A Testing Framework for Executable Domain-Specific Languages

Faezeh Khorram
IMT Atlantique

Reporters *:*

- Prof. Anne ETIEN, Université de Lille, France
- Prof. Juergen DINGEL, Queen's University, Canada

Examiners:

- Prof. Benoît BAUDRY, KTH Royal Institute of Technology, Sweden
- Dr. Javier TROYA, Universidad de Málaga, Spain
- Prof. Antoine BEUGNARD, IMT Atlantique, France

Thesis director*:*

- Prof. Gerson SUNYE, Nantes Université, France

Thesis supervisors:

- Dr. Jean-Marie MOTTU, Nantes Université, France
- Dr. Erwan BOUSSE, Nantes Université, France

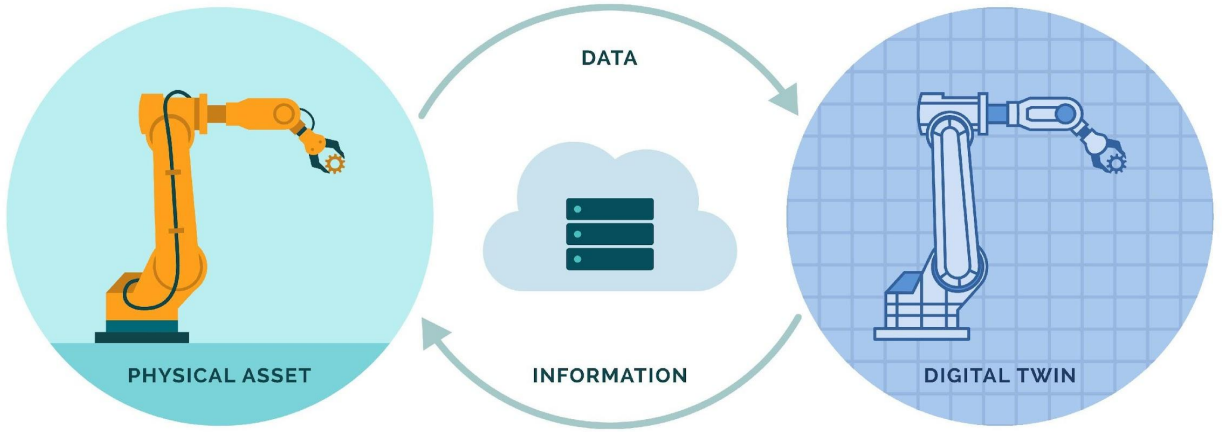
Outline

- Introduction
- State-of-the-art
- Contribution & Evaluation
- Conclusion & Perspectives

Introduction

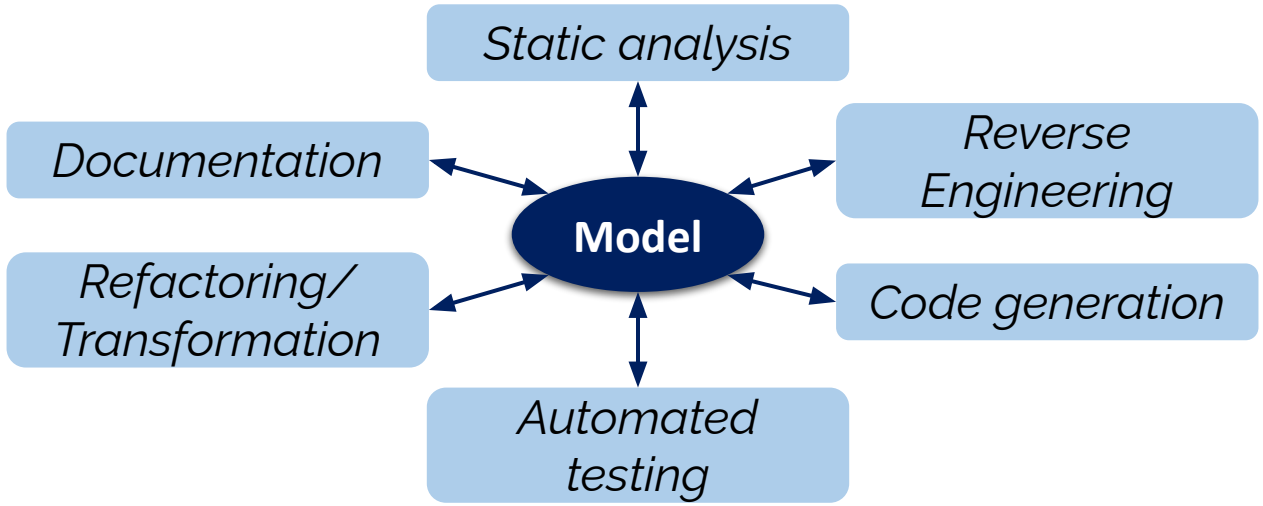
How to develop ***complex*** software?

- Dealing with complexity of the application domain
- Involving the stakeholders with diverse knowledge and expertise in the development lifecycle



Model-Driven Engineering (MDE)

- A software development paradigm
- Uses models as the central artifact of software development



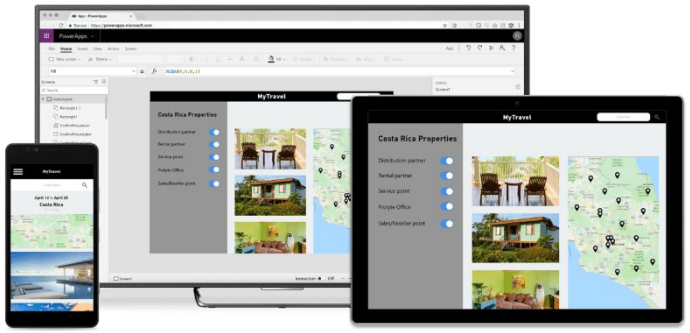
Model

- an abstraction of an original system for a specific purpose
- reflects a relevant selection of the system's properties
- usable in place of the system with respect to some purpose

Domain-Specific Language (DSLs)

- Languages for the definition of models,
- made for specific technical or application domains,
- tailored to be used by domain experts.

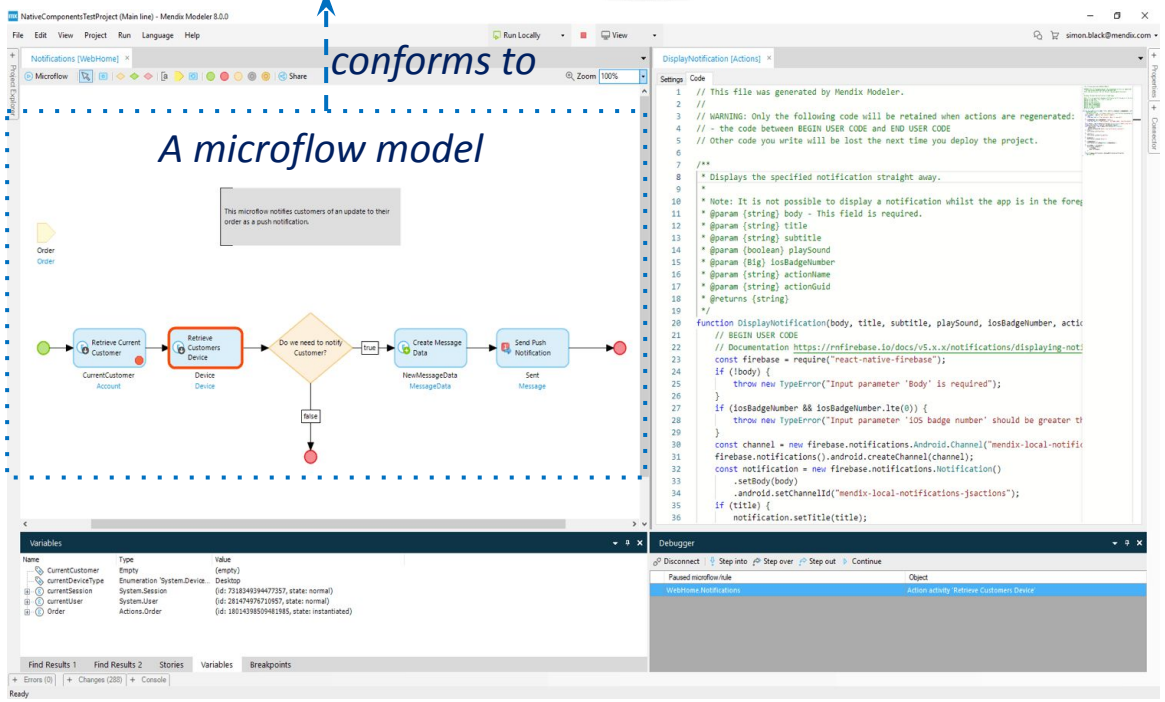
Low-Code Development Platforms (LCDPs)



Mendix Microflow DSL

conforms to

A microflow model

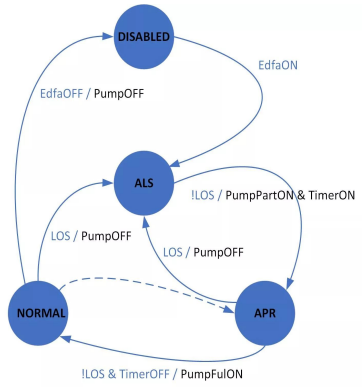


Behavioral models

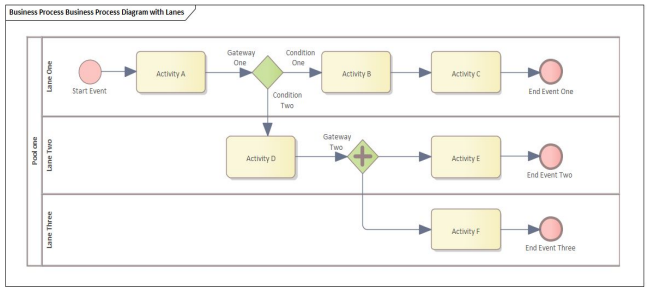
- Representing ***dynamic*** aspects of a system
- How to make sure if the model behaves correctly?

We need to execute the model

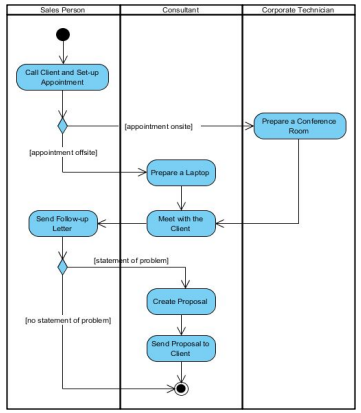
State machine



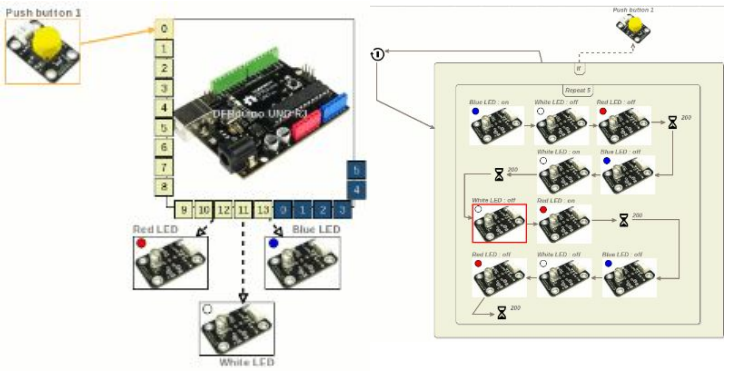
Business process



Activity diagram

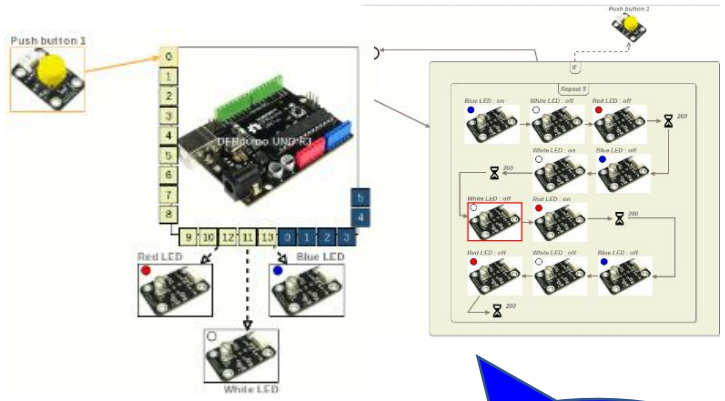


Arduino model



Executable Models

- Models can be executed if the DSL provides an execution semantics

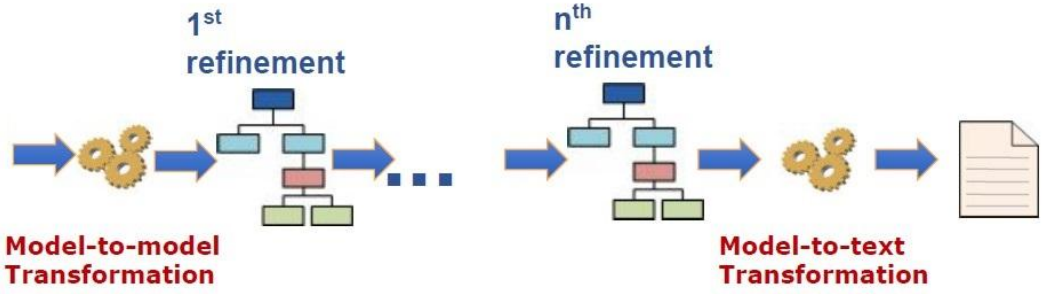


Translational semantics

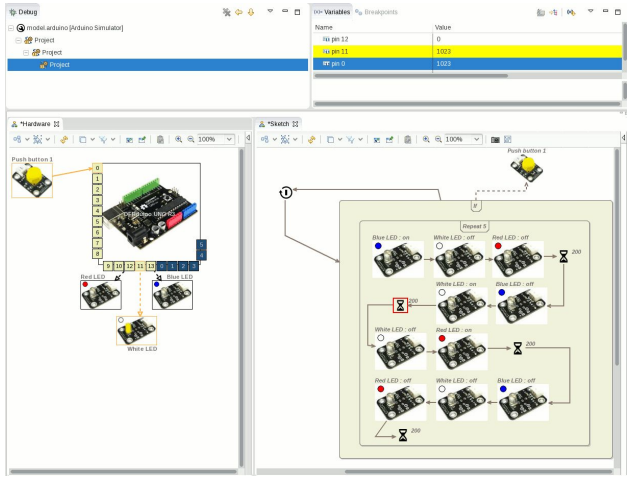
Operational semantics

How to execute it?

*Running the code generated by a **compiler***



*Running the model itself using an **interpreter***



Early Dynamic Verification and Validation (V&V) of models

Dynamic Verification of Executable Models

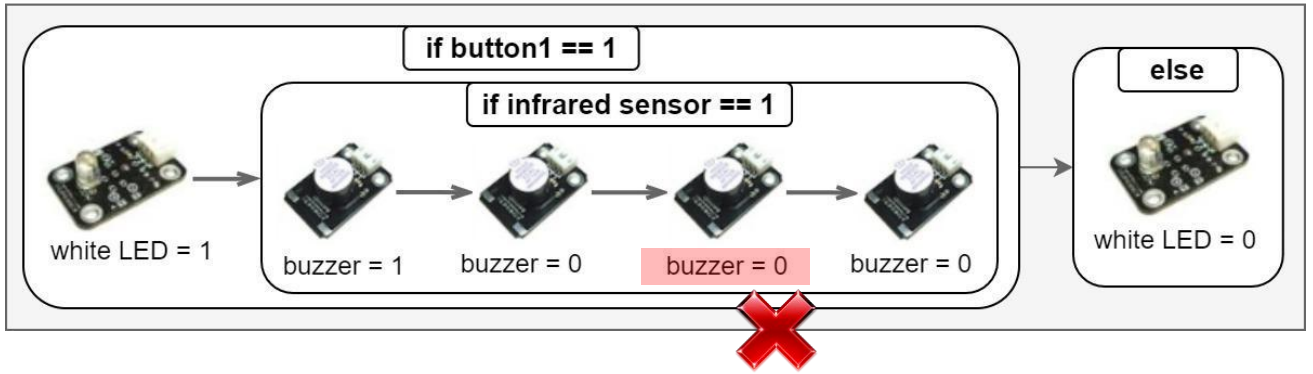
- What if the model execution is not as expected?

There is a defect in the model

- How to find the cause of un-expected behavior?

Testing executable models

Running example: A basic intrusion alarm system



Testing Executable Models

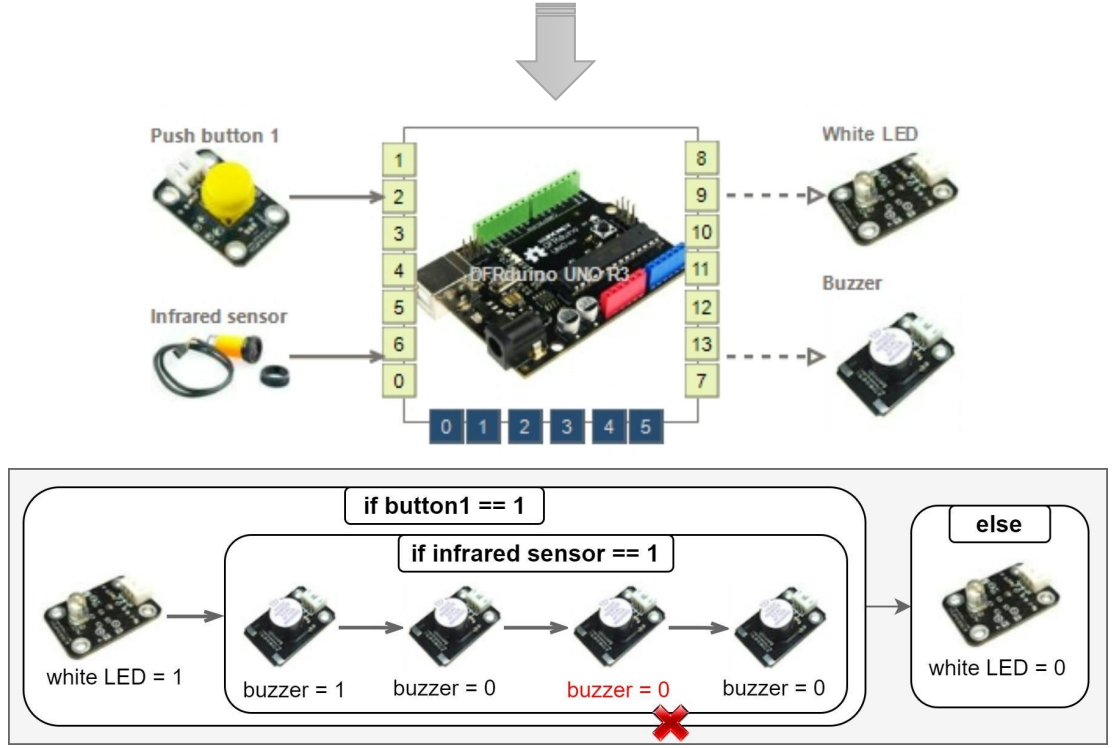
- Testing is the primary method used for evaluating software systems

Testing involves:

- executing systems in interesting scenarios,
- observing whether they act as expected.

