

Un Environnement de Test pour les Langages Dédiés Exécutables

Faezeh Khorram

GDR-GPL 2022 Prix de thèse, l'accessit

NaoMod, LS2N, IMT Atlantique

Directeur de thèse

- Prof. Gerson SUNYE, Nantes Université, France

Encadrants

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

Soutenu le 12/12/2022

Qui suis-je?

Avant:

- Master en génie logiciel (2016-2019)
 - Sharif University of Technology (Iran)
 - Méthodologie dirigée par les modèles pour le développement de «Serious games»
- Doctorat en génie logiciel (2019-2022)
 - IMT Atlantique (Nantes, France)
 - Projet européen Lowcomote
 - Détachement à l'Université JKU de Linz (Autriche)
 - Détachement à l'université UAM de Madrid (Espagne)



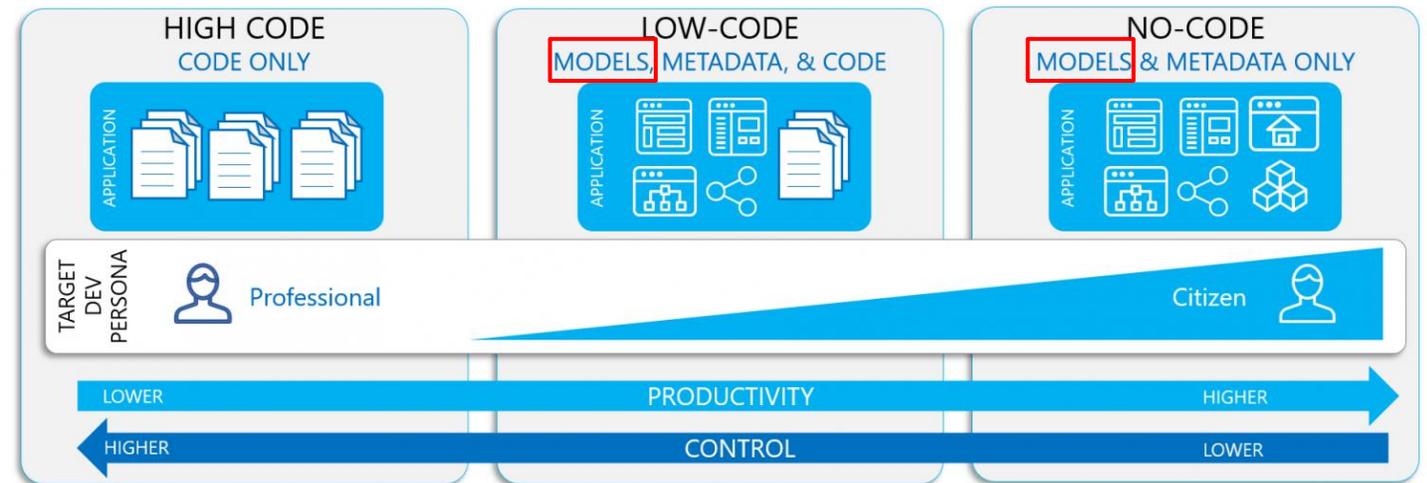
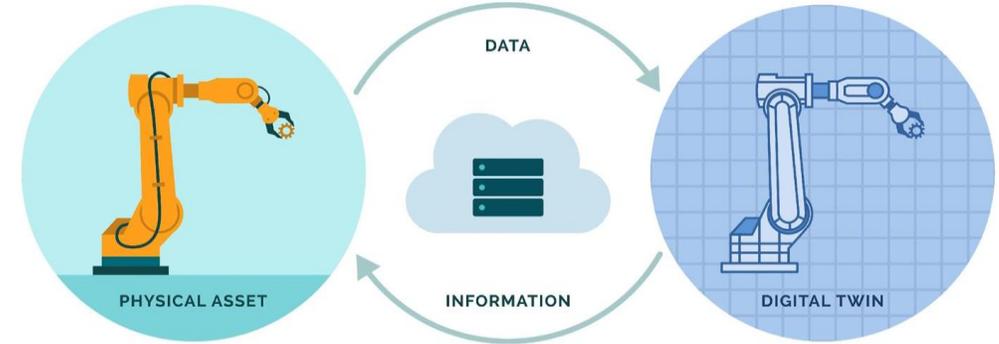
Aujourd'hui:

