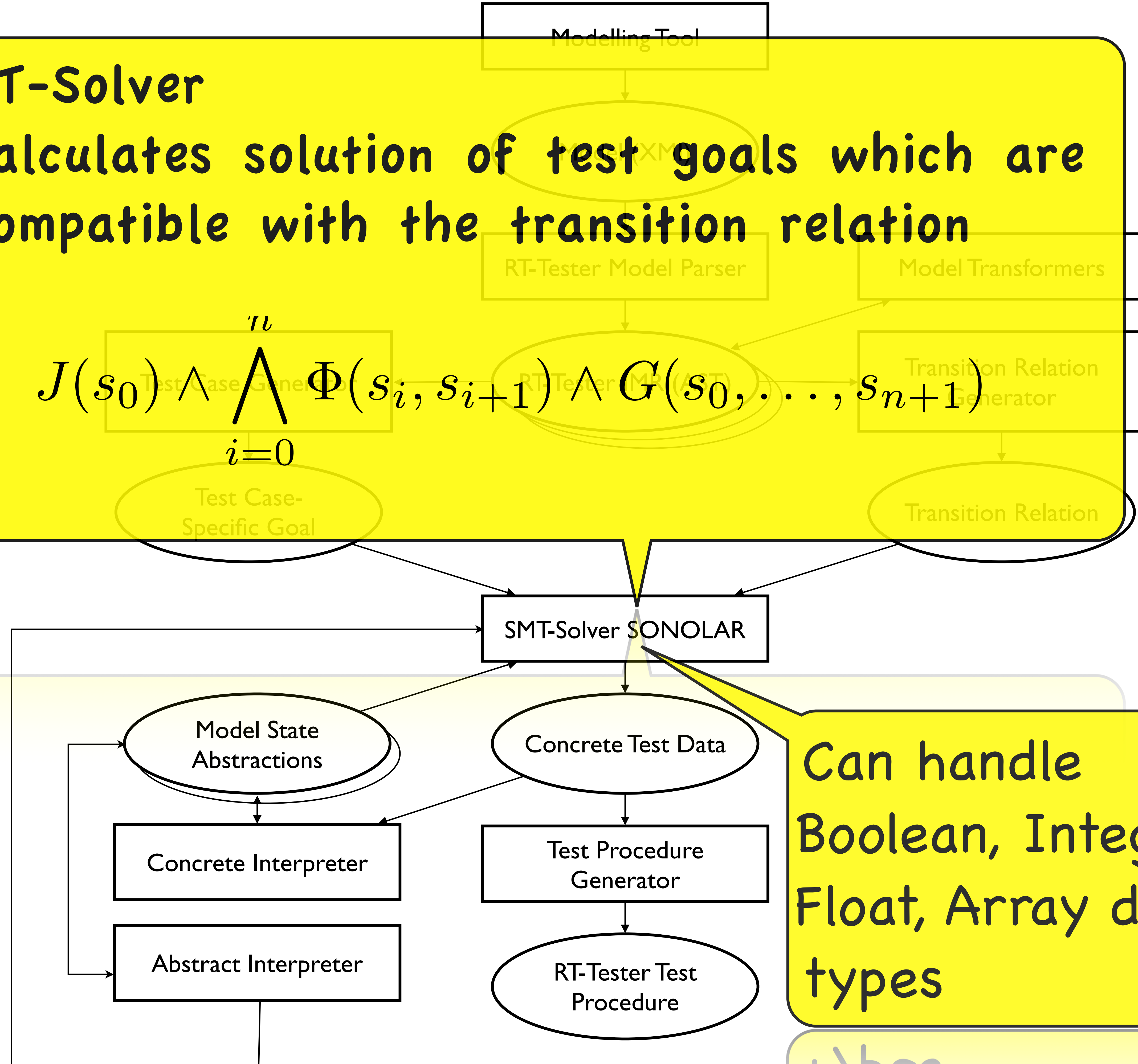
$$\Phi(s, s')$$



SMT-Solver

- calculates solution of test goals which are compatible with the transition relation