How to define input and output?

Input: button pressed, sensor detected



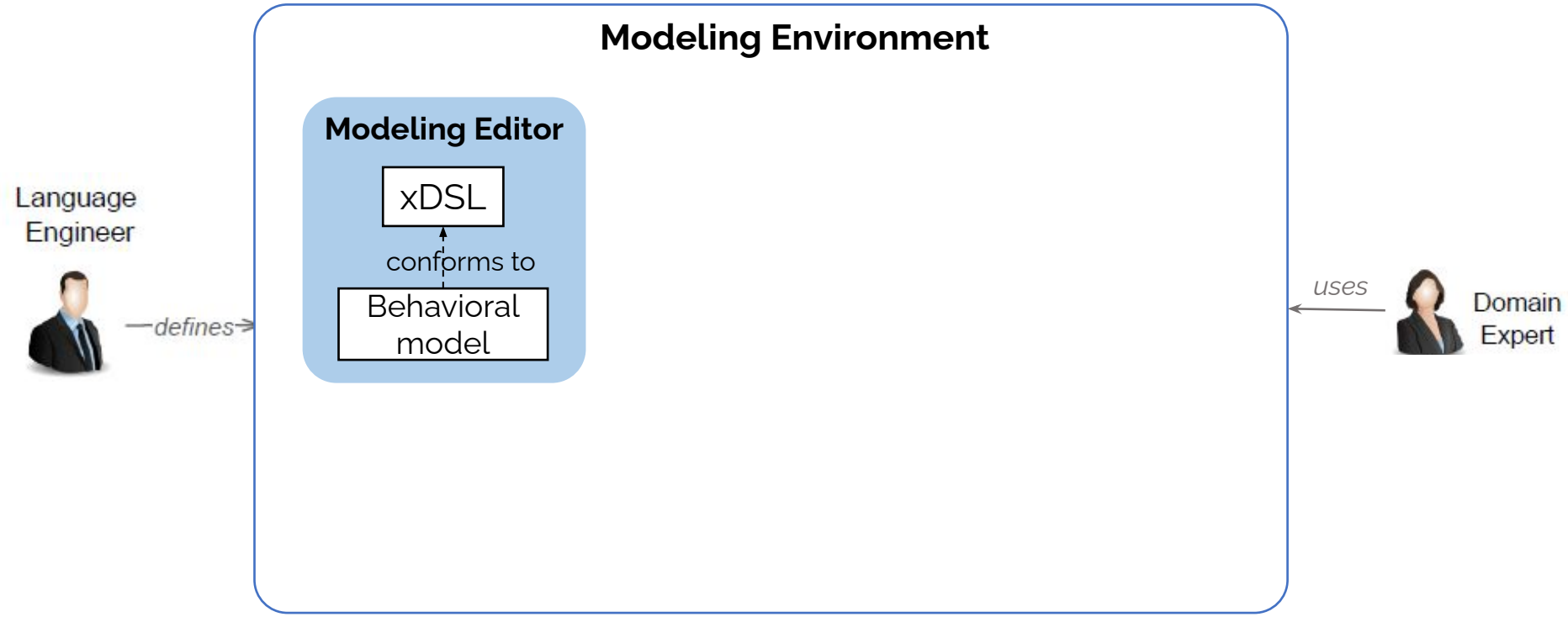
Expected output: LED turned on, buzzer turned on and off 2 times

Execution output: LED turned on, buzzer turned on and off once

Problem Statement

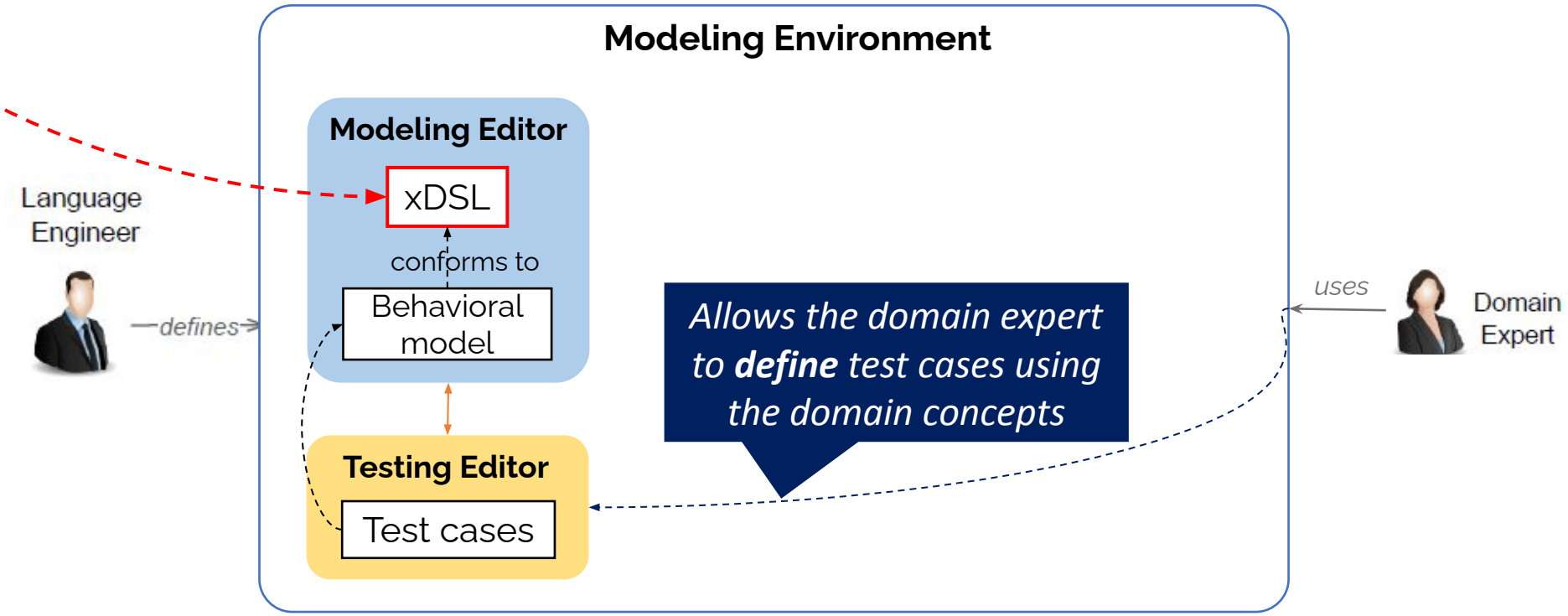
Roles:

- *Language Engineer:* defines an xDSL and tools for using it
- *Domain Expert:* user of the xDSL



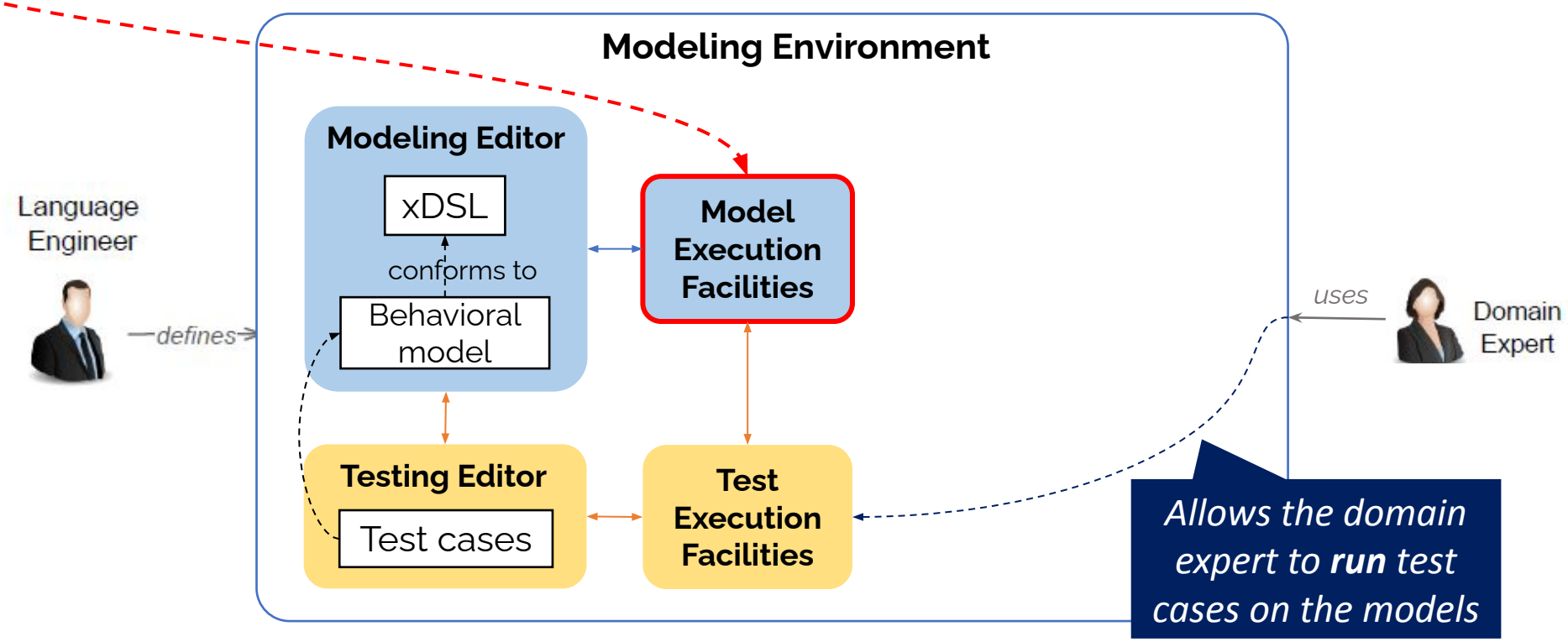
Problem Statement

- Given an xDSL, its domain experts can test the conforming models if the **domain concepts** can be used in the specification of test cases
- **Challenge#1**: Domain concepts differ from one xDSL to another



Problem Statement

- Written test cases must be executed in unison with the models under test
- Test execution must be somehow connected to the model execution
- **Challenge#2**: Model execution facilities differ from one xDSL to another

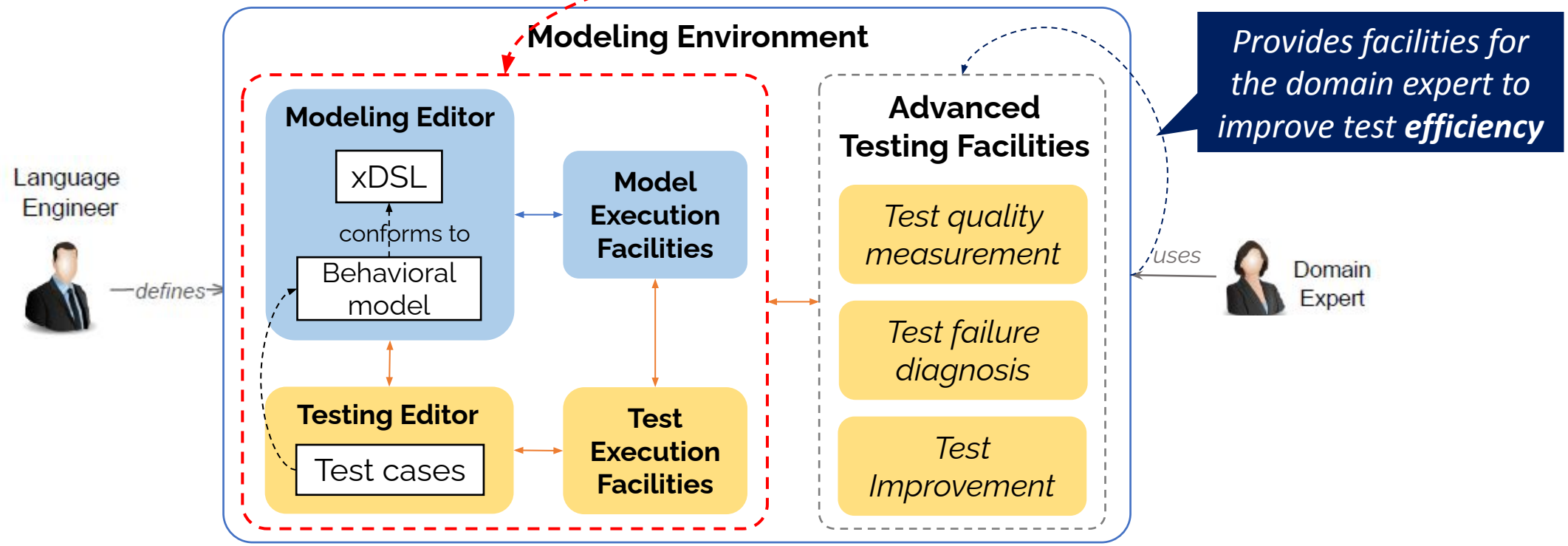


Problem Statement

Providing facilities to improve test efficiency:

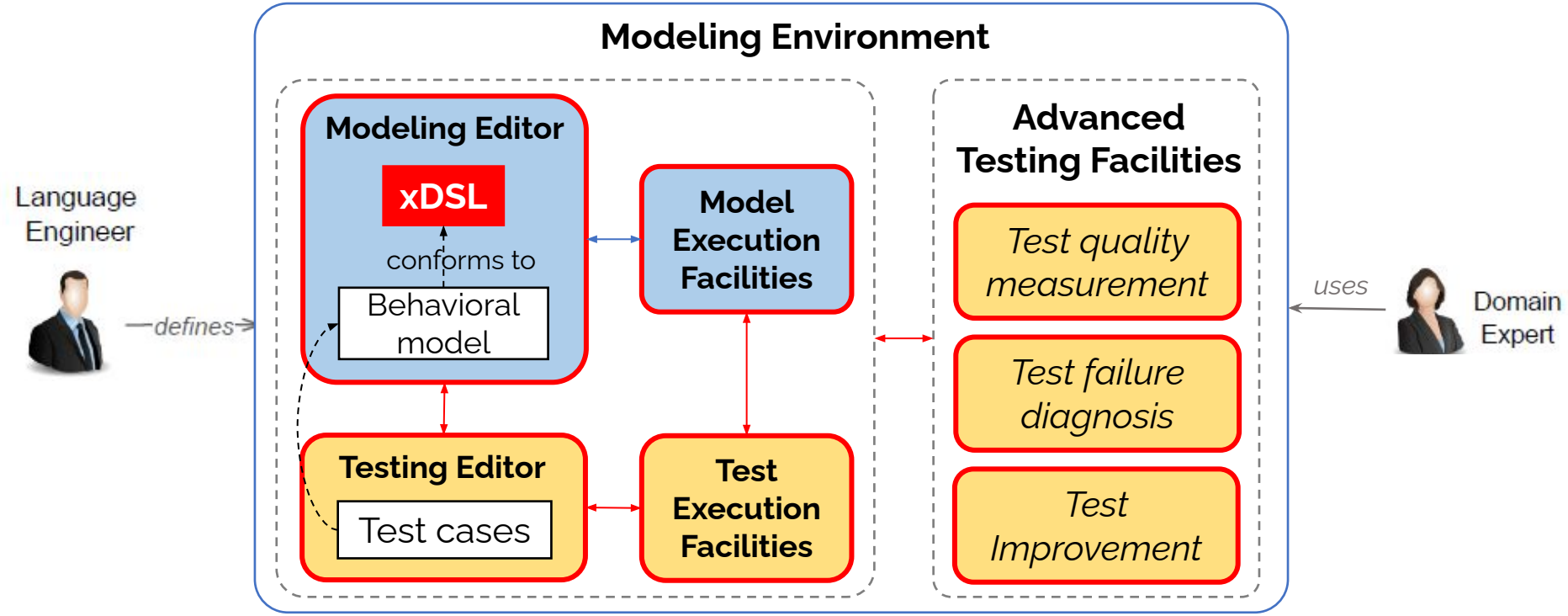
- Evaluating whether the written test cases are good enough
- Diagnosing the faults when test cases fail on a model
- Improving the strength of the written test cases

Challenge#3: dependency to testing frameworks as they need to directly manipulate test cases and their system under test



