A Generic White-Box Model Transformation Testing Framework

Dorian Leroy

JKU Linz

October 19th 2017

The TETRA Box Project

Austrian research project funded by the FWF resulting of the collaboration between:

- Business Informatics Group (BIG) of TU Wien,
- Cooperative Information Systems (CIS) group of JKU Linz.

My PhD:

- Under the supervision of Manuel Wimmer (TU Wien) and Benoit Combemale (UT2J)
- One year in France, two years in Austria

Context: Model Transformation Testing

- The role of model transformations is to analyze, create and modify models.
- They are an essential tool to manage vast quantities of models.
- Testing them is thus critical to the success of model-driven engineering (MDE).

About Existing Approaches

Existing model transformation testing (MTT) approaches:

- are **tightly coupled** with specific transformation languages,
- use **black-box testing**: they do not leverage the definition of the transformation to improve results.

Open Challenge

There is thus a need for a **language-generic**, **white-box** MTT framework.

About Existing Approaches

Existing model transformation testing (MTT) approaches:

- are **tightly coupled** with specific transformation languages,
- use **black-box testing**: they do not leverage the definition of the transformation to improve results.

Open Challenge

There is thus a need for a **language-generic**, **white-box** MTT framework.

MTT: A Use-Case for Model Testing?

Observation

Model transformation languages are **executable domain-specific languages** (xDSLs), and model transformations are **models**.

Enabling testing for any xDSLs is critical to the success of MDE:

- It allows to design test oracles using domain concepts.
- It enables fault localization at design time.

Generalized Challenge

Why not aim for a language-generic white-box **model testing** framework and use MTT as a **use case**?

Imagine a generic and improved JUnit for models of any language!

MTT: A Use-Case for Model Testing?

Observation

Model transformation languages are **executable domain-specific languages** (xDSLs), and model transformations are **models**.

Enabling testing for any xDSLs is critical to the success of MDE:

- It allows to design test oracles using domain concepts.
- It enables fault localization at design time.

Generalized Challenge

Why not aim for a language-generic white-box **model testing** framework and use MTT as a **use case**?

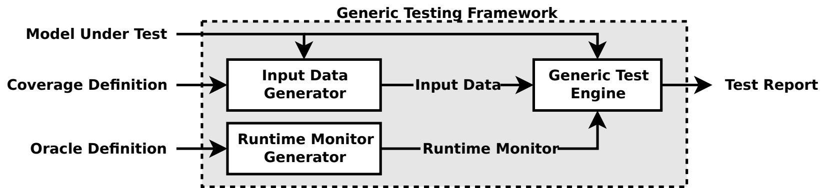
Imagine a generic and improved JUnit for models of any language!

Informal Recipe for Generic Model Testing

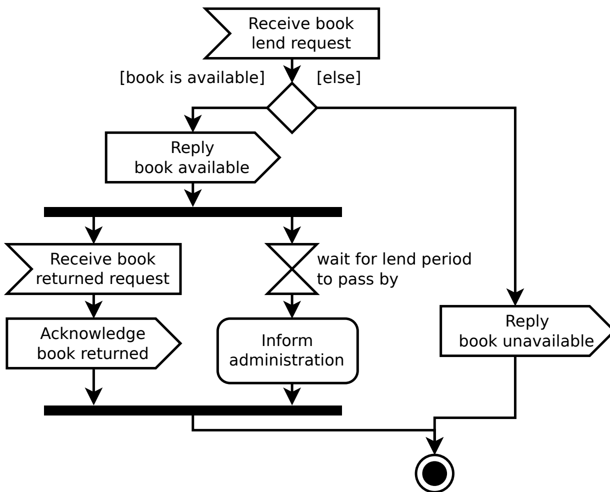
What is needed for model testing:

- language-generic execution engine,
- mean to interact with executed models,
- generator of input test data,
- way to specify temporal properties,
- approach to use them as oracles,
- manner to track failures back to their source.

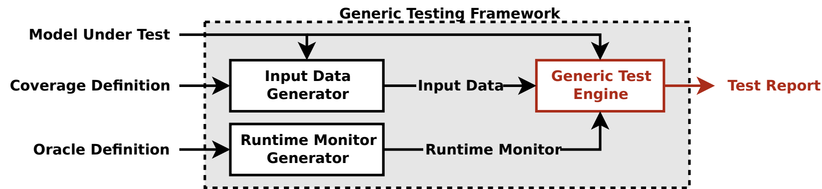
Proposed Solution



Running Example: Library Activity Diagram



Generic Testing Engine



Required Features

A generic testing engine should be able to:

- execute models conforming to executable DSLs,
- enable interaction with external actors,
- monitor executions to validate oracles and build test reports.