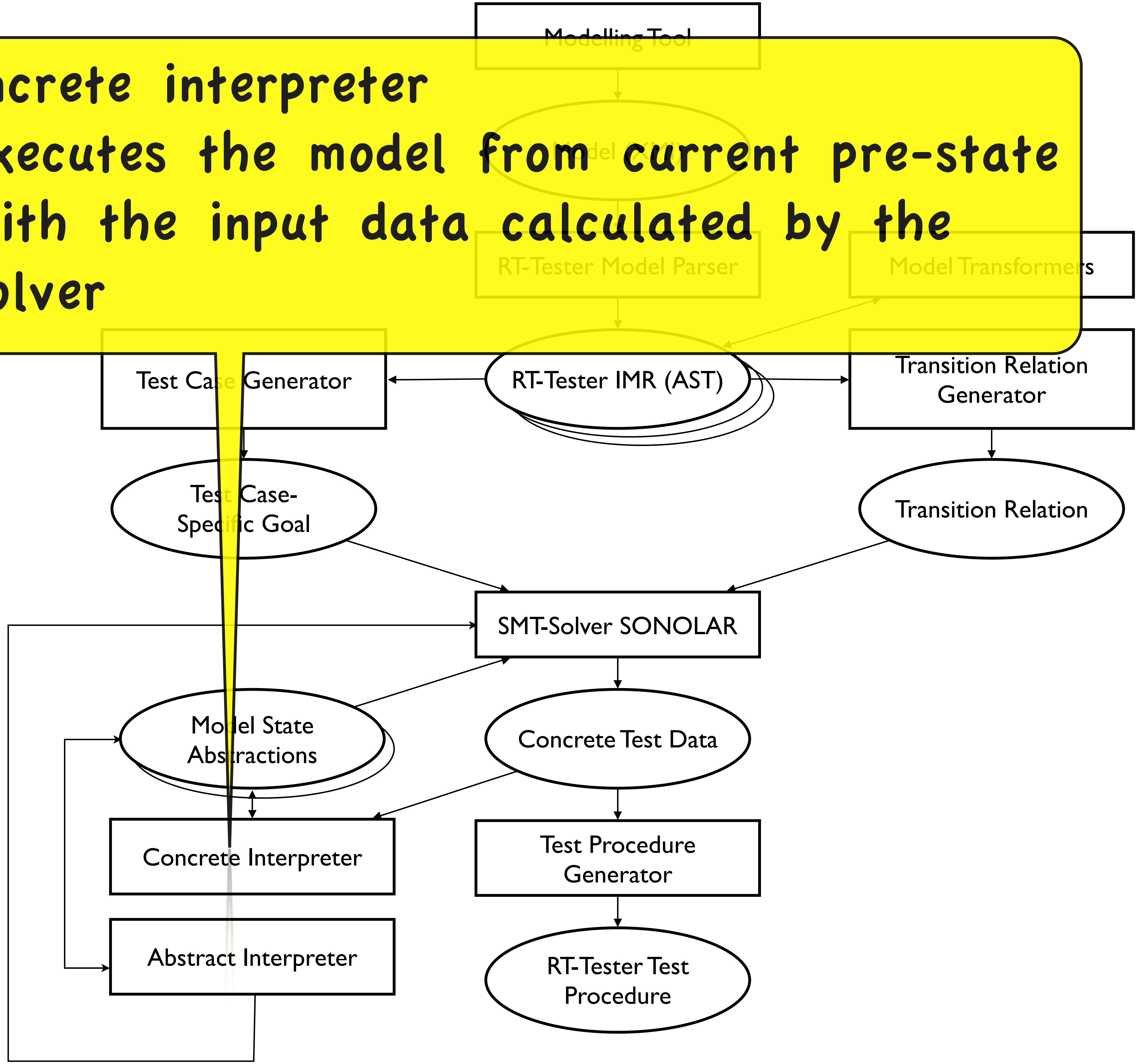
$$J(s_0) \wedge \bigwedge_{i=0}^{n} \Phi(s_i, s_{i+1}) \wedge G(s_0, \ldots, s_{n+1})$$