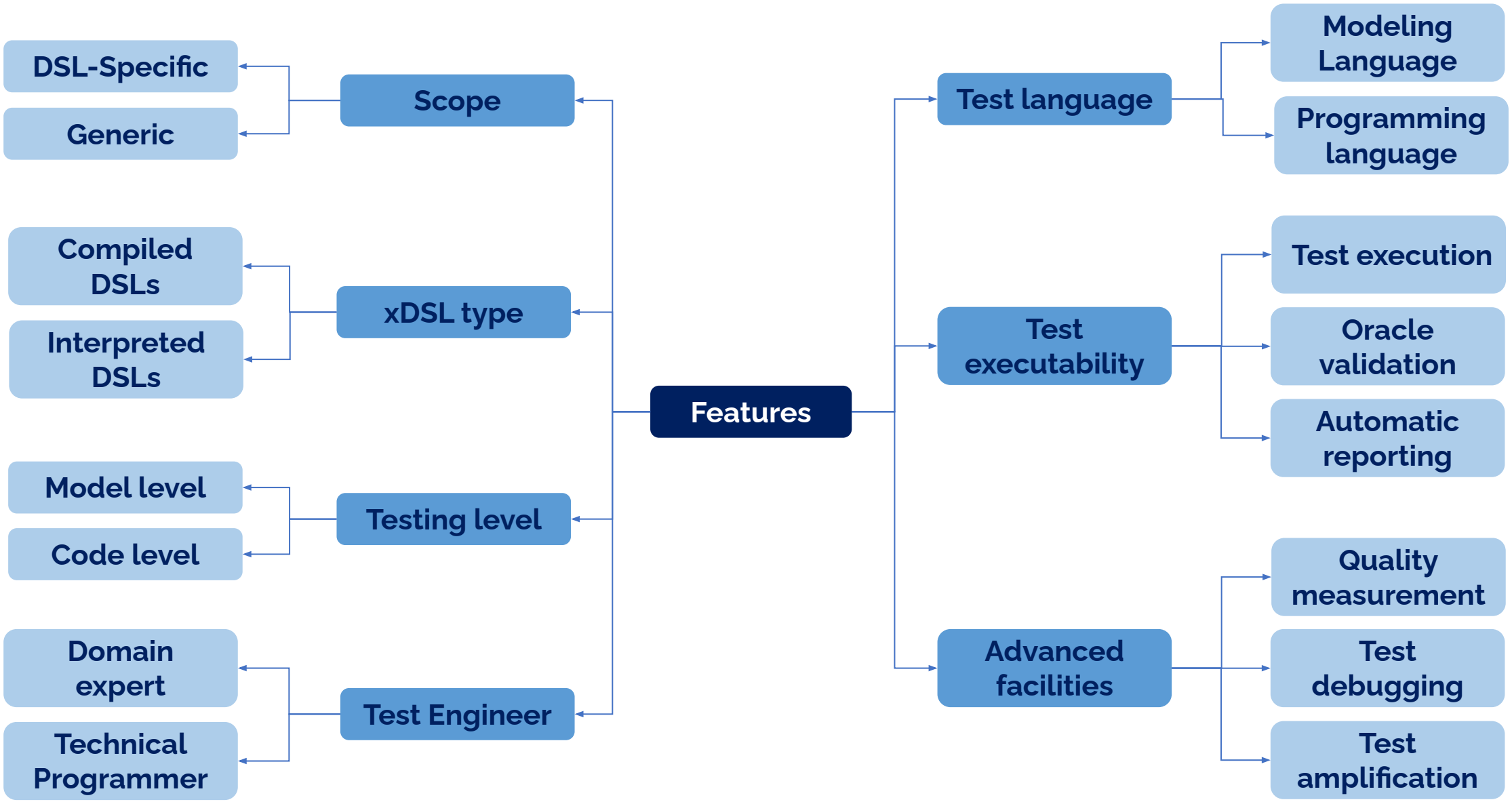
Problem Statement:
Diversity and Heterogeneity of xDSLs

- New xDSL ⇒ new domain concepts, new execution facilities
- Each time a new xDSL is engineered, a new testing framework must be created from scratch



Solution: a *systematic* approach to provide testing support for *every* given xDSL

State-of-the-art: Considered features



State-of-the-art: 13 approaches

Required features to overcome said challenges

Paper	Scope		xDSL type		Testing level		Test engineer		Test language		Test executability			Advanced facilities		
	DSL-Specific	Generic	Compiled DSLs	Interpreted DSLs	Model-level	Code-level	Domain expert	Technical programmer	Modeling language	Programming language	Test execution	Oracle validation	Automatic reporting	Quality measurement	Test debugging	Test amplification
Mens et al. [103]	•			•		•		•		•	•	•	•		•	
Iqbal et al. [75]	•		•		•		•		•	•	•	•	•	•	•	
Hili et al. [71]	•		•			•		•		•	•				•	
Santiago et al. [124]	•		•		•		•		•		•	•			•	
Kos et al. [87]	•		•		•		•		•		•	•	•		•	
Lubke and Van Lessen [97]	•			•	•		•		•		•	•	•			
Lazar et al. [91]	•			•	•		•		•		•	•	•			
Arnaud et al. [18]	•			•	•		•		•		•	•	•	•		•
Mijatov et al. [106]	•			•	•		•		•		•	•	•			
Iqbal et al. [74]	•		•			•		•		•	•	•	•	•		•
Wu et al. [152]		•	•			•		•		•	•	•	•		•	
Canizares et al. [36]		•	•	•	•		•		•		•					
Meyers et al. [105]		•		•	•		•		•		•	•				
Our Goal		*		*	*		*		*		*	*	*	*	*	*

Lack of reusability

Only for compiled DSLs

Only for metamorphic testing

No advanced facility

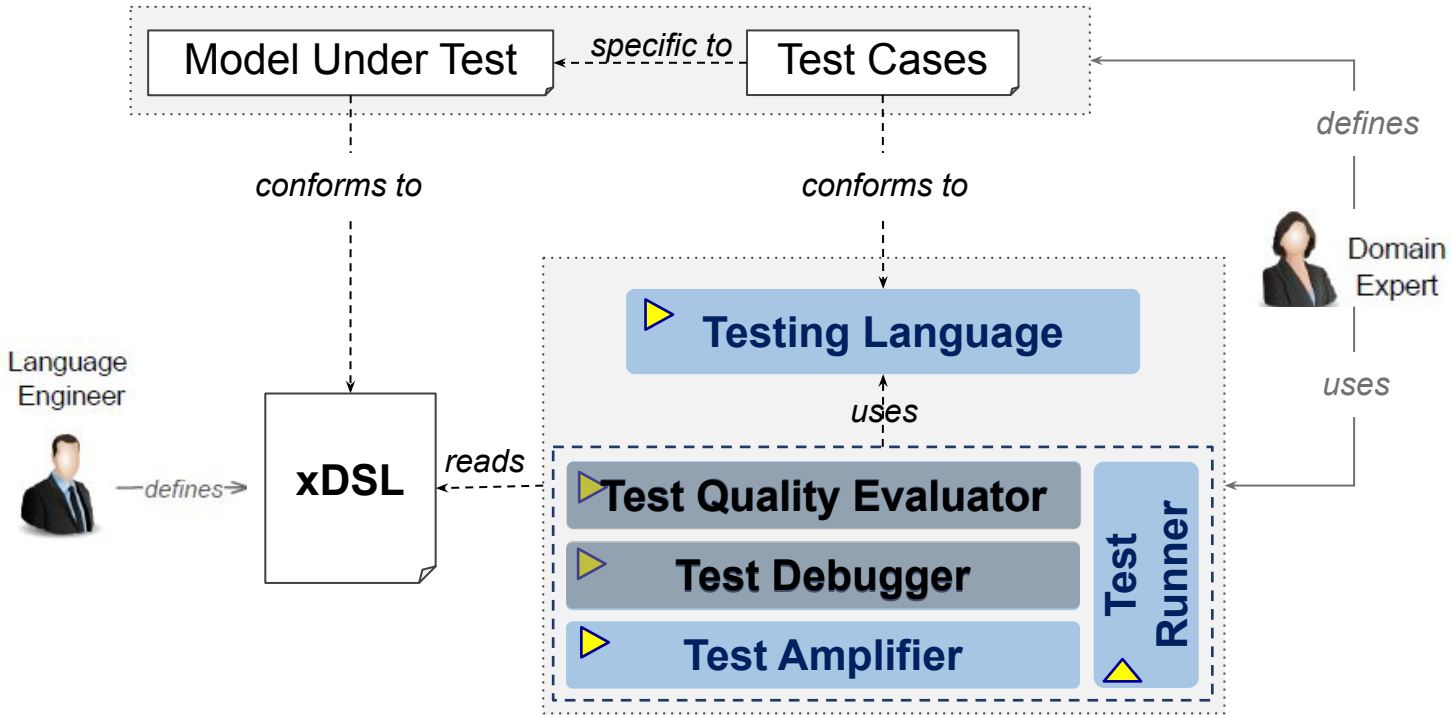
Proposal:
A generic testing framework for xDSLs

Users

- Enabling *language engineers* to provide testing support for their xDSLs
- Enabling *domain experts* to efficiently test behavioral models as early as possible

Contributions:

- Test case definition for models
- Test execution on models
- Test quality measurement
- Test debugging
- Test amplification for improving regression testing



blue == part of the defense presentation

Test Case Definition and Execution

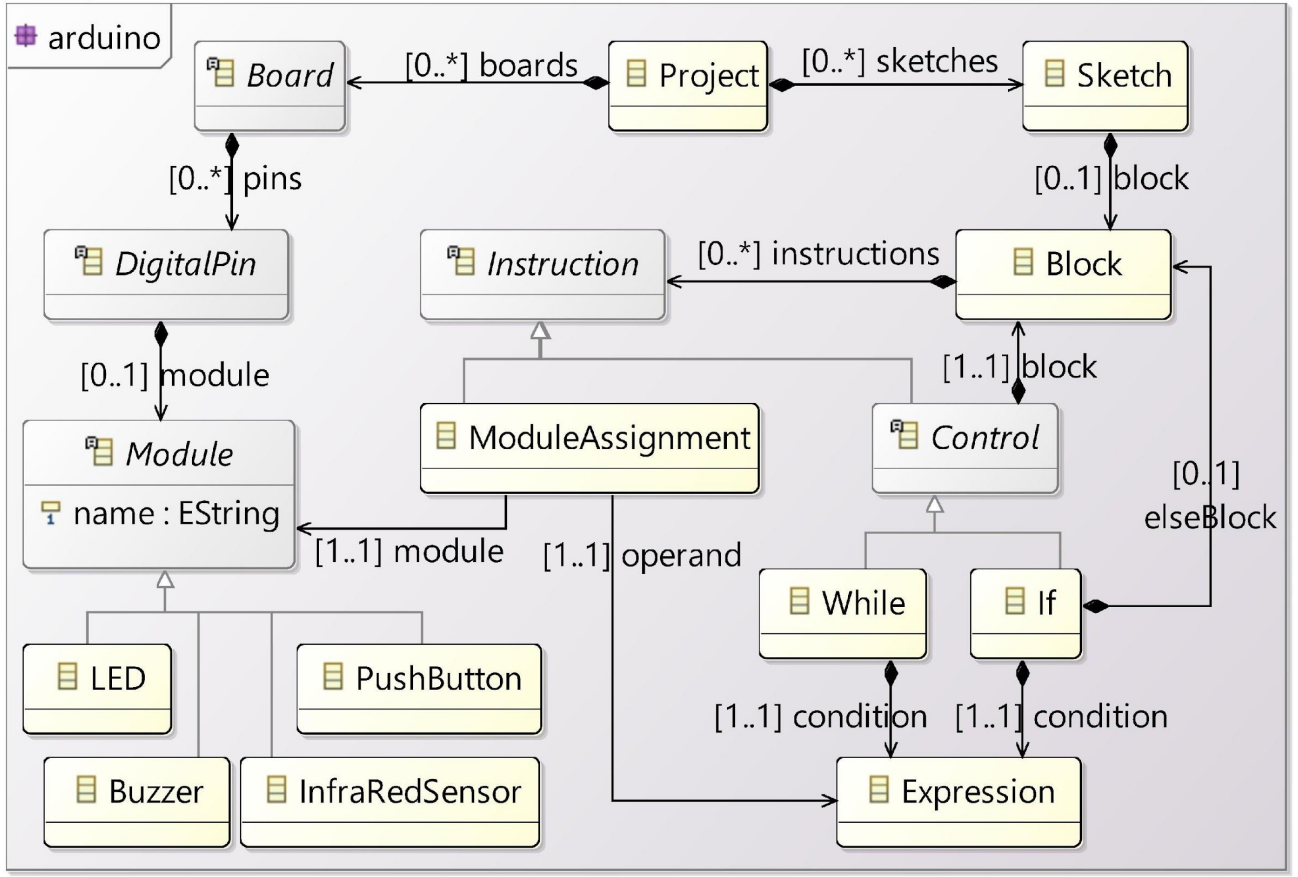
Chapter 3 of the manuscript

Executable Domain-Specific Languages (xDSLs)
- *Abstract Syntax*

Domain concepts and their relationships

- defined in an Ecore metamodel

Running Example: an xDSL for modeling and simulating Arduino boards and their behaviors (xArduino)

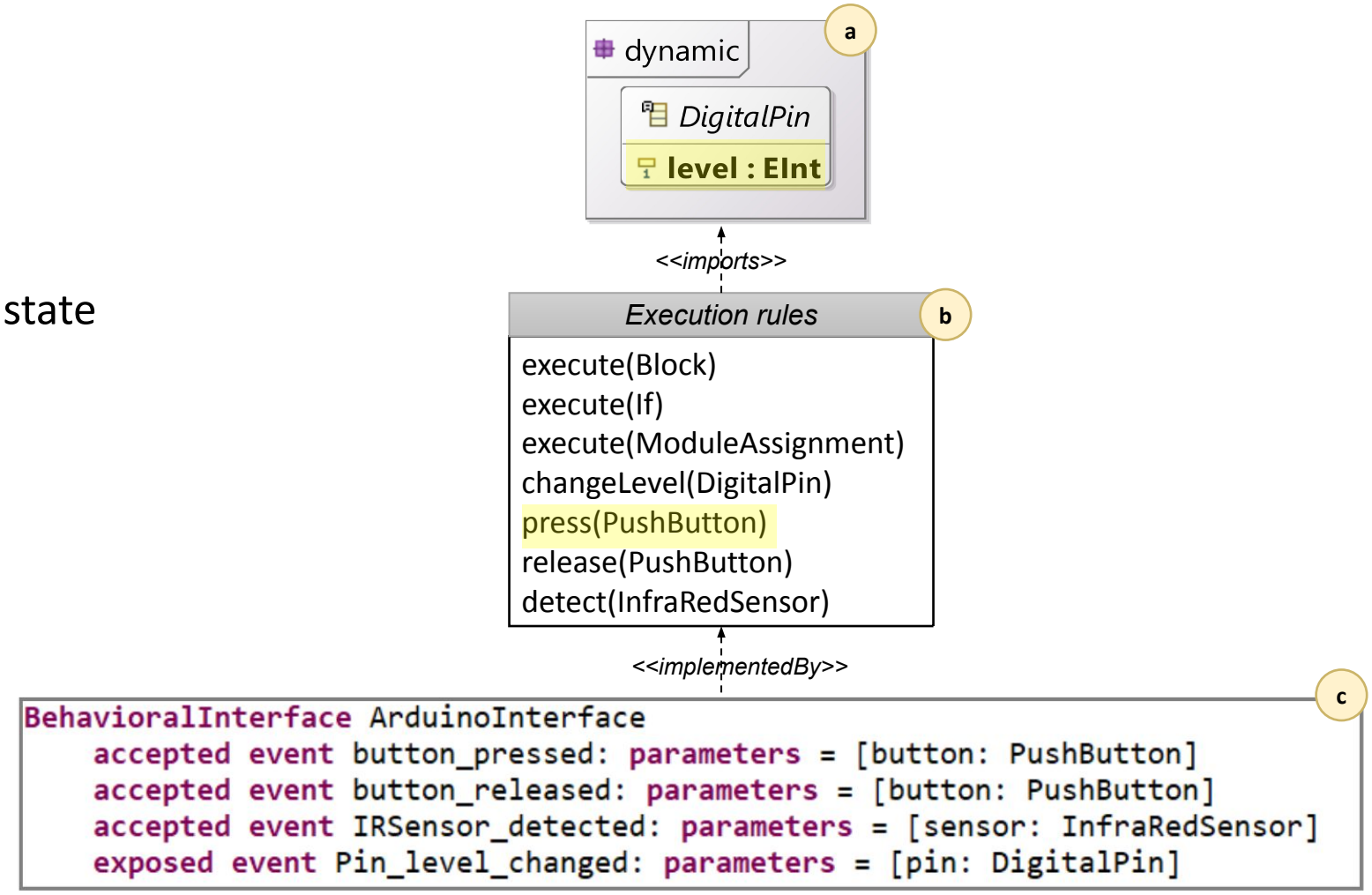


Executable Domain-Specific Languages (xDSLs)
- *Operational Semantics*

a **Definition of runtime state**

b **Execution rules**: changing runtime state to execute a model

c **Behavioral Interface (Optional)**: How to interact with a running model

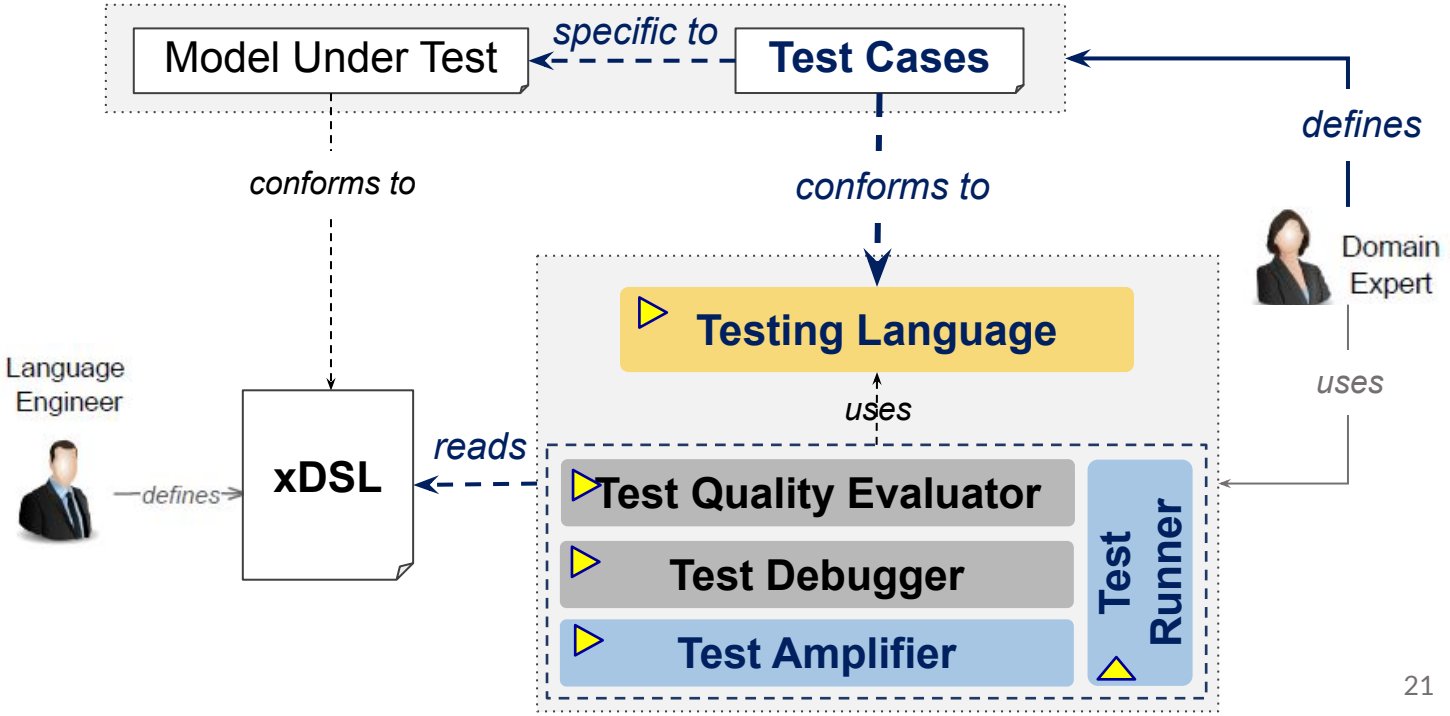


Defining Test Cases for Models

Question: How to enable the domain experts to write test cases for their behavioral models?