- Ingénieur de recherche senior (2023-actuel)
 - Département R&D de HUAWEI Technologies (Grenoble, France)
 - Vérification formelle basée sur les modèles



Ingénierie Dirigée par les Modèles (IDM)

- Un paradigme de développement logiciel
- Utilise des **modèles** comme artefact central du développement logiciel
- Traite de la **complexité** du domaine d'application
- Permet aux **non-programmeurs** de développer des logiciels



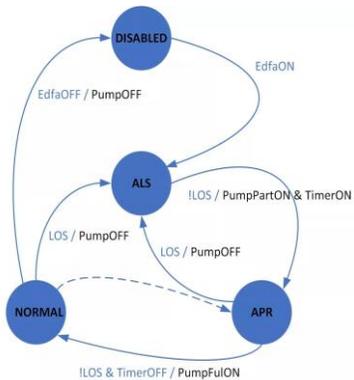
Langage de Modélisation Dédié (LMD)

Les langages pour la définition de modèles conçus pour des domaines techniques ou applicatifs spécifiques

FSM LMD

conforme à

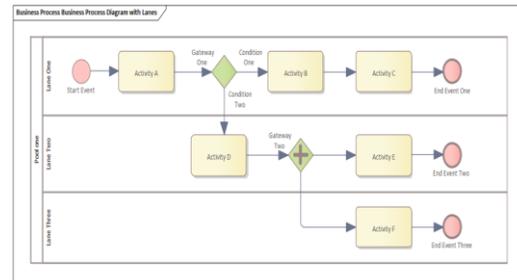
State machine



BPMN LMD

conforme à

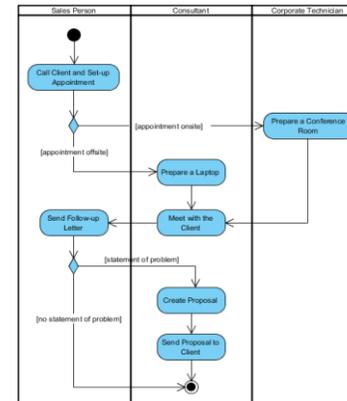
Business process



Activity LMD

conforme à

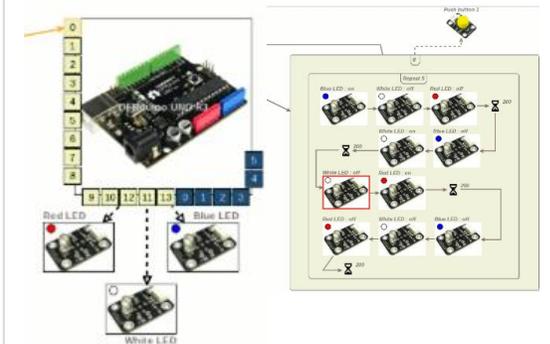
Activity diagram



Arduino LMD

conforme à

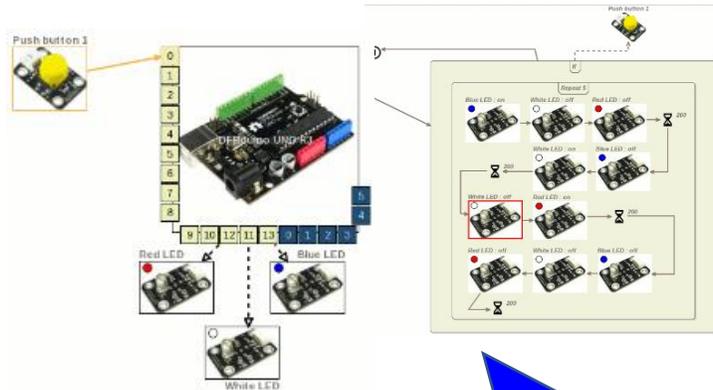
Arduino model



Modèles comportementaux: décrivant les aspects dynamiques des systèmes, doivent être exécutés pour garantir leur exactitude