Can handle Boolean, Integer, Float, Array data types

Concrete interpreter

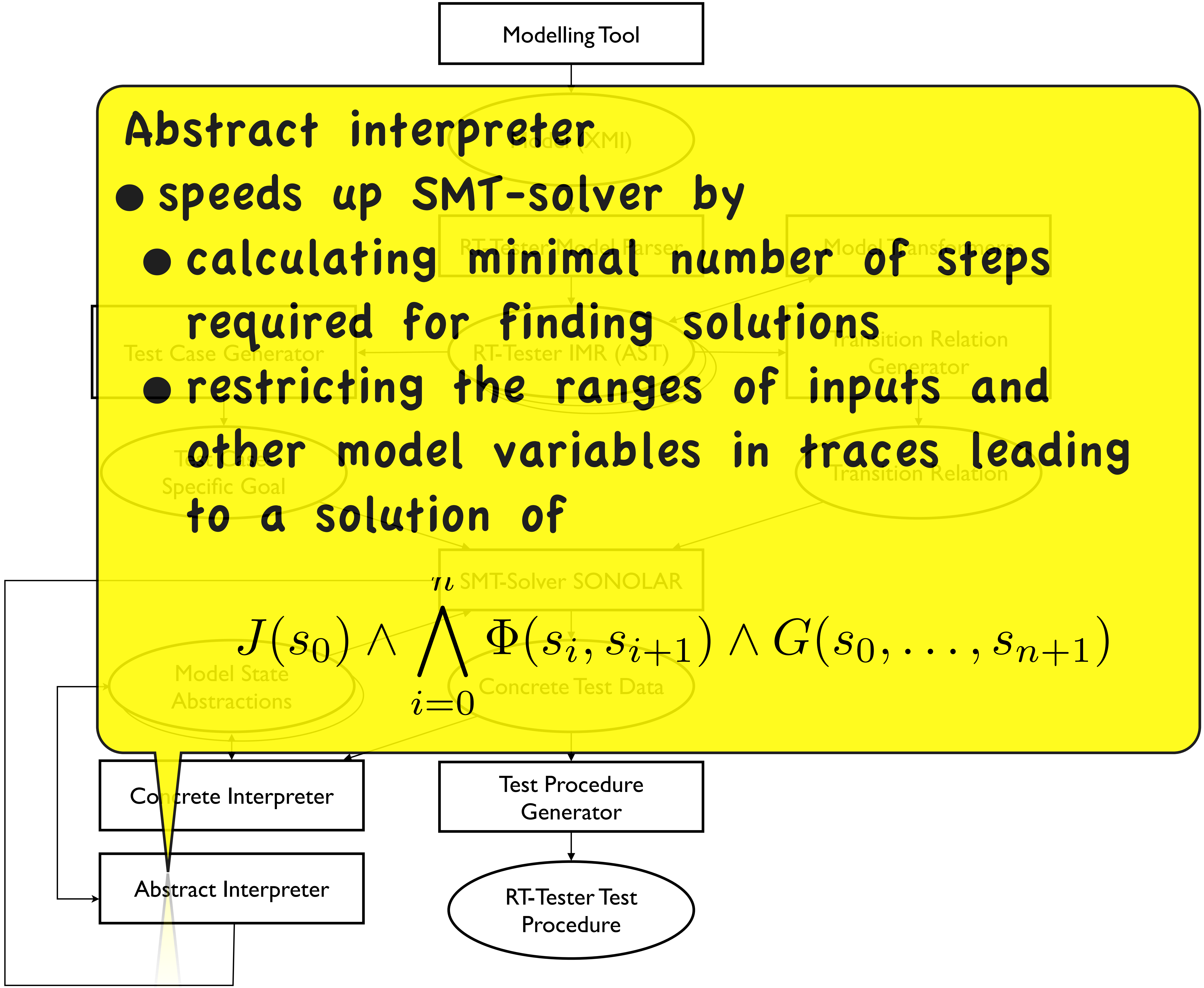
- executes the model from current pre-state with the input data calculated by the solver