Answer: *providing a testing language that meets three requirements*:

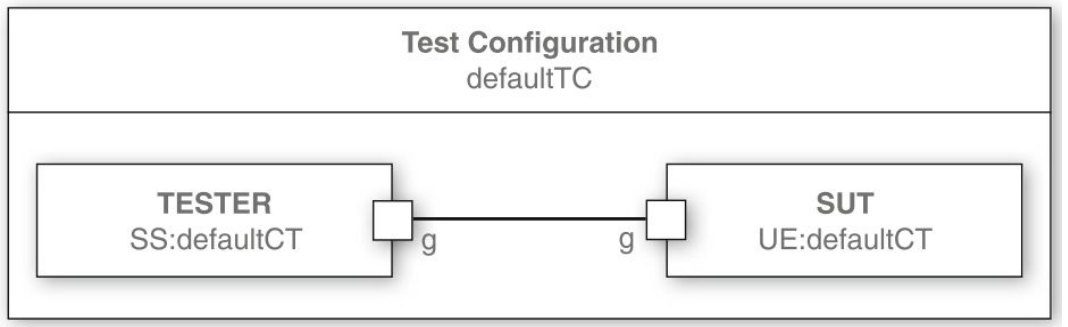
1) Allows the use of the **domain concepts** in defining test cases
2) Can launch the execution of the model under test
3) Provides facilities to investigate whether the model under test behaves as expected

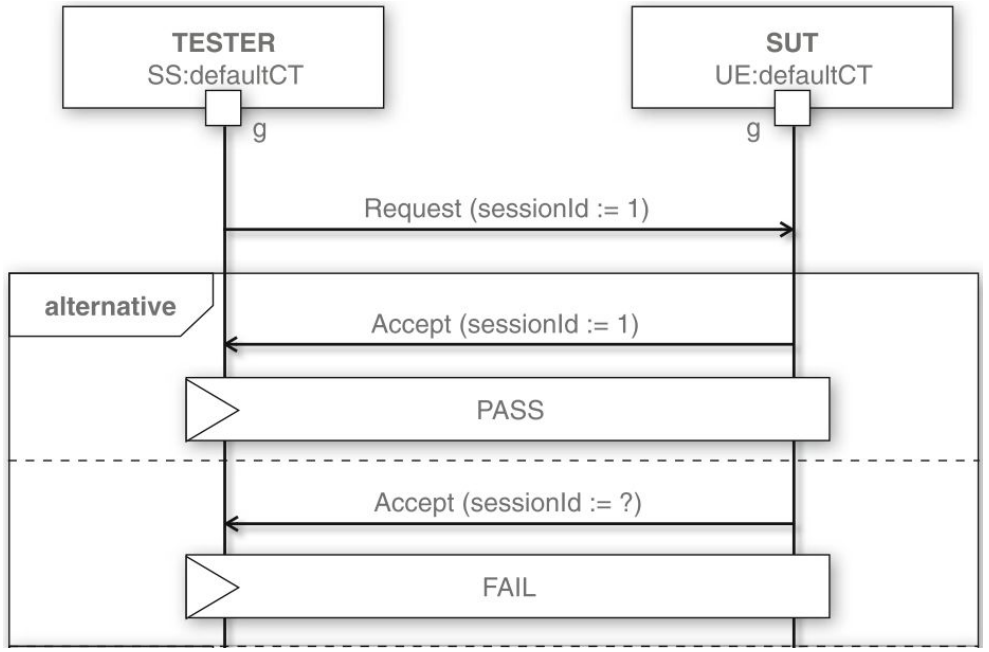


Candidate: Test Description Language [1]

Advantages of TDL:

- ✔ A standardized language for the specification of test cases
- ✔ Not specific to any language (GPL or DSL)
- ✔ Designed as a simple language for testers lacking programming knowledge, so a good fit for domain experts





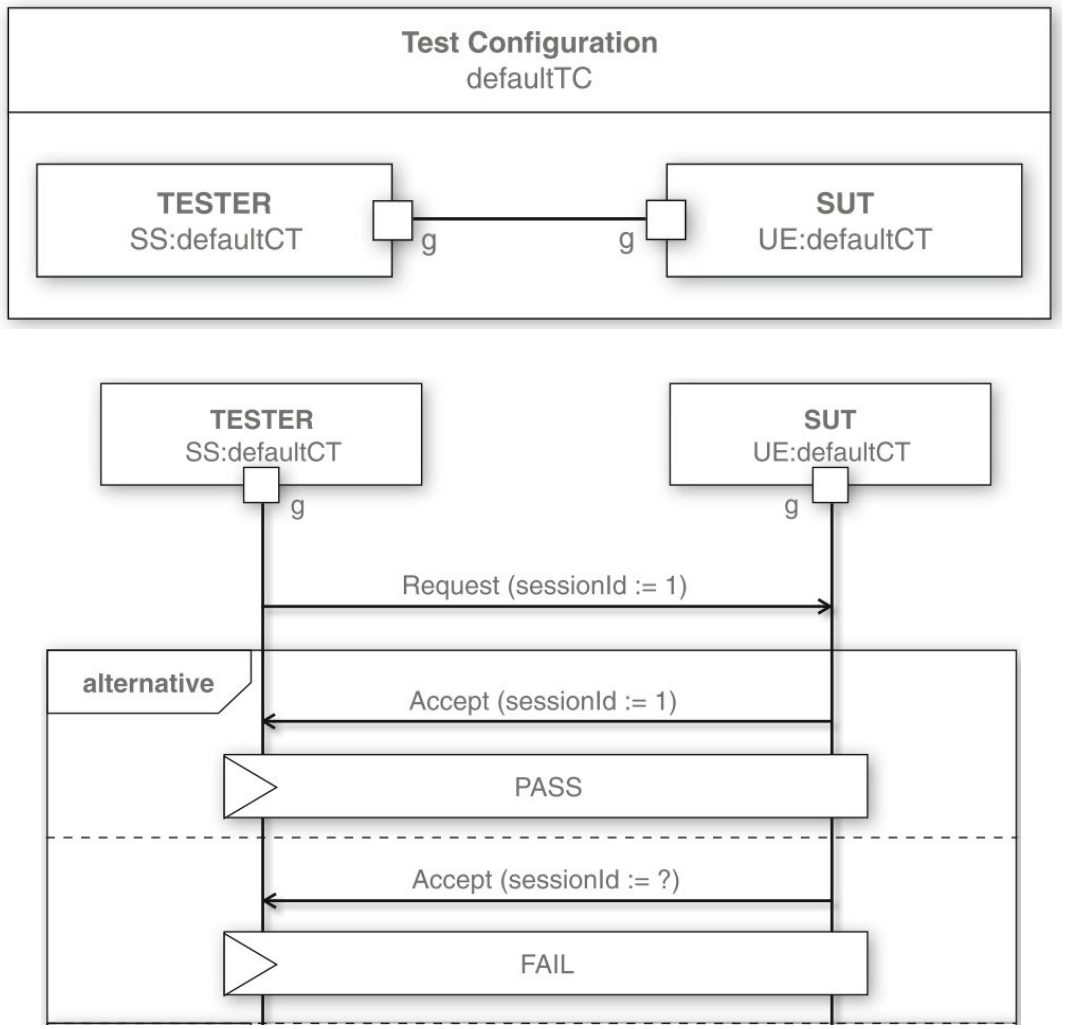
[1] Makedonski, P., Adamis, G., Käärik, M. et al. "*Test descriptions with ETSI TDL*". Software Quality Journal, vol. 27, no.2 , 885–917 (2019)

Candidate: Test Description Language [1]

Challenges:

- The domain expert must first manually define the required domain-specific concepts, and then write test cases
- No clear way to enable TDL test cases to execute models conforming to a DSL
- Relying on a simple representation of the expected behavior of the system under test

How to resolve the challenges of using TDL for testing models?



[1] Makedonski, P., Adamis, G., Käärik, M. et al. "*Test descriptions with ETSI TDL*". Software Quality Journal, vol. 27, no.2 , 885–917 (2019)

Contributions

Adapting the standardized Test Description Language (TDL) to the testing of executable models

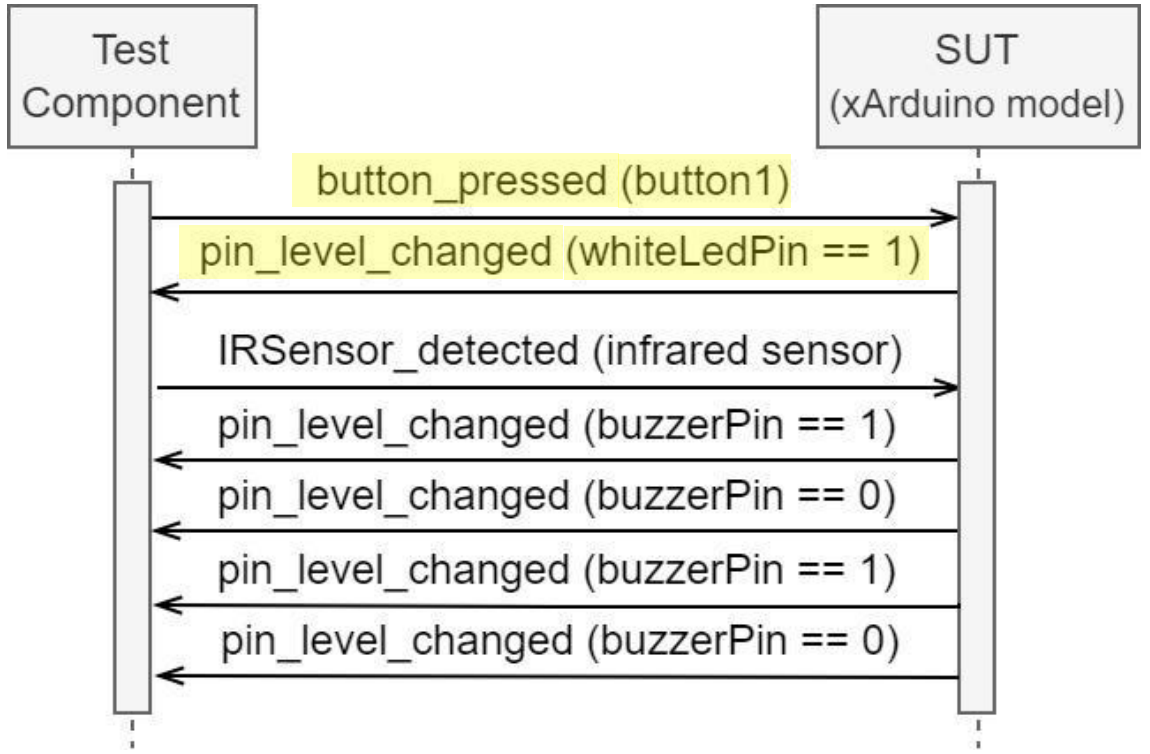
- **Cont#1. *TDL Library Generator:*** generating a domain-specific TDL library for each given xDSL to be used for writing test cases for the xDSL's conforming models
- **Cont#2. *TDL Interpreter*** : a test execution engine dedicated to running TDL test cases on executable models

An Example TDL Test Case for the xArduino model

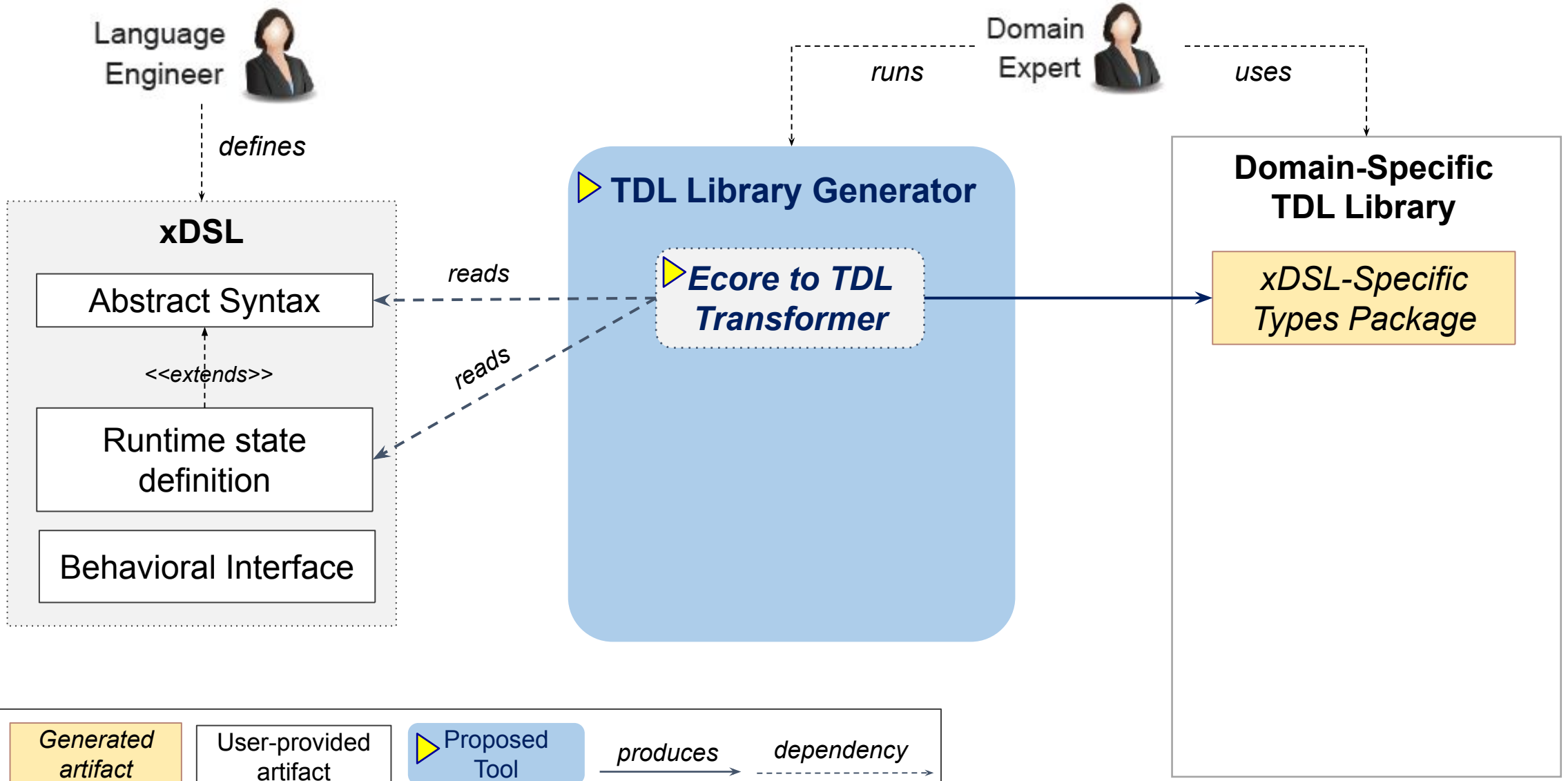
Using events of the behavioral interface and types of the abstract syntax and runtime state definition to define test data