Modèles exécutables

Les modèles peuvent être exécutés si le LMD fournit une sémantique d'exécution



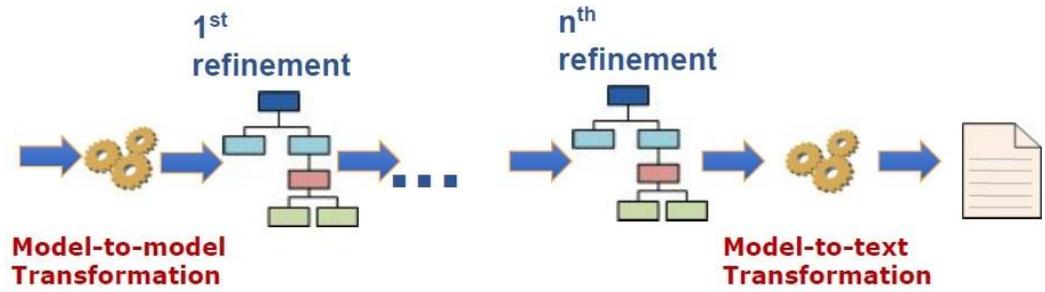
Comment l'exécuter ?

sémantique translationnelle

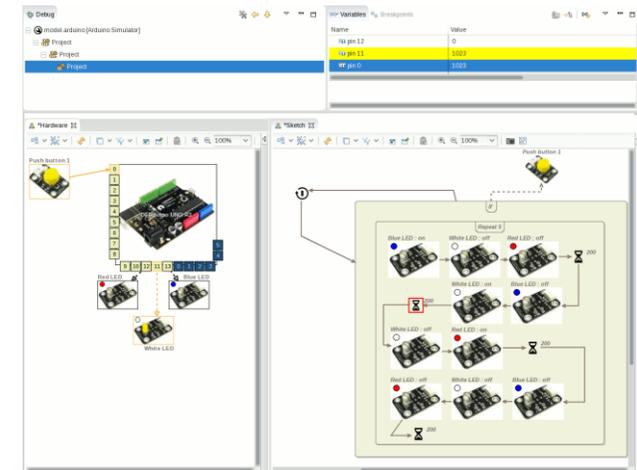
sémantique opérationnelle

Vérification et Validation (V&V) dynamiques précoces des modèles

Exécuter le code généré par un **compilateur**



Exécution du modèle lui-même à l'aide d'un **interpréteur**



Tester des modèles exécutables

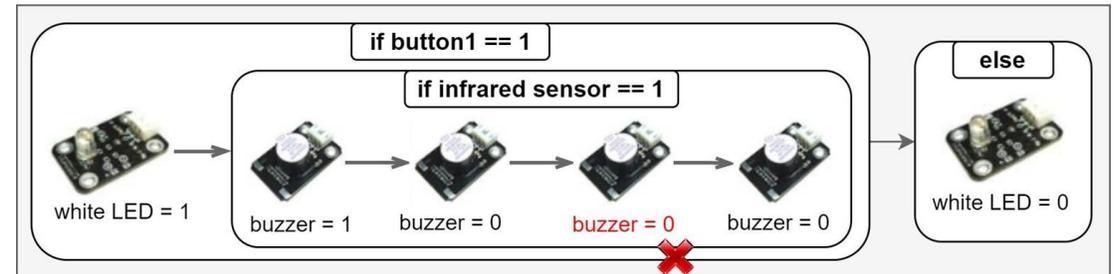
- Le test est la principale méthode utilisée pour évaluer les systèmes logiciels

Les tests impliquent :

- exécuter des systèmes dans des scénarios intéressants,
- observer s'ils agissent comme prévu.

Comment définir les données d'entrée et de sortie?

entrée: bouton enfoncé, capteur détecté



résultats attendus: LED allumée, buzzer allumé et éteint 2 fois

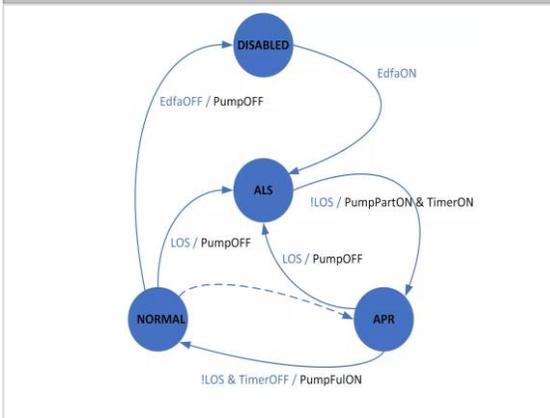
Sortie d'exécution: LED allumée, buzzer allumé et éteint une fois