State of the Art: the GEMOC Studio

The approach to language engineering implemented as part of the GEMOC Studio ticks two of these boxes:

- it enables model execution for any DSL,
- it can be extended with engine add-ons, that can be used (among other purposes) to monitor the execution.

However, interaction with the environment of executed models is missing.

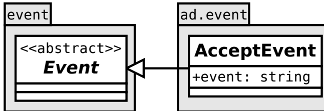
Defining the Behavioral Interface of DSLs

To enable interaction with external actors, we propose an approach to define the behavioral interface of DSLs including:

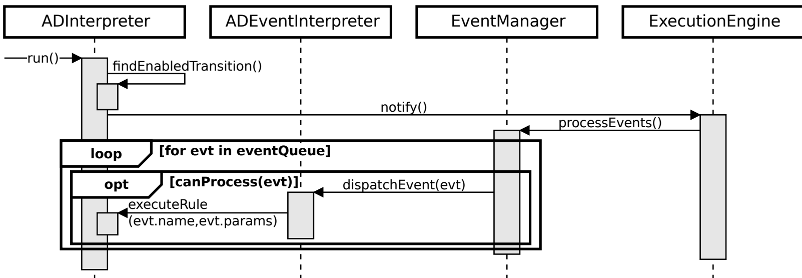
- non-intrusive annotation of execution rules,
- generation of an interface to inject event occurrences,
- reuse of an event queue and interruption mechanism.

Defining the Behavioral Interface of DSLs

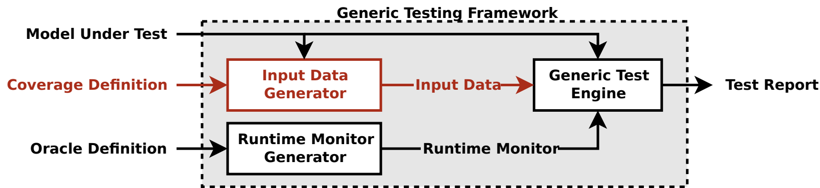
Generated event metamodel for Activity Diagram:



Event queue and interruption mechanism:



Test Data Generation



White-Box Model Testing

Definition

In white-box model testing, we have access to the **model under test** as well as the **abstract syntax**, **execution semantics** and **parameter metamodel** of the language it conforms to.

Test Coverage Definition

Problem statement

How can we perform white-box model testing in a generic way?
How can we leverage it to improve model testing results?

Proposed solution:

- Combine coverage types to generate relevant test data (*e.g.*, model element coverage, external stimuli coverage).
- Allow the user to provide customized coverage definition to fine-tune generated input data.

Research questions

Does it yield interesting results for various DSLs, including model transformation languages?

Test Coverage Definition

Problem statement

How can we perform white-box model testing in a generic way?
How can we leverage it to improve model testing results?

Proposed solution:

- Combine coverage types to generate relevant test data (*e.g.*, model element coverage, external stimuli coverage).
- Allow the user to provide customized coverage definition to fine-tune generated input data.

Research questions

Does it yield interesting results for various DSLs, including model transformation languages?

Test Coverage Definition

Problem statement

How can we perform white-box model testing in a generic way?
How can we leverage it to improve model testing results?

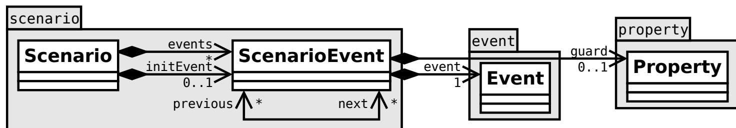
Proposed solution:

- Combine coverage types to generate relevant test data (*e.g.*, model element coverage, external stimuli coverage).
- Allow the user to provide customized coverage definition to fine-tune generated input data.

Research questions

Does it yield interesting results for various DSLs, including model transformation languages?

Generated Test Data

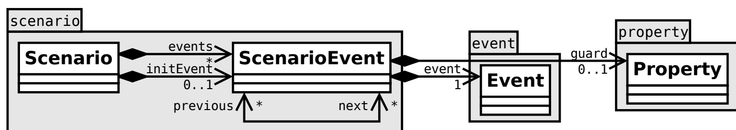


Takes the form of a scenario consisting of:

- an initialization event containing the input parameters,
- any number of subsequent events (possibly none).