- test input data and expected output are both a trace of events

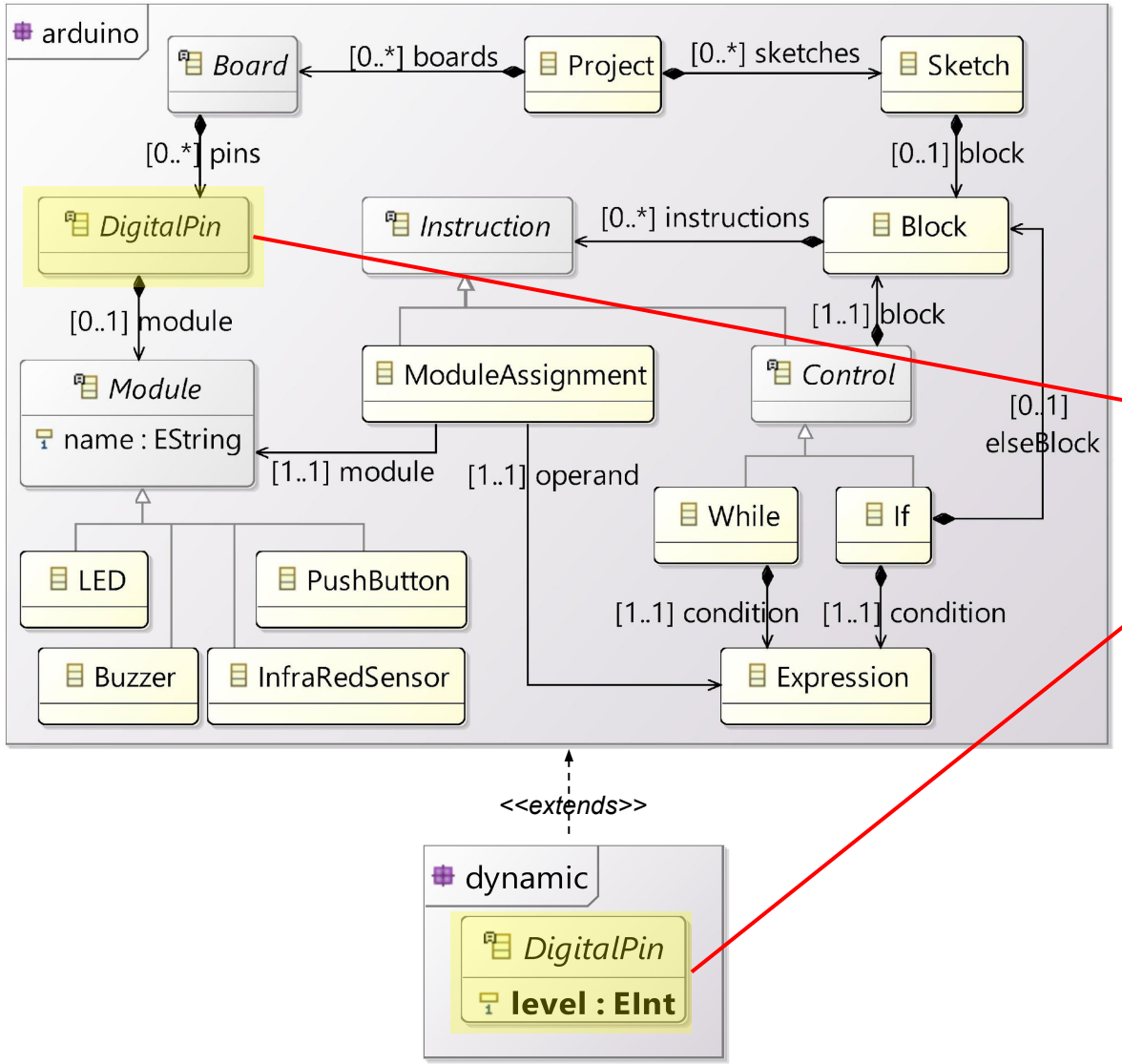
```
BehavioralInterface ArduinoInterface
    accepted event button_pressed:
        parameters = [button: PushButton]
    accepted event button_released:
        parameters = [button: PushButton]
    accepted event IRSensor_detected:
        parameters = [sensor: InfraRedSensor]
    exposed event Pin_level_changed:
        parameters = [pin: Pin]
```



TDL Library Generator

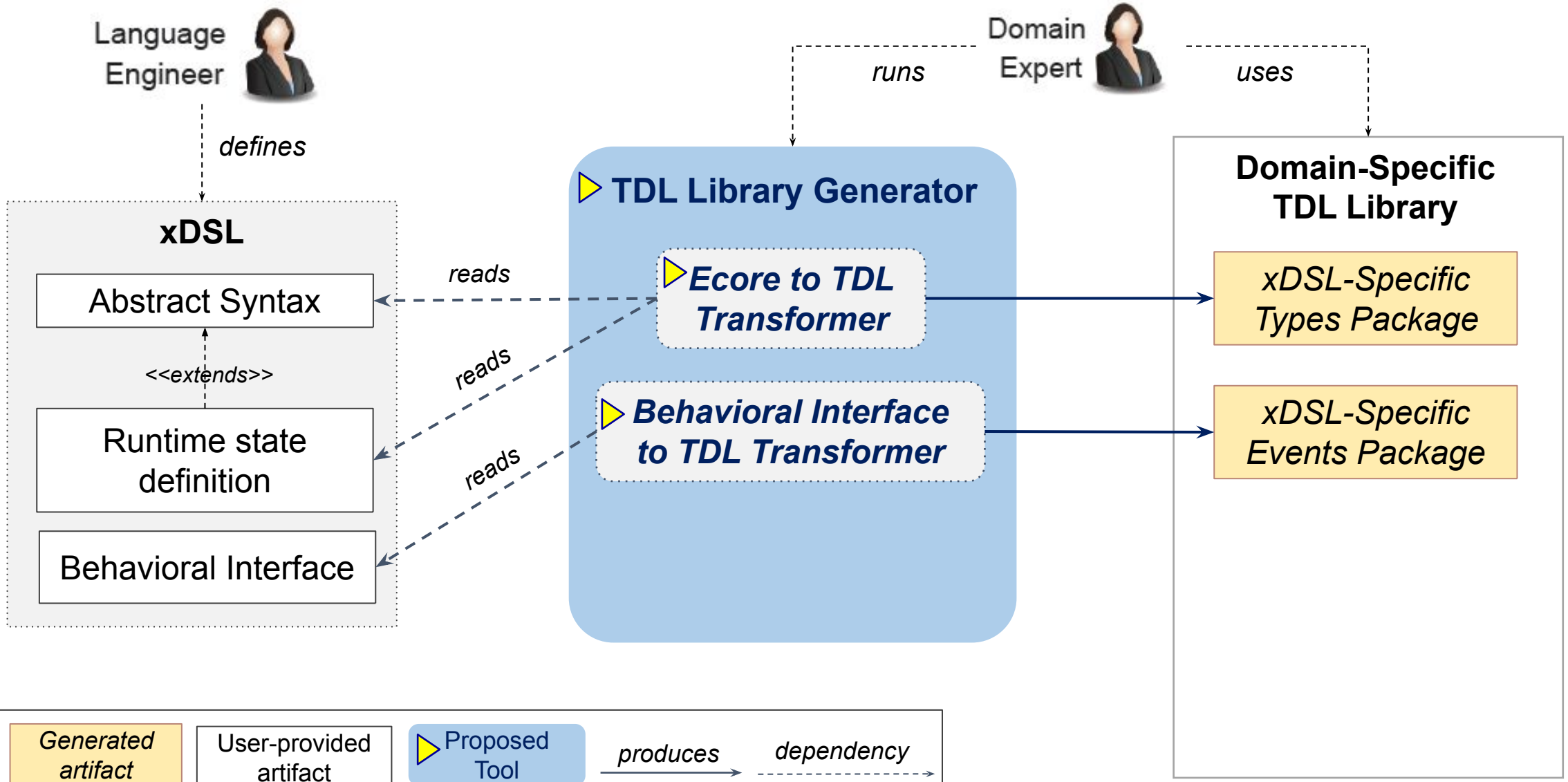


TDL Library Generator: *Ecore to TDL Transformation*




```
1  Package xArduinoTypes {
2      Type Project (
3          boards of type Board,
4          sketches of type Sketch
5      ) with {abstract;};
6      Type Board (pins of type DigitalPin) with {abstract;};
7      Type Sketch (block of type Block);
8      Type DigitalPin (
9          level of type EInt with {dynamic;},
10         module of type Module
11     ) with {abstract;};
12     Type Module (
13         _name of type EString
14     ) with {abstract;};
15     Type PushButton extends Module();
16     Type InfraRedSensor extends Module();
17     ...
18 }
```

TDL Library Generator

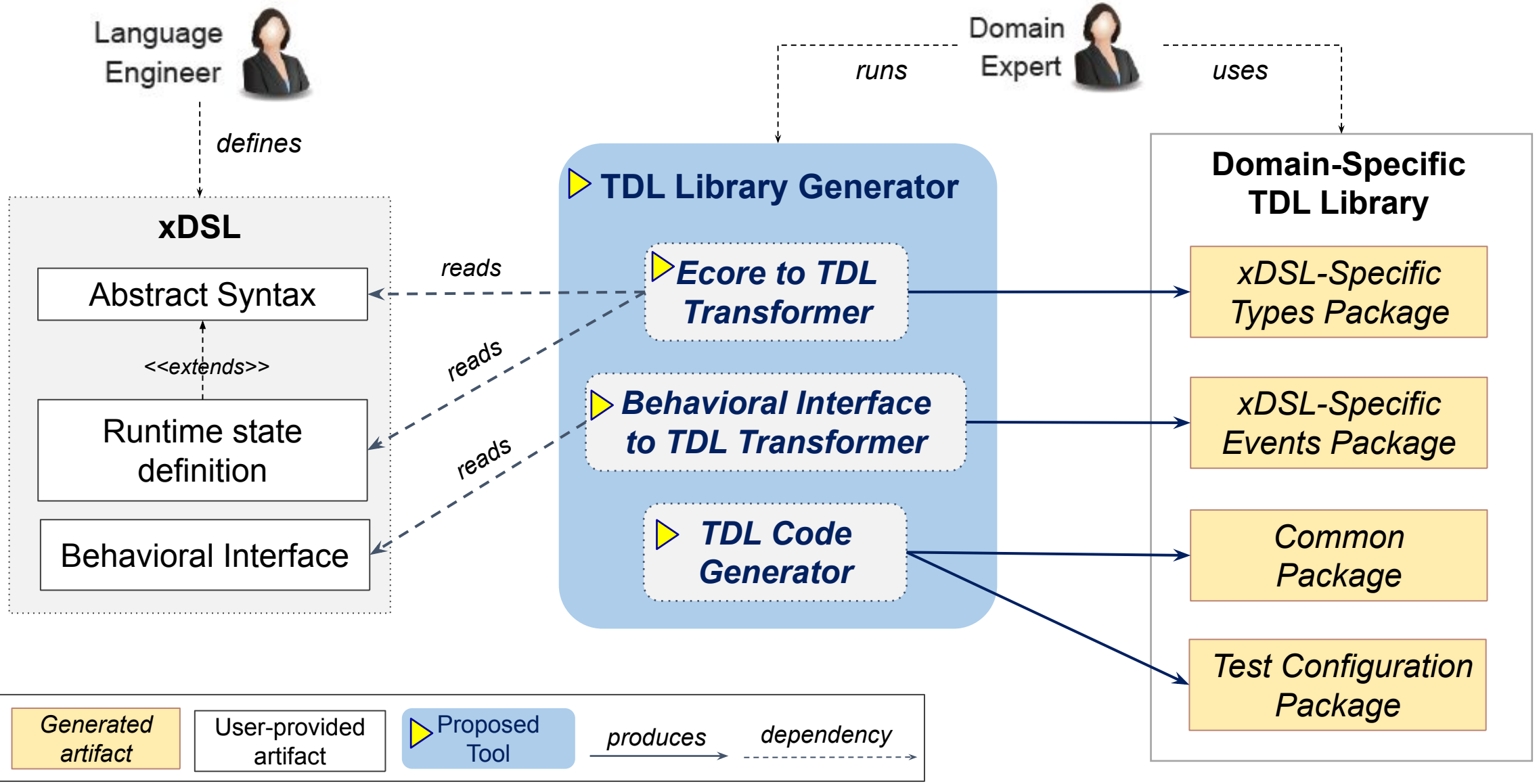


TDL Library Generator: *Behavioral Interface to TDL Transformation*


```
BehavioralInterface ArduinoInterface
    accepted event button_pressed:
        parameters = [button: PushButton]
    accepted event button_released:
        parameters = [button: PushButton]
    accepted event IRSensor_detected:
        parameters = [sensor: InfraRedSensor]
    exposed event Pin_level_changed:
        parameters = [pin: Pin]
```

```
1  Package xArduinoEvents {
2      Import all from xArduinoTypes_r;
3
4      Annotation AcceptedEvent;
5      Annotation ExposedEvent;
6
7      Type button_pressed (button of type PushButton)
8          with {AcceptedEvent;};
9      Type button_released (button of type PushButton)
10         with {AcceptedEvent;};
11     Type IRSensor_detected (sensor of type InfraRedSensor)
12         with {AcceptedEvent;};
13     Type pin_Level_Changed (pin of type DigitalPin)
14         with {ExposedEvent;};
15 }
```

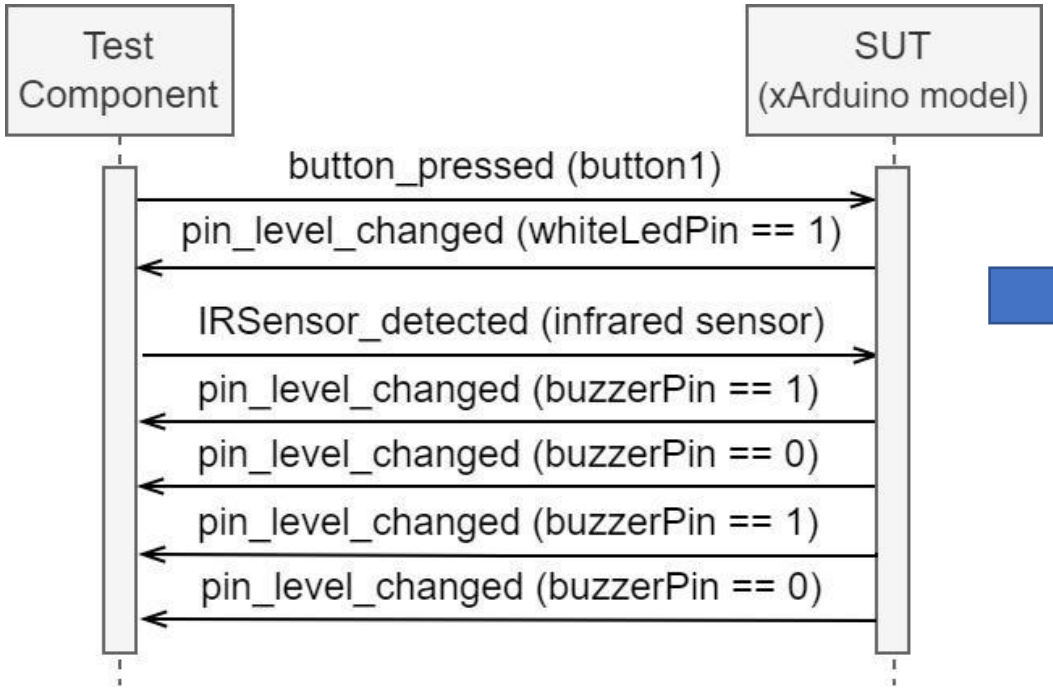
TDL Library Generator



Writing executable TDL test cases for models using the generated xArduino-specific TDL library

Importing the generated TDL library

Defining model elements in TDL to be used as test data

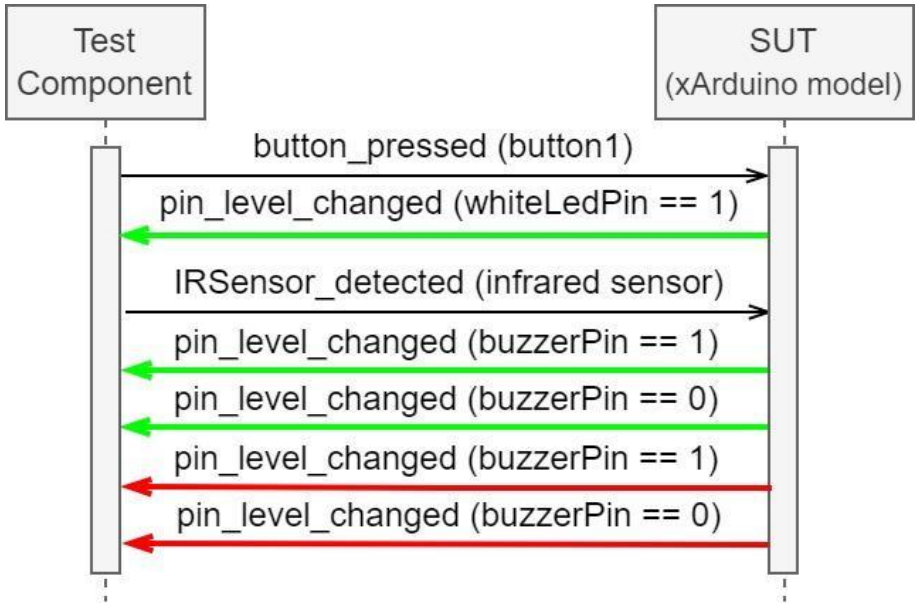


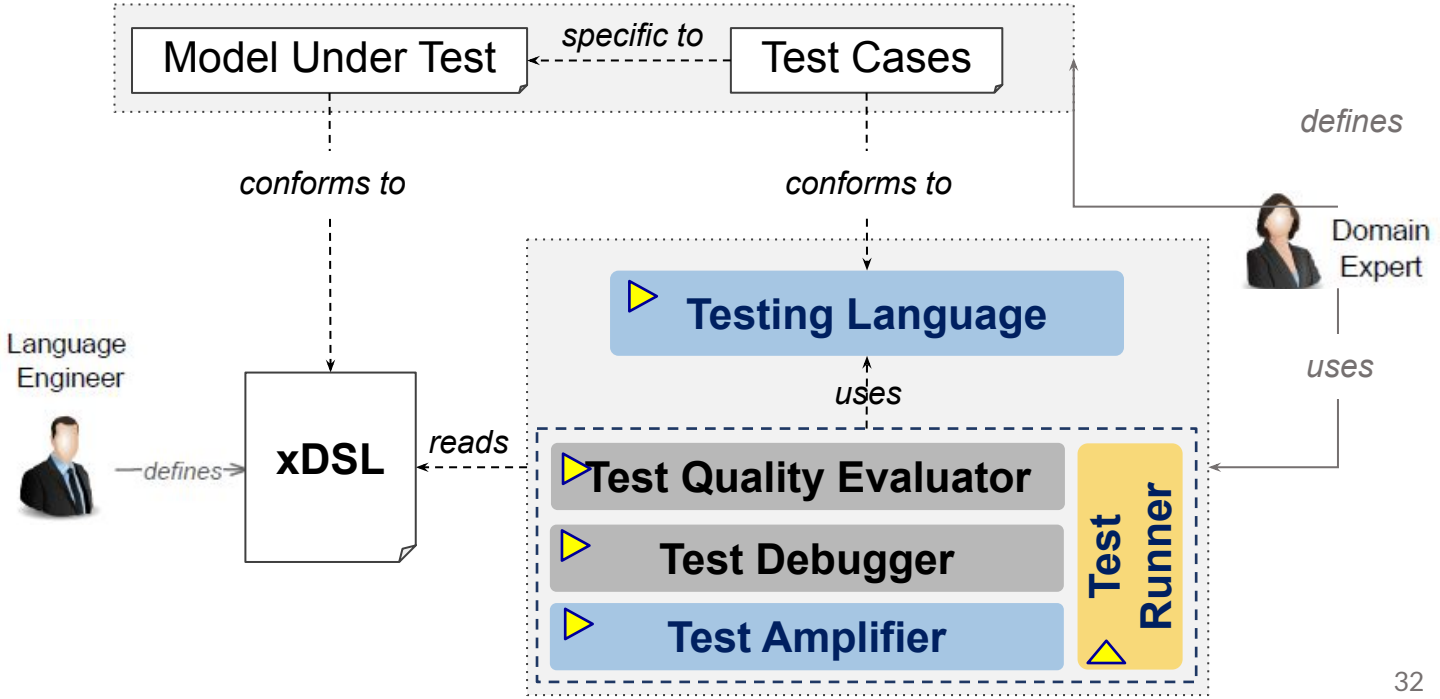

```
1  Package TestSuite4reactive {
2      Import all from common;
3      Import all from xArduinoTypes_r;
4      Import all from xArduinoEvents;
5      Import all from testConfiguration_r;
6
7      //test data
8      InfraRedSensor IRSensor(_name = "infrared sensor");
9      DigitalPin whiteLedPin (_name = "whiteLedPin", level =?);
10     PushButton button1 (_name = "button1");
11     DigitalPin buzzerPin (_name = "buzzerPin", level =?);
12
13     //test cases
14     Test Description test1 uses configuration xArduinoConfiguration_r{
15         tester.reactiveGate sends button_pressed (
16             button = button1) to arduino.reactiveGate;
17         arduino.reactiveGate sends pin_Level_Changed (
18             pin = whiteLedPin (level = '1')) to tester.reactiveGate;
19         tester.reactiveGate sends IRSensor_detected (
               sensor = IRSensor) to arduino.reactiveGate;
           arduino.reactiveGate sends pin_Level_Changed (
22             pin = buzzer_pin (level = '1')) to tester.reactiveGate;
23         arduino.reactiveGate sends pin_Level_Changed (
24             pin = buzzer_pin (level = '0')) to tester.reactiveGate;
25         arduino.reactiveGate sends pin_Level_Changed (
26             pin = buzzer_pin (level = '1')) to tester.reactiveGate;
27         arduino.reactiveGate sends pin_Level_Changed (
28             pin = buzzer_pin (level = '0')) to tester.reactiveGate;
29     }
30 }
```