(1) Un langage de test par LMD

FSM LMD

conforme à

State machine



spécifique à

cas de test

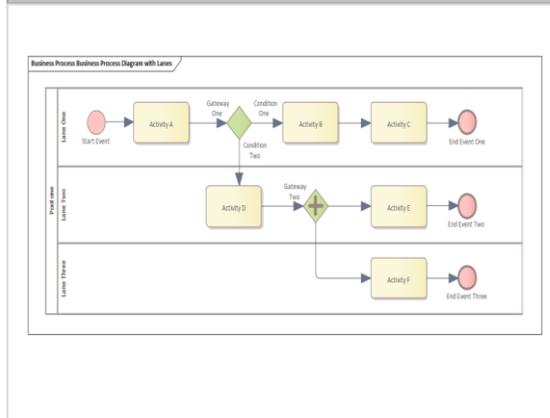
conforme à

langage de test FSM

BPMN LMD

conforme à

Business process



spécifique à

cas de test

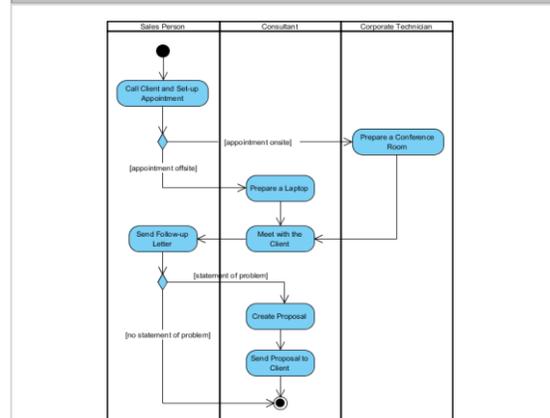
conforme à

langage de test BPMN

Activity LMD

conforme à

Activity diagram



spécifique à

cas de test

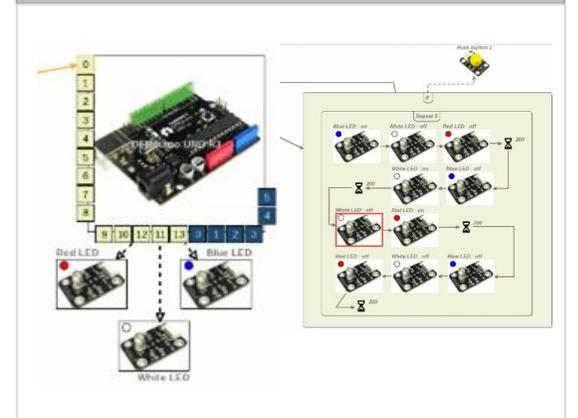