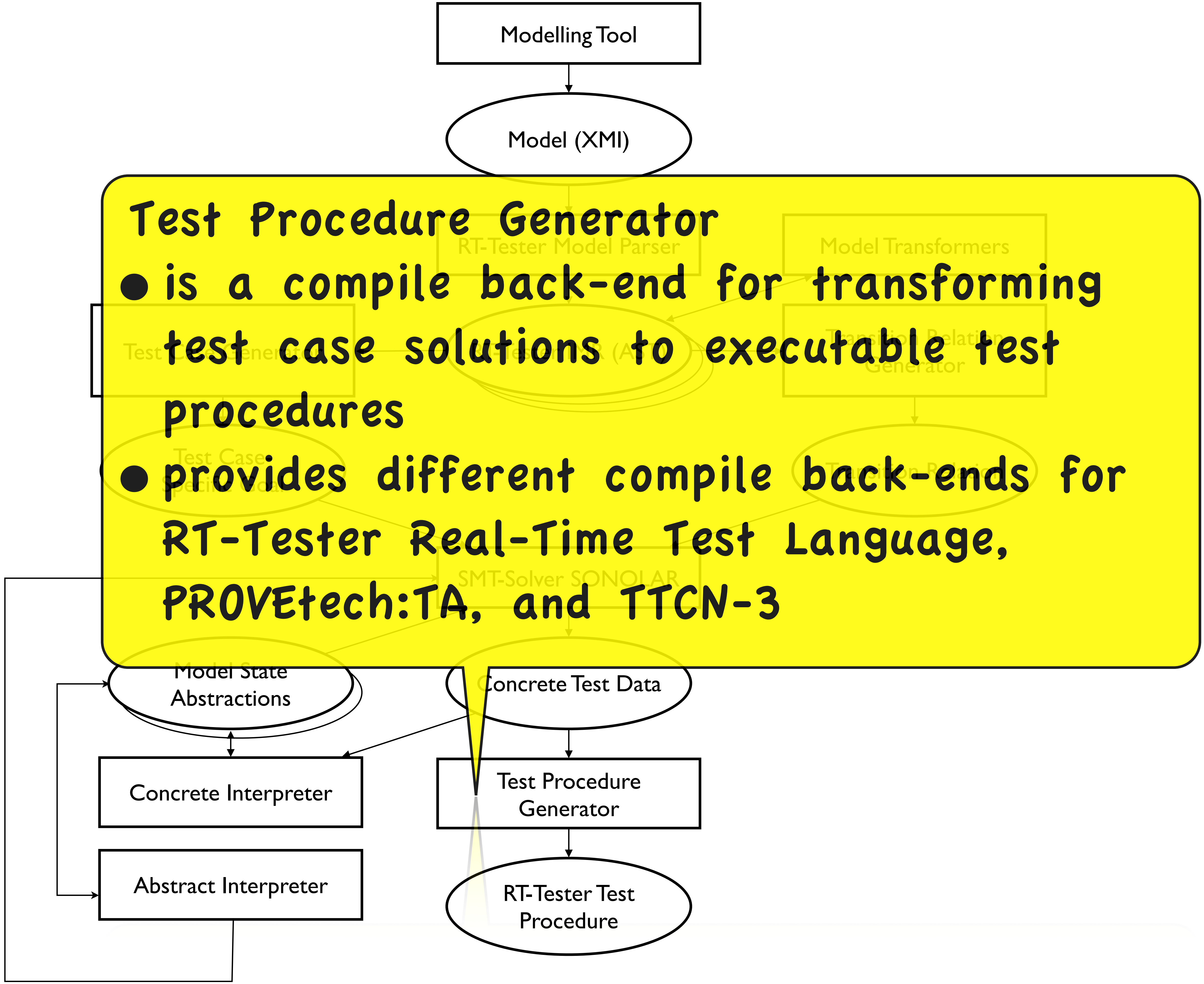


Abstract interpreter

- speeds up SMT-solver by
  - calculating minimal number of steps required for finding solutions
  - restricting the ranges of inputs and other model variables in traces leading to a solution of

$$J(s_0) \wedge \bigwedge_{i=0}^{n} \Phi(s_i, s_{i+1}) \wedge G(s_0, \ldots, s_{n+1})$$