Running Test Cases on Models

Proposing an operational semantics for TDL:

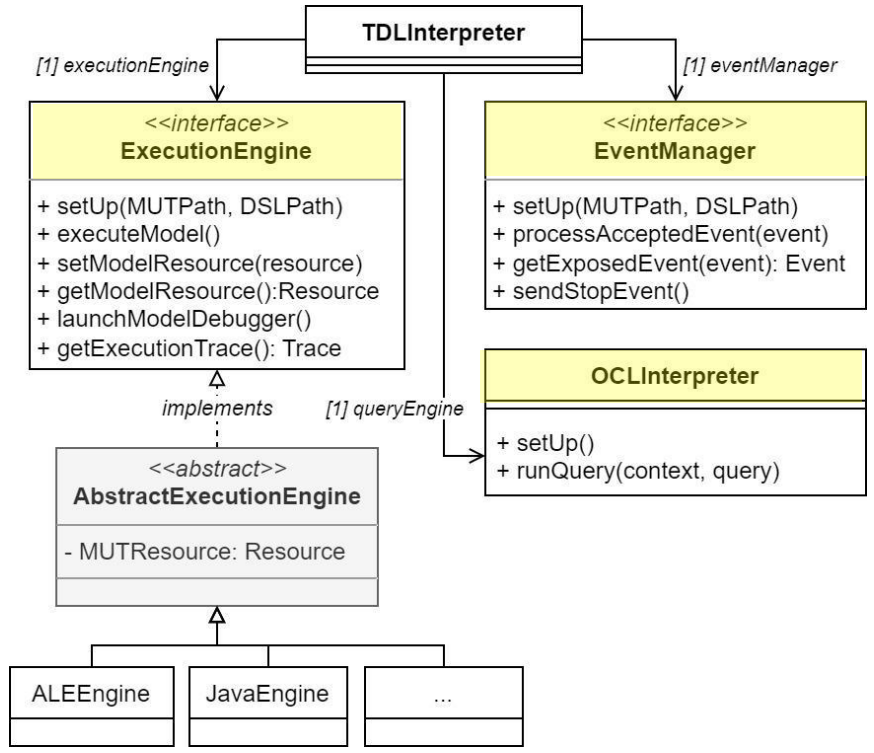
- Can run test cases on executable models
- Provides several facilities to interrogate the behavior of a model in its execution by a test case
- Produces the test execution results





TDL Interpreter

TDL Interpreter dependencies to external components



The TDL Interpreter main loop

Input:
package: the TDL package containing the TDL test cases to be executed

```
1  begin
2      foreach testcase ∈ package.testCases do
3          testcase.configuration.activate()
4          foreach behavior ∈ testcase.behaviors do
5              if behavior is Message then
6                  sourceGate ← behavior.source
7                  targetGate ← behavior.target
8                  if sourceGate.component.role is Tester then
9                      request ← behavior.argument
10                     targetGate.sendRequestToSUT(request)
11                 else if sourceGate.component.role is SUT then
12                     testOracle ← behavior.argument
13                     targetGate.assert(testOracle)
14             else if behavior is <other behavior types> then
15                 ...
```

Evaluation

- **RQ#1**: Does the approach provide testing facilities for xDSLs in which their abstract syntax is designed for ***different domains***?
- **RQ#2**: Does the approach provide testing facilities for xDSLs in which their operational semantics is implemented using ***different metaprogramming approaches***?

		xFSM	xBPMN	xMiniJava	xArduino	xPSSM
xDSL size	Abstract syntax size (# EClasses)	3	39	76	59	39
	Semantics size (LoC)	K3: 110 ALE: 90	ALE: 318	K3: 1042	K3:768	K3: 975
Tested Models	Number of tested Models	5	2	6	6	5 + 60
	Size range of tested models (# EObjects)	7-133	26-46	31-571	18-59	13-154

Evaluation Result

- **RQ#1**: Does the approach provide testing facilities for xDSLs in which their abstract syntax is designed for ***different domains***?
- **RQ#2**: Does the approach provide testing facilities for xDSLs in which their operational semantics is implemented using ***different metaprogramming approaches***?

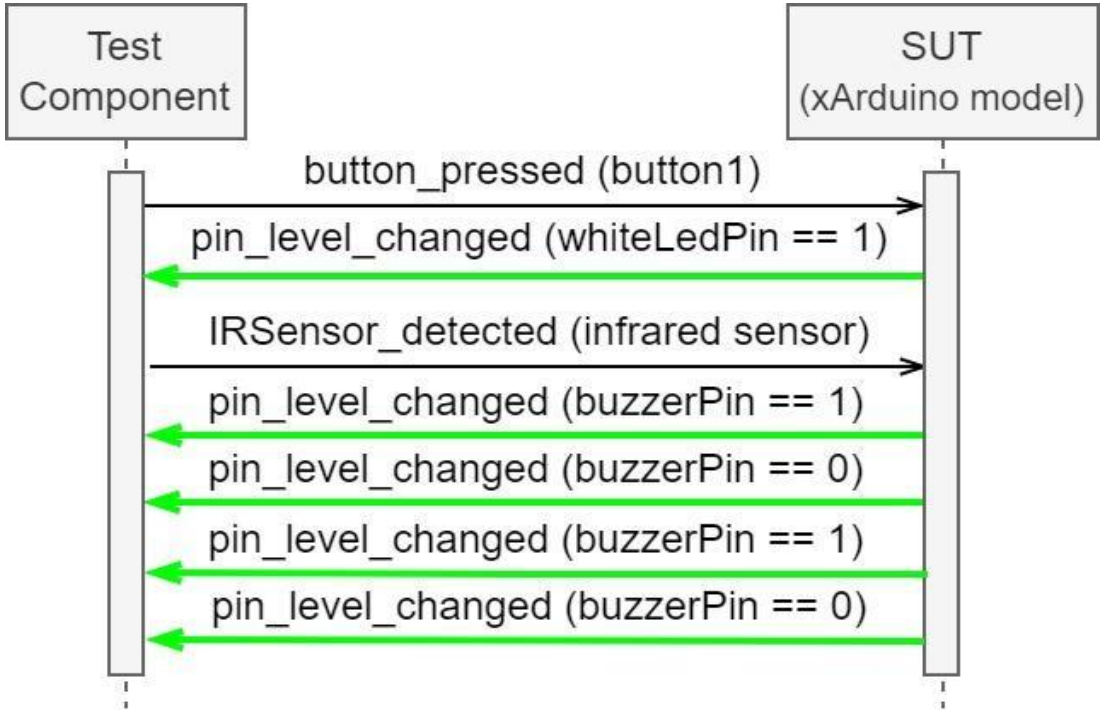
		xFSM	xBPMN	xMiniJava	xArduino	xPSSM
xDSL size	Abstract syntax size (# EClasses)	3	39	76	59	39
	Semantics size (LoC)	K3: 110 ALE: 90	ALE: 318	K3: 1042	K3:768	K3: 975
Tested Models	Number of tested Models	5	2	6	6	5 + 60
	Size range of tested models (# EObjects)	7-133	26-46	31-571	18-59	13-154
Test Artifacts	TDL Library size (LoC generated)	76	170	189	251	203
	Total n. of test cases	45	6	77	22	216
	Size range of test suites (LoC)	50-157	33-50	33-188	30-132	25-1311

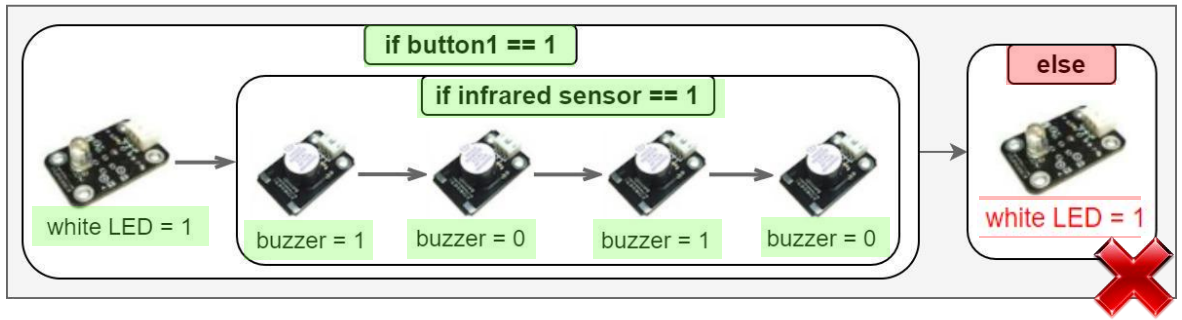
Test Improvement

Chapter 5 of the manuscript

Limits of Manually Written Test Suite for Regression Testing

- Testing a model ensure the correctness of its current version, but the model may be affected by faults in future updates





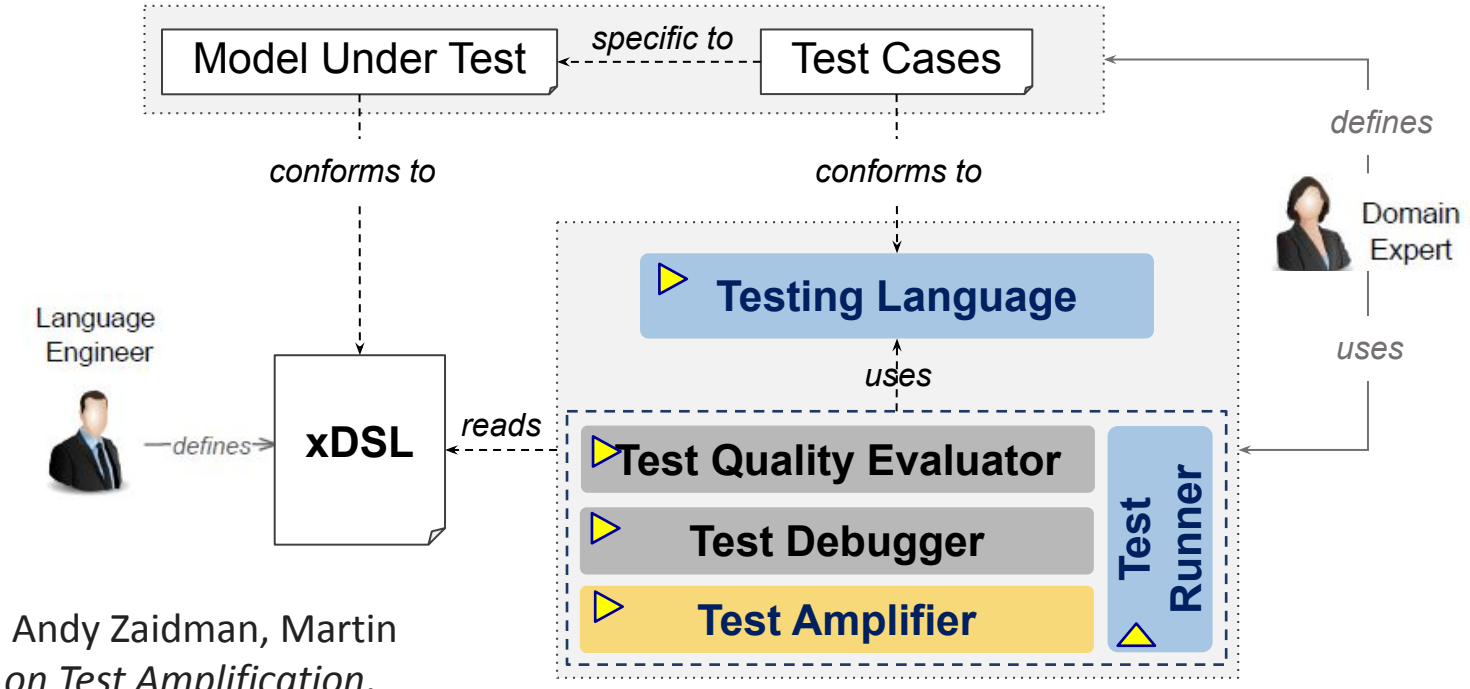
Legend: *covered* *not-covered* *Defect in the model*

Test Amplification

Leveraging the value of existing manually-written tests to achieve a specific engineering goal [2]

Amplification by Addition: adding new test cases by modifying existing test cases to improve them for regression testing

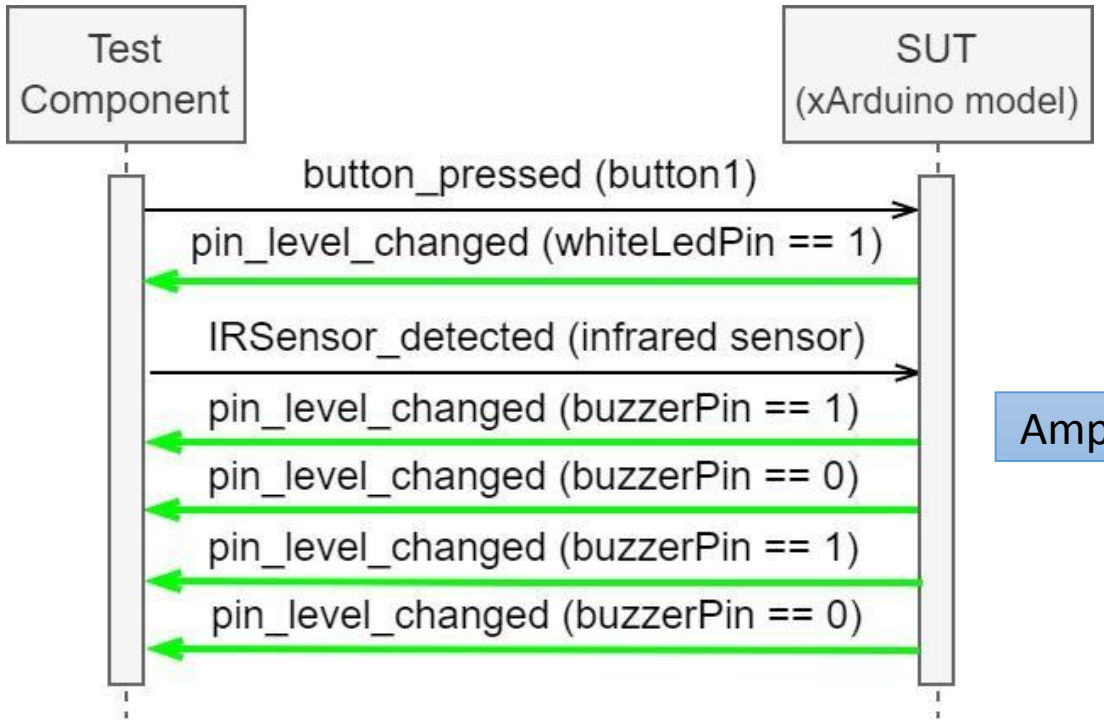
Objective:
Test amplification for xDSLs



[2] Benjamin Danglot, Oscar Luis Vera-Pérez, Zhongxing Yu, Andy Zaidman, Martin Monperrus, Benoit Baudry. *A Snowballing Literature Study on Test Amplification*. Journal of Systems and Software, Elsevier, 2019, 157, pp.1-16.