For Library.ad:

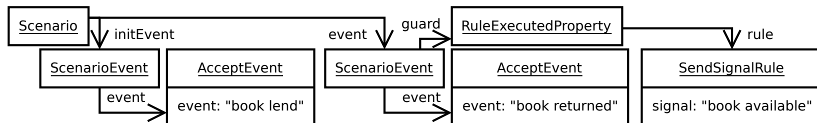
Generated Test Data



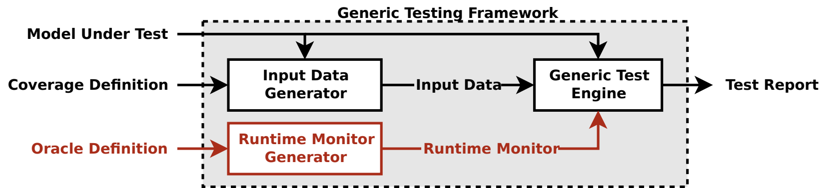
Takes the form of a scenario consisting of:

- an initialization event containing the input parameters,
- any number of subsequent events (possibly none).

For Library.ad:



Generic Temporal Property Language



Adapting PSL to MDE

Problem statement

How to check, on models conforming to any DSL, temporal properties on model state and events?

First, a look at the Property Specification Language (PSL):

- Temporal logic extending LTL
- Used in the hardware design and verification industry
- Boolean layer compatible with several Hardware Description Languages (HDLs)

Adapting PSL to MDE

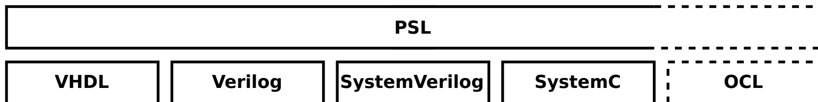
Problem statement

How to check, on models conforming to any DSL, temporal properties on model state and events?

First, a look at the Property Specification Language (PSL):

- Temporal logic extending LTL
- Used in the hardware design and verification industry
- Boolean layer compatible with several Hardware Description Languages (HDLs)

Adapting PSL to MDE



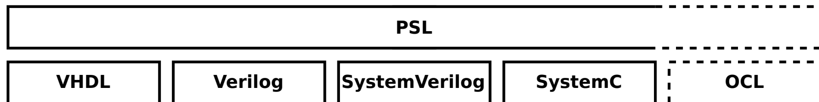
Proposed solution:

- Use OCL's operators in PSL's boolean layer
- Allow to express properties on execution rules scheduling
- Automatically derive runtime monitors as state machines

Research questions

Is the language expressive enough? How significant is the overhead of the monitors?

Adapting PSL to MDE



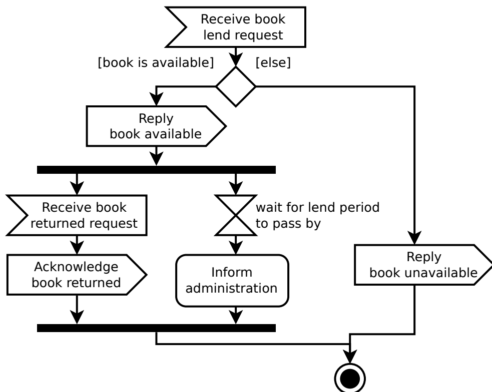
Proposed solution:

- Use OCL's operators in PSL's boolean layer
- Allow to express properties on execution rules scheduling
- Automatically derive runtime monitors as state machines

Research questions

Is the language expressive enough? How significant is the overhead of the monitors?

Example Property for Library




```
SendSignal("book available") |=>
        (AcceptEvent("book returned") | Action("Inform administration"))
```

Publications Wishlist

- Leveraging Operational Semantics to Generate the Behavioral Interface of Executable DSLs (Re-submission planned)
- A Generic Temporal Property Language for Runtime Monitoring your Executable DSLs (WIP)
- A DSL for Test Coverage Metrics Definition and Test Input Data Generation (TODO)
- An End-to-end Generic White-box Testing Framework for Executable DSLs (TODO)

Conclusion

Model transformation languages as executable DSLs:

- Is a generic model testing framework suitable for MTT?
- If not, what is the minimal addition to make it suitable?
- Can model testing benefit from MTT's specificities?

Language-generic white-box testing:

- Can language-generic white-box testing be made efficient?
- What should be provided to improve it for a given DSL?