Test Procedure Generator

- is a compile back-end for transforming test case solutions to executable test procedures
- provides different compile back-ends for RT-Tester Real-Time Test Language, PROVEtech:TA, and TTCN-3

Overview

- Model-based testing
- Test Modelling With Papyrus
- Model-based Testing With RT-Tester
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises

Model Semantics

- Based on Kripke Structures
- Equivalent to alternative operational semantics based on labelled transition systems

$$K = (S, S_0, R, L)$$

$$S : \text{State space}$$

$$S_0 \subseteq S : \text{Initial states}$$

$$R \subseteq S \times S : \text{Transition relation}$$

$$L : S \to 2^{AP} : \text{Labelling function}$$

$$AP : \text{Atomic propositions}$$

Requirements

- Each requirement is reflected by set of model computations

$$\pi = s_0.s_1.s_2\ldots$$

- Computation sets can be characterised by Linear Temporal Logic (LTL)

$$\mathbf{G}\phi : \text{Globally } \phi \text{ holds on path } \pi$$

$$\mathbf{X}\phi : \text{In the next state on path } \pi\text{, formula } \phi \text{ holds.}$$

$$\mathbf{F}\phi : \text{Finally } \phi \text{ holds on path } \pi$$

$$\phi\mathbf{U}\psi : \mathbf{F}\psi \text{ and } \phi \text{ holds on path } \pi \text{ until } \psi \text{ is fulfilled}$$

Requirements Tracing – Complex Requirements

- Computations contributing to complex requirements require full LTL expressions
- Insert LTL formula in constraint
- Link constraint to requirement via <<satisfy>> relation

Test Cases

- Test cases are finite witnesses of model computations
- Trace = finite prefix of a computation
- If computation satisfies LTL formula associated with a requirement, trace prefixes must at least not violate this formula
- Some formulas can only be verified on an infinite computation (liveness formulas, e.g. fairness properties)
- But these properties can only be partially verified by testing

Test Data Computation

- LTL formulas interpreted on finite traces can be transformed into first order expressions

$$tc \equiv J(s_0) \wedge \bigwedge_{i=0}^{n} \Phi(s_i, s_{i+1}) \wedge G(s_0, \ldots, s_{n+1})$$

- Recall. These formulas can be solved by an SMT solver

Model Coverage Strategies

Strategies currently realised in RT-Tester

- Basic control state coverage
- Transition coverage
- MC/DC coverage
- Hierarchic transition coverage
- Equivalence class and boundary value coverage
- Basic control state pairs coverage
- Interface coverage (under construction)
- Block coverage (under construction)
- Equivalence class partitioning (under construction)

Overview

- Model-based testing
- Test Modelling With Papyrus
- Model-based Testing With RT-Tester
- Requirements, test cases, procedures, results, and Traceability
- Demonstration and Practical Exercises

Test Generation Context and Test Execution Context

- Test generation context. Configure the test procedure to be generated
- Test execution context. Execute the test procedure against the system under test

Work Flow

- Create the test model (Papyrus perspective)
- Create RT-Tester project (RT-Tester perspective)
- Import model to RT-Tester project
- Configure and create initial test procedure – model-coverage approach
  - Configuration file
  - Signal map
  - Analyse signal flow

Work Flow

- Optional: create a simulation
- Compile and run test procedure
- Replay test procedure
- Analyse requirements and test cases
- Create new generation context
- Allocate test cases to procedure to be generated