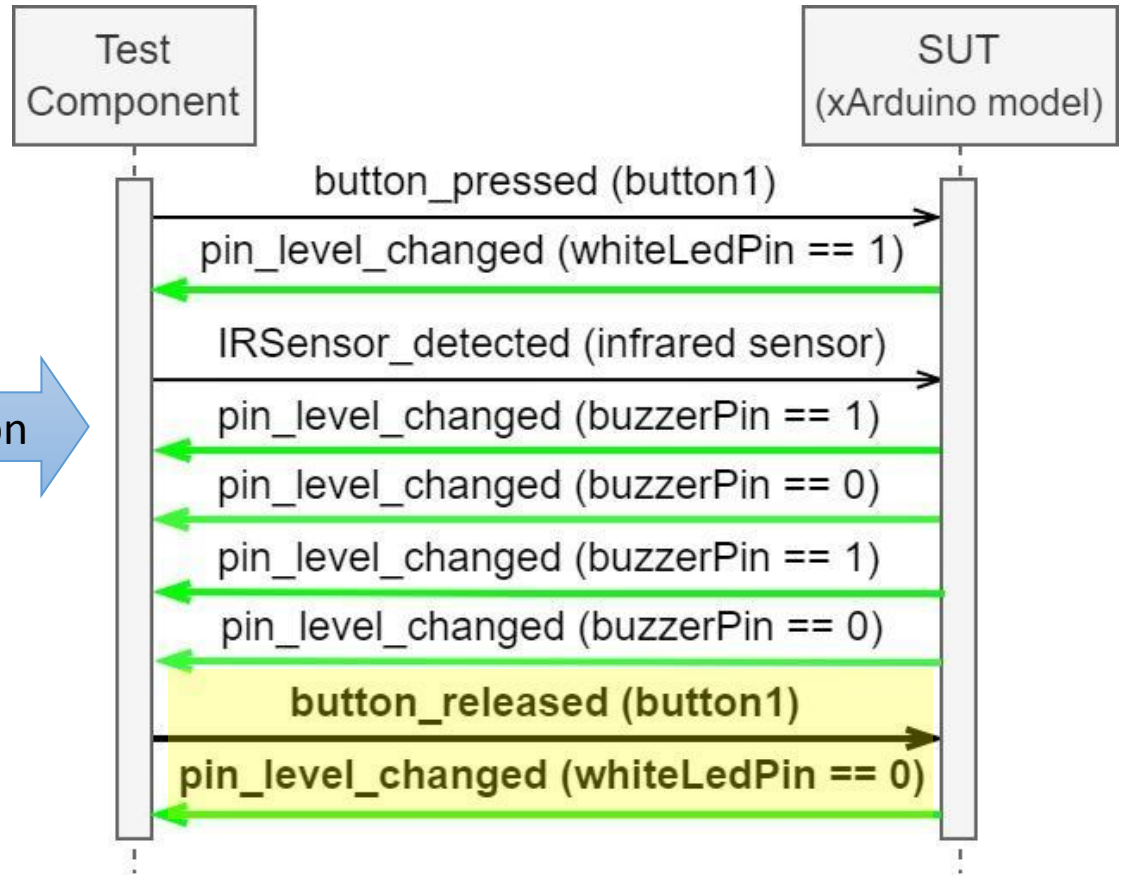
Test Amplification Example

The manually written test case (input)

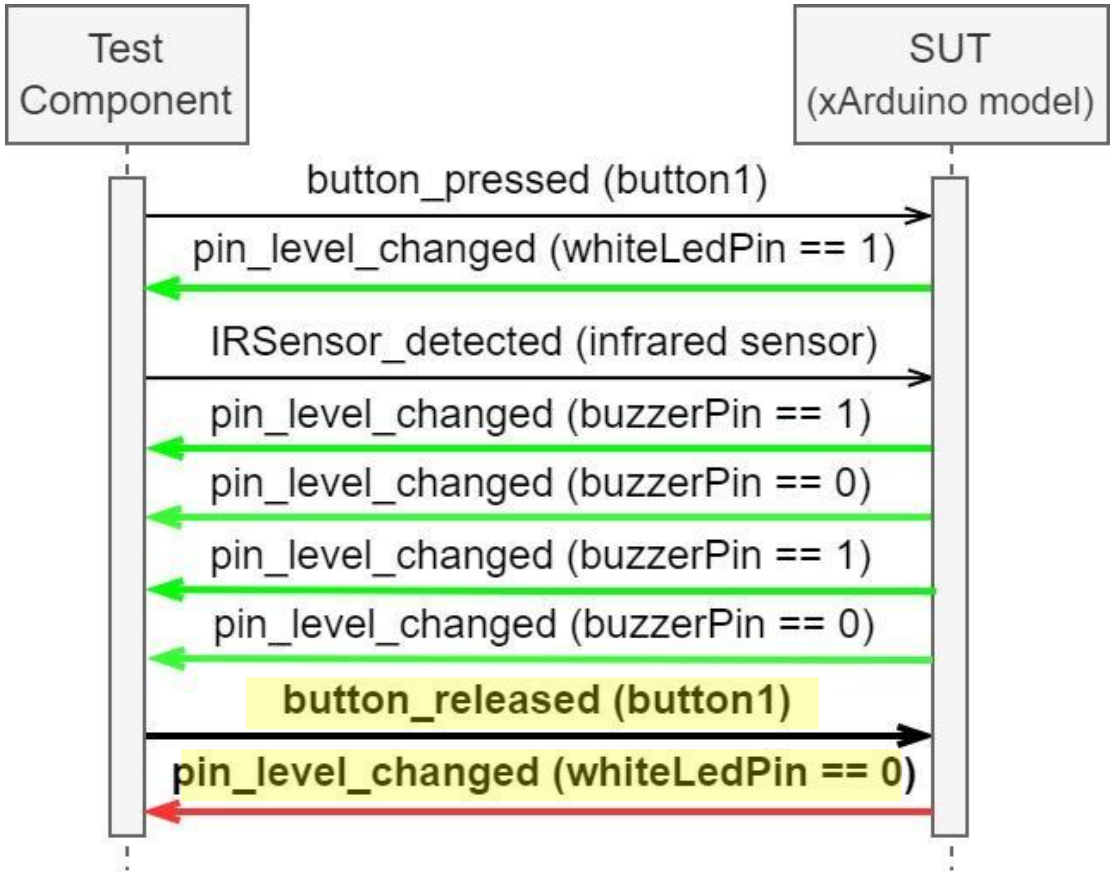


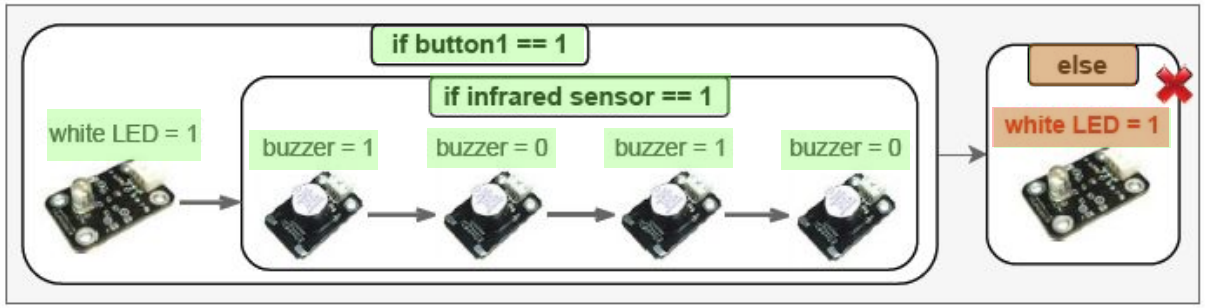
Amplification

A test case to be generated by amplification (output)



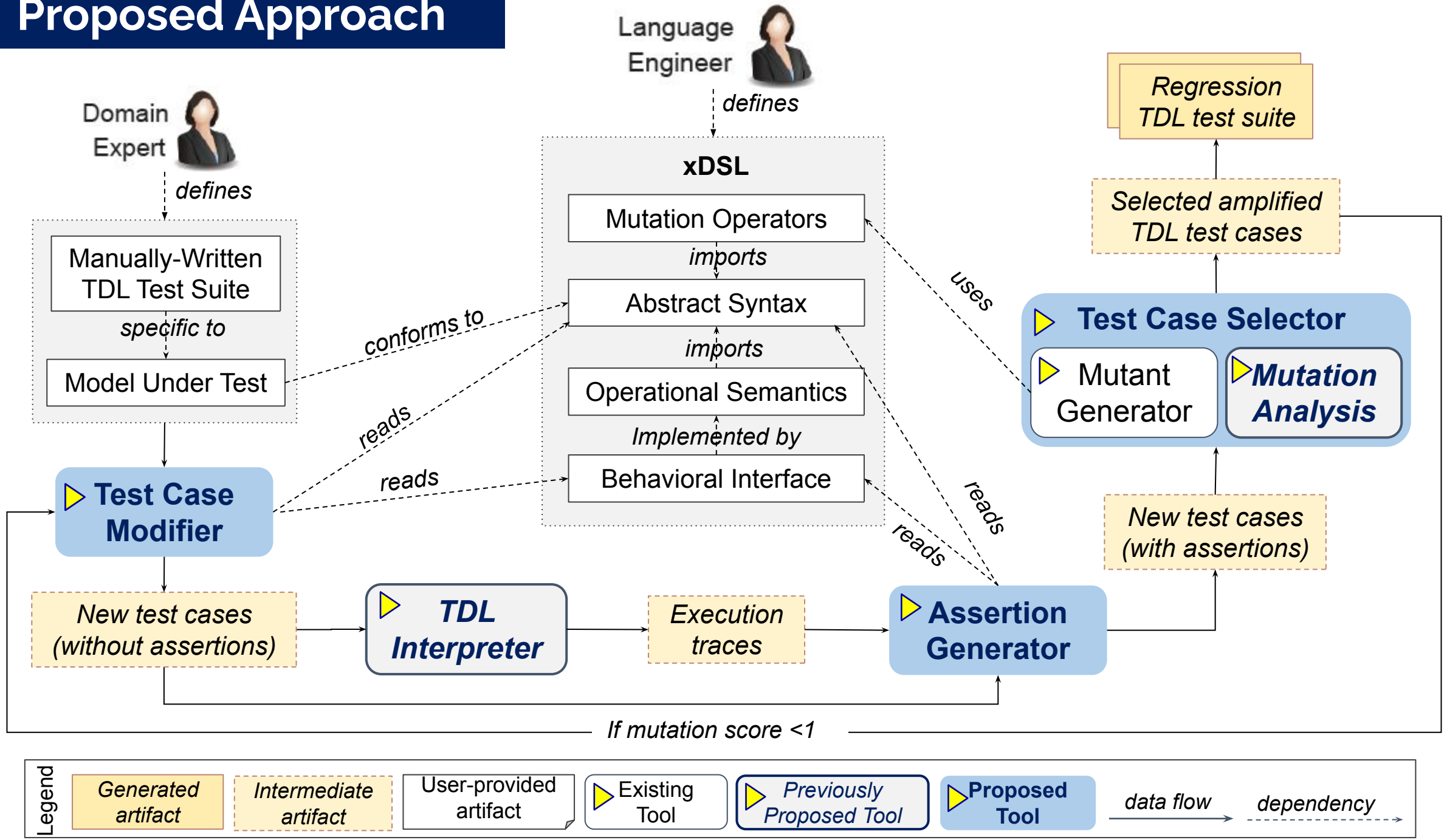
The Amplified Test Case & its trace on the faulty model





The last assertion fails, so the test case fails => detecting the regression fault

Proposed Approach



First tool: Test Case Modifier

Step 1: Removing Assertions

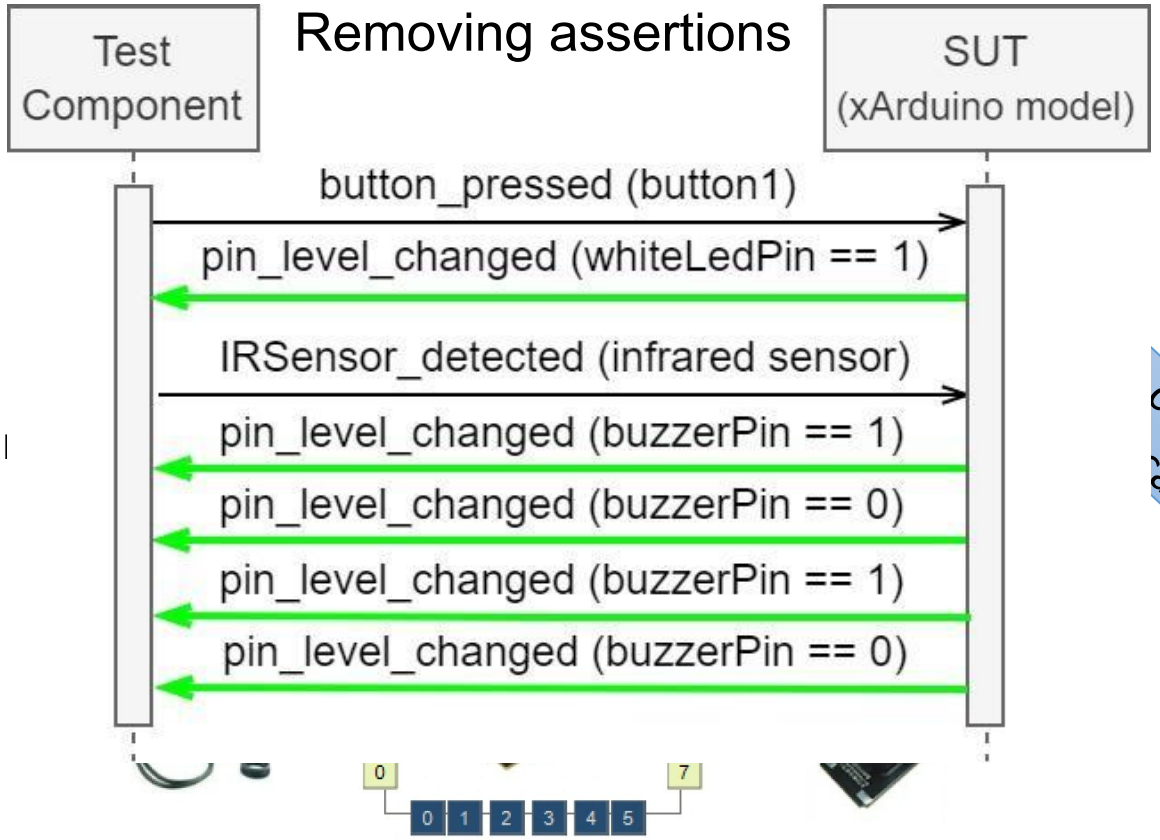
Step 2: Test Input Data Modification Operators

- **Modification of Primitive Data:**
  - A *numeric* value n is replaced.
  - A *string* value is modified.
  - A *boolean* value is negated.

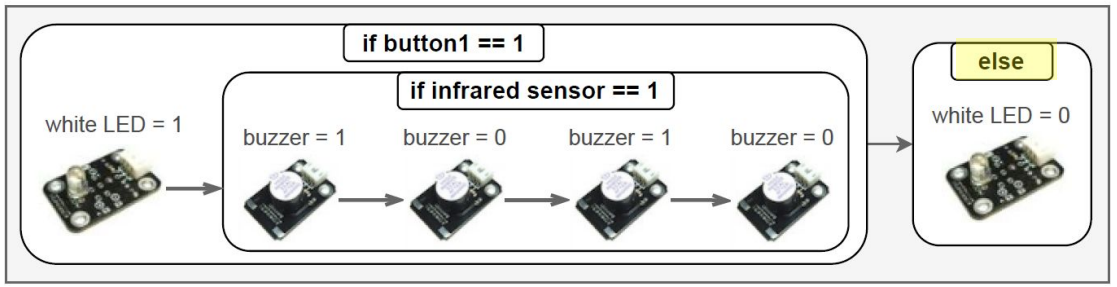
- **Modification of Event Sequences:**
  - Event duplication
  - Event deletion
  - Event permutation
  - Event creation
  - Event modification

First tool: Test Case Modifier

Removing assertions



Fi

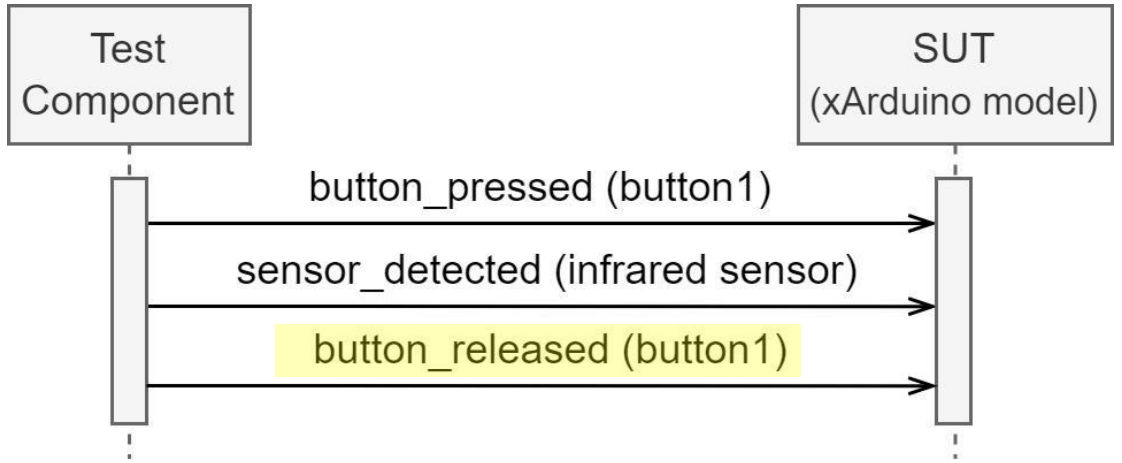


Finding not-used events of the interface:

```
BehavioralInterface ArduinoInterface
    accepted event button_pressed:
        parameters = [button: PushButton]
    accepted event button_released:
        parameters = [button: PushButton]
    accepted event IRSensor_detected:
        parameters = [sensor: InfraRedSensor]
```

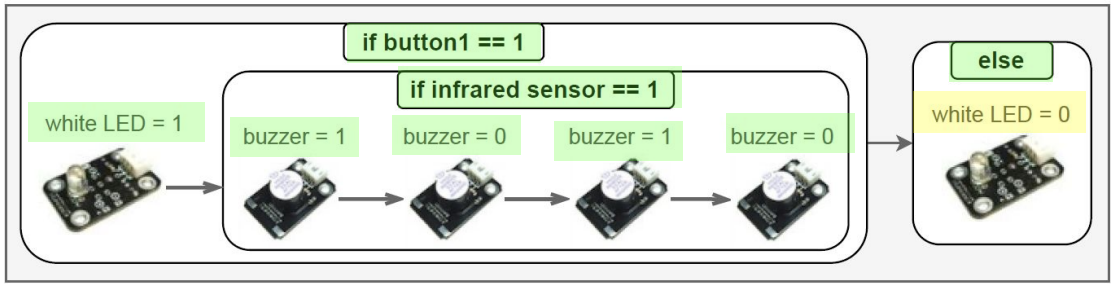
case
ation

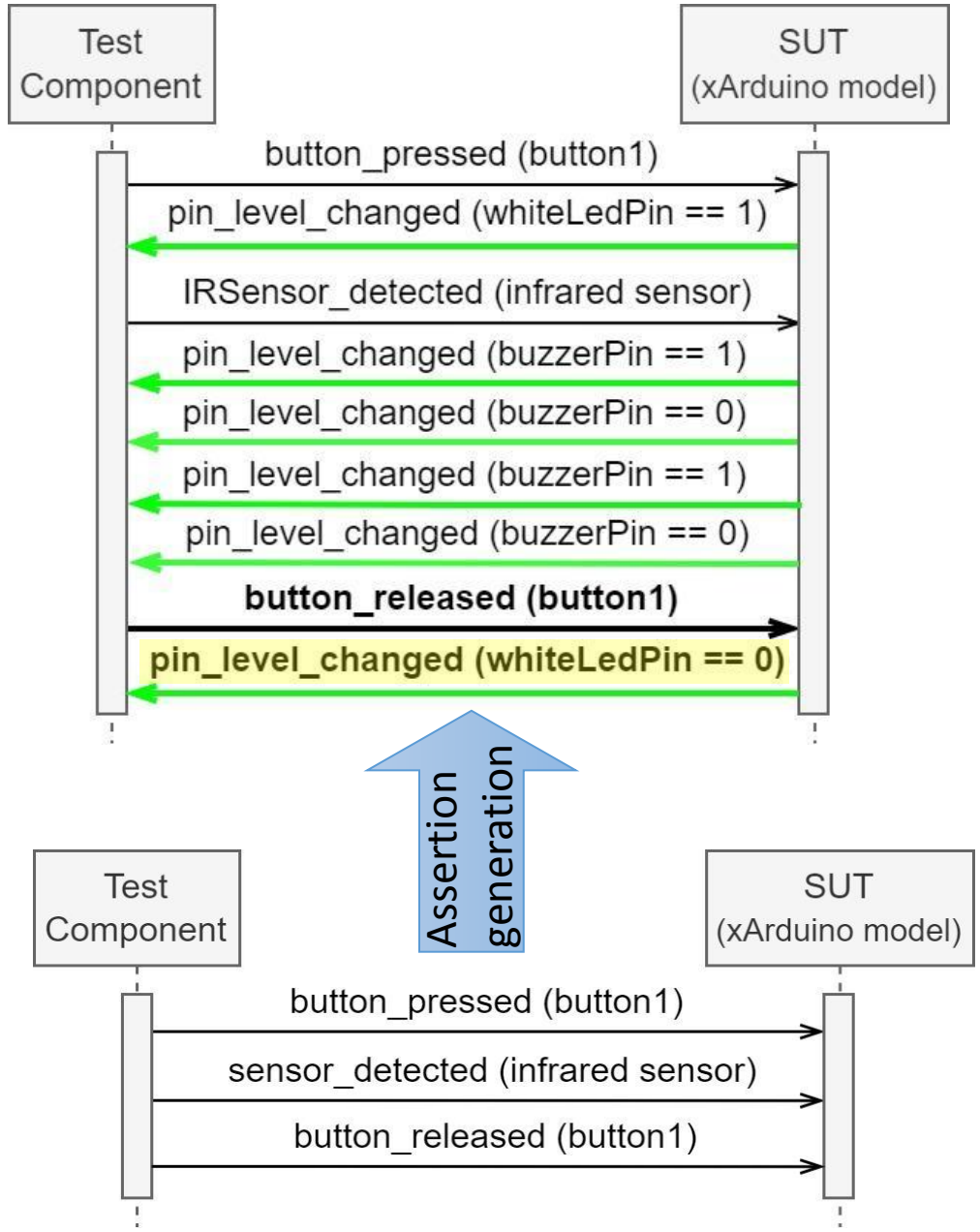
Instantiating events and adding them to the test case:



Second Tool: Assertion generator

Executing the new test case on the original model, the trace provides the *exposed events* that can be transformed into the test case assertions



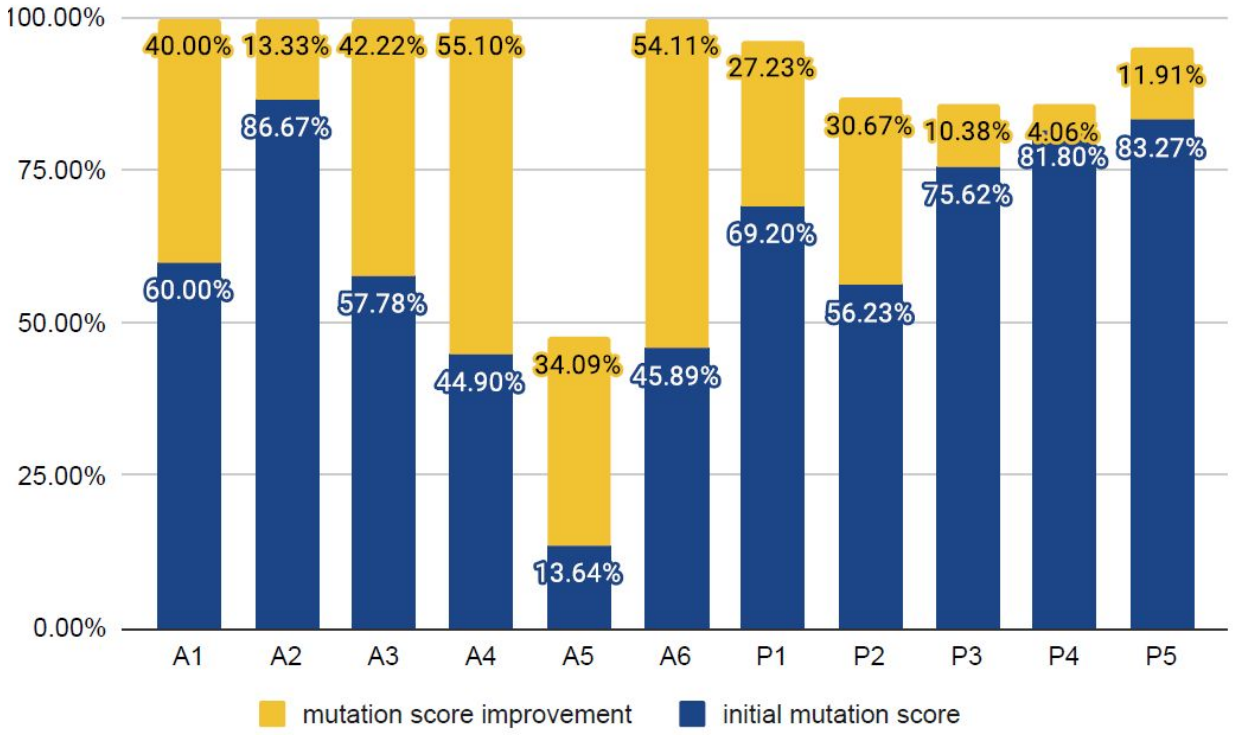


Evaluation

RQ1 How much genericity is provided by the framework in terms of the supported xDSLs?

RQ2 To what extent do the generated test cases increase the mutation score of the original, manually-written, test cases?

	xArduino	xPSSM
Number of tested models	6	5 + 60
Size range of models (#EObjects)	18-59	13-154
Initial test suite size (#test cases)	22	216
#generated mutants	394	12,087



Mutation score improvement for 11 test suites of manually defined models:
6 xArduino models (A bars)
5 xPSSM models (P bars)

Evaluation Results

RQ3: To what extent do the size and the quality of the original test suites impact the amplification result?

- Different datasets based on size and mutation score (threshold = 80%)
- Two types of comparison:
  - same size, different qualities
  - different sizes, similar qualities

⇒ by amplifying **high-quality tests** and/or **more test cases**, it is more probable to generate new effective test cases

⇒ the original test cases with **higher quality** have **more contribution** to test amplification

Conclusion & Perspectives

Chapter 6 of the manuscript

Proposal:
A generic testing framework for xDSLs

Users

- Enabling *language engineers* to provide testing support for their xDSLs
- Enabling *domain experts* to test behavioral models as early as possible

Contributions:

- Test case definition, execution, and reporting
- Test quality measurement (*in collaboration with JKU and UAM*)
- Test debugging (*in collaboration with JKU*)
- Test amplification for improving regression testing (*in collaboration with UAM*)

Publications

— International journal

- **Faezeh Khorram**, Erwan Bousse, Jean-Marie Mottu, Gerson Sunyé, “Advanced Testing and Debugging Support for Reactive Executable DSLs”, *Software and Systems Modeling* (2022).
- **Faezeh Khorram**, Erwan Bousse, Jean-Marie Mottu, Gerson Sunyé, “Adapting TDL to Provide Testing Support for Executable DSLs”, *The Journal of Object Technology*, 20(3), pp.6:1-15, 2021.

— International conferences

- **Faezeh Khorram**, Erwan Bousse, Antonio Garmendía, Jean-Marie Mottu, Gerson Sunye, Manuel Wimmer, “From Coverage Computation to Fault Localization: A Generic Framework for Domain-Specific Languages”, *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering (SLE)*, 2022.
- **Faezeh Khorram**, Erwan Bousse, Jean-Marie Mottu, Gerson Sunyé, Pablo Gómez-Abajo, Pablo C.Cañizares, Esther Guerra, Juan de Lara, “Automatic Test Amplification for Executable Models”, *Proceedings of the ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2022.

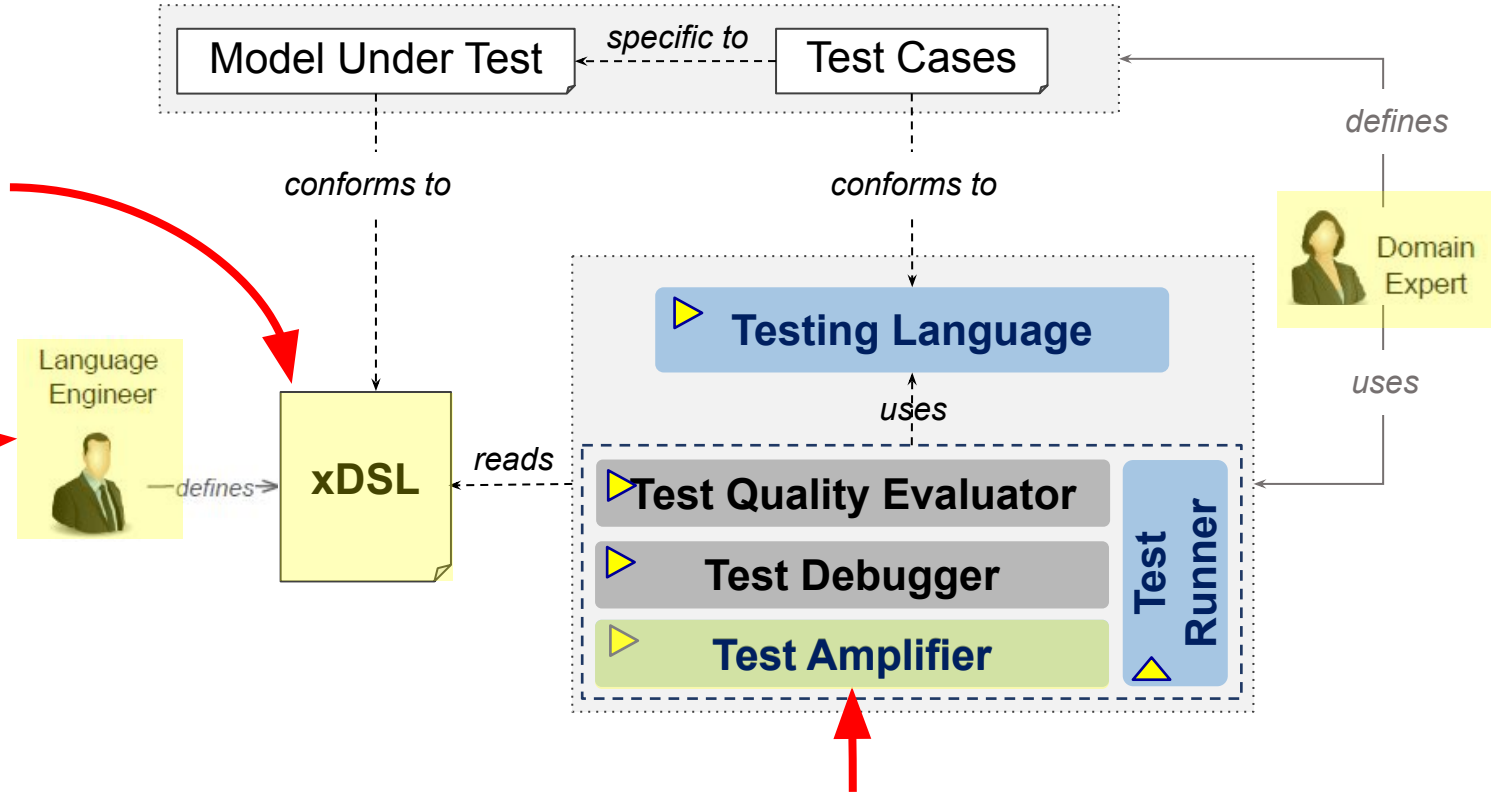
— International workshops

- **Faezeh Khorram**, Jean-Marie Mottu, Gerson Sunyé, “Challenges & Opportunities in Low-Code Testing”, *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020, Virtual.

Limitations

Genericity regarding supported xDSLs: evaluation on more xDSLs is needed

Usability for the language engineer must be assessed



Usability for the domain expert must be examined

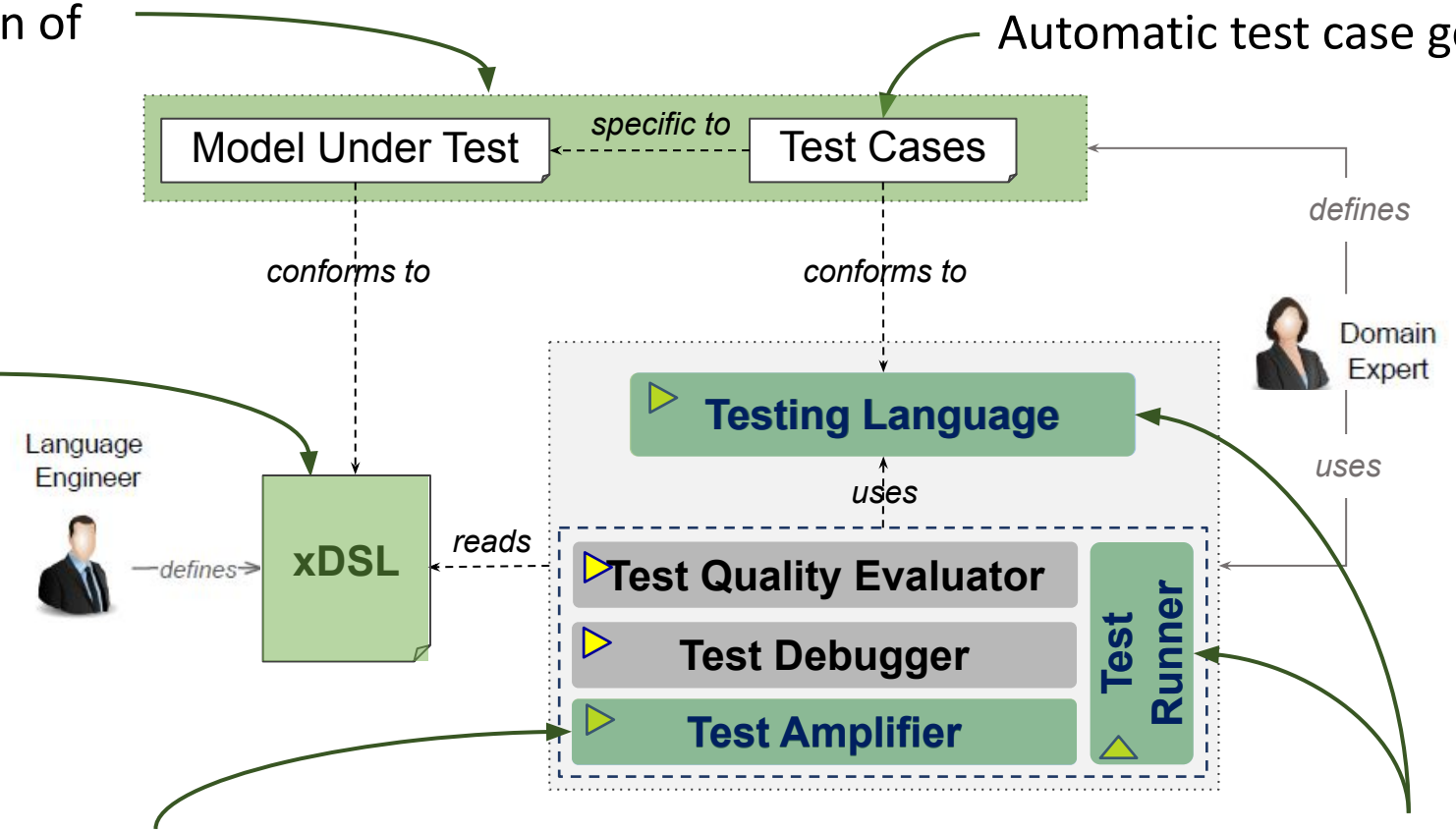
The ***impact*** of different parameters (e.g., test data modifiers, #of iterations, …) on the amplification results must be studied

Perspectives

Automatic co-evolution of models and test cases

Automatic test case generation

Testing support for compiled executable DSLs



Test amplification for other objectives (e.g., improving coverage, reproducing crashes, detecting new faults,…)

Broadening *test oracle definition* approaches (e.g., using temporal properties to define oracles)

A Testing Framework for Executable Domain-Specific Languages

Faezeh Khorram
IMT Atlantique

Reporters *:*

- Prof. Anne ETIEN, Université de Lille, France
- Prof. Juergen DINGEL, Queen's University, Canada

Examiners:

- Prof. Benoît BAUDRY, KTH Royal Institute of Technology, Sweden
- Dr. Javier TROYA, Universidad de Málaga, Spain
- Prof. Antoine BEUGNARD, IMT Atlantique, France

Thesis director*:*

- Prof. Gerson SUNYE, Nantes Université, France

Thesis supervisors:

- Dr. Jean-Marie MOTTU, Nantes Université, France
- Dr. Erwan BOUSSE, Nantes Université, France