conforme à

langage de test Activity

Arduino LMD

conforme à

Arduino model



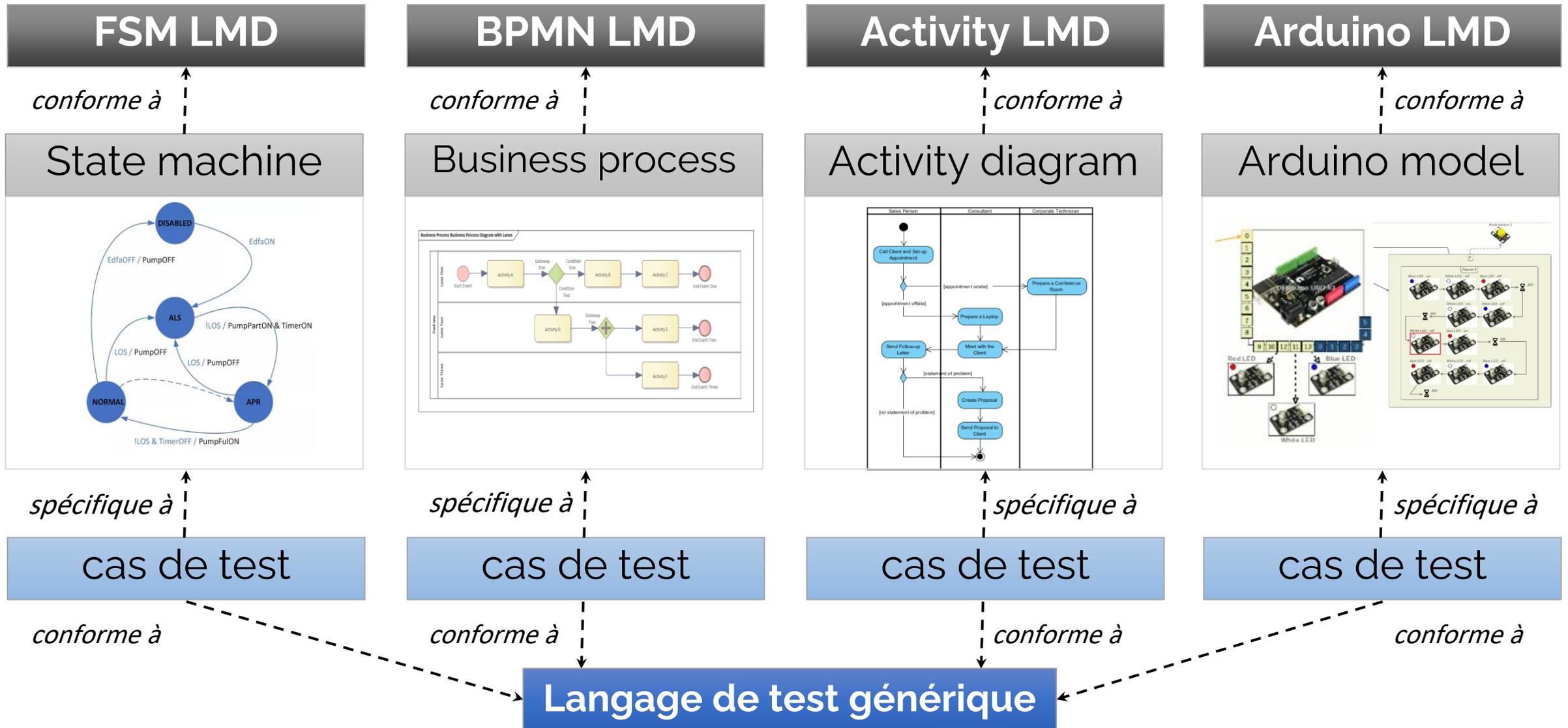
spécifique à

cas de test

conforme à

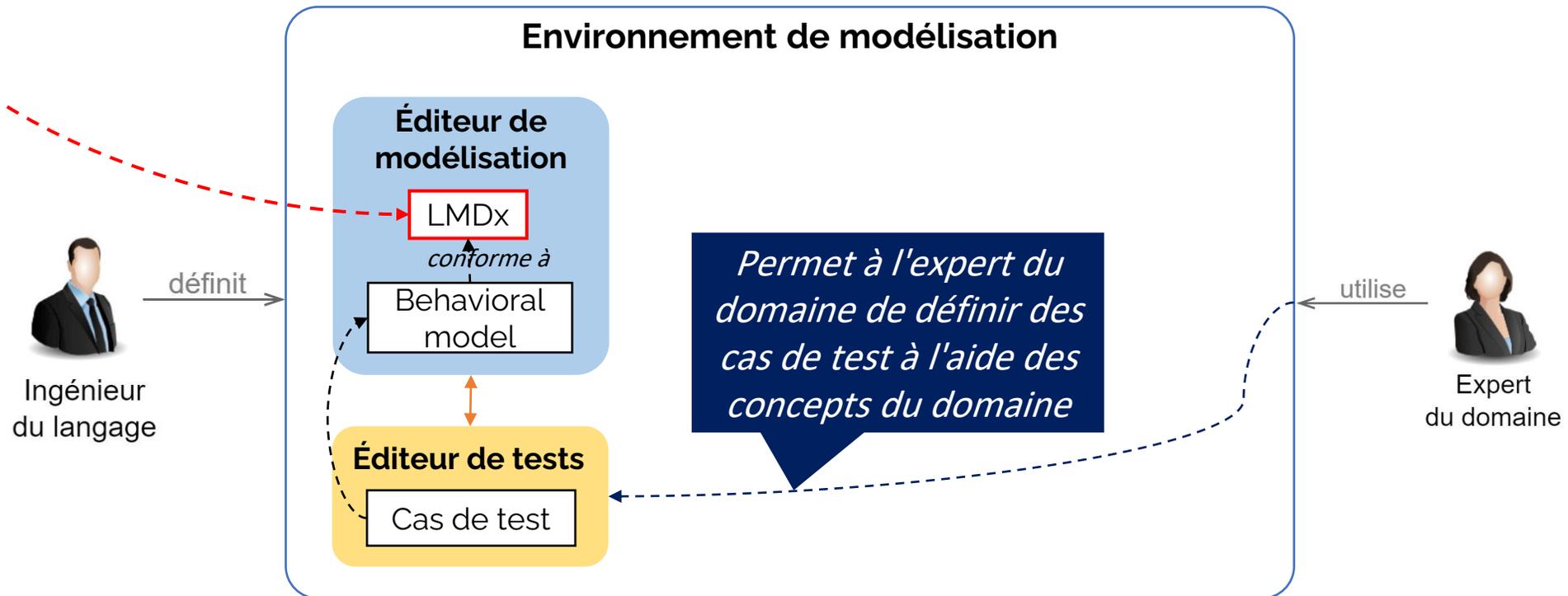
langage de test Arduino

(2) Un langage de test générique pour tous les LMD



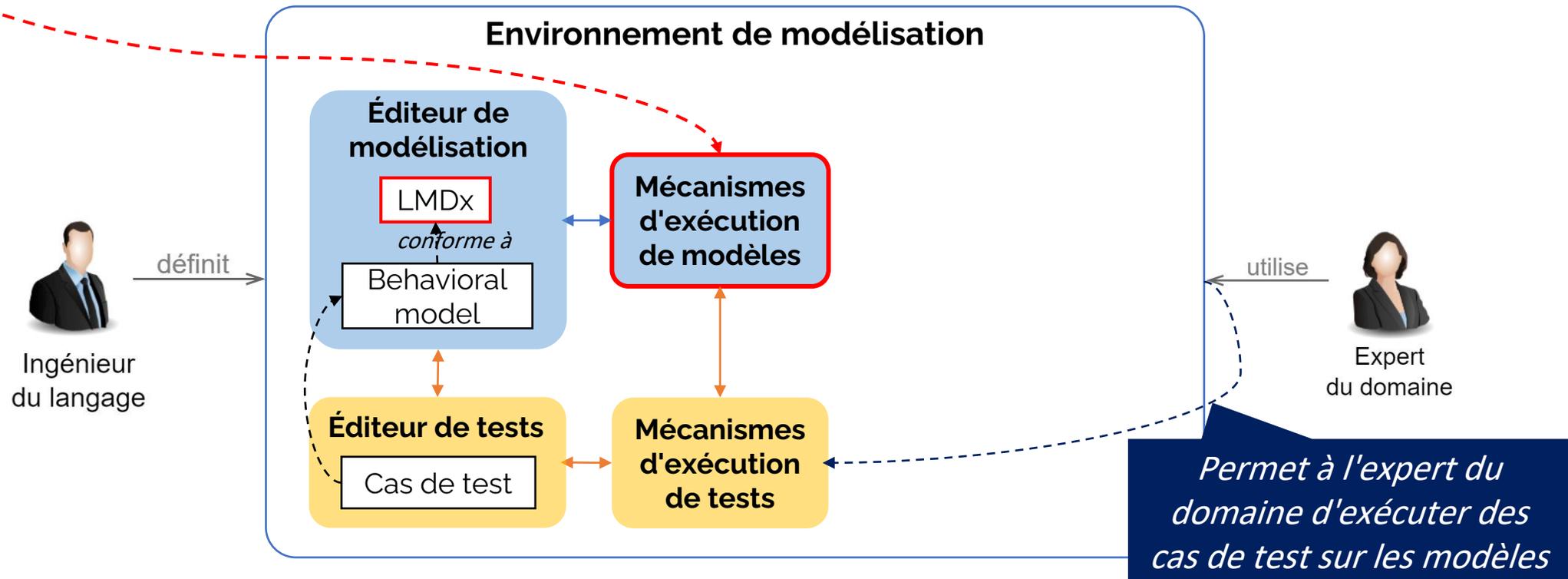
Énoncé des problèmes

- Étant donné un LMDx, les experts du domaine peuvent tester les modèles conformes seulement si les concepts du domaine peuvent être utilisés dans la spécification des cas de test
- Défi#1:** Les concepts de domaine diffèrent d'un LMDx à l'autre



Énoncé des problèmes

- Les cas de test écrits doivent être exécutés à l'unisson avec les modèles testés
- L'exécution du test doit être en quelque sorte liée à l'exécution du modèle
- **Défi#2:** Les mécanismes d'exécution des modèles diffèrent d'un LMDx à l'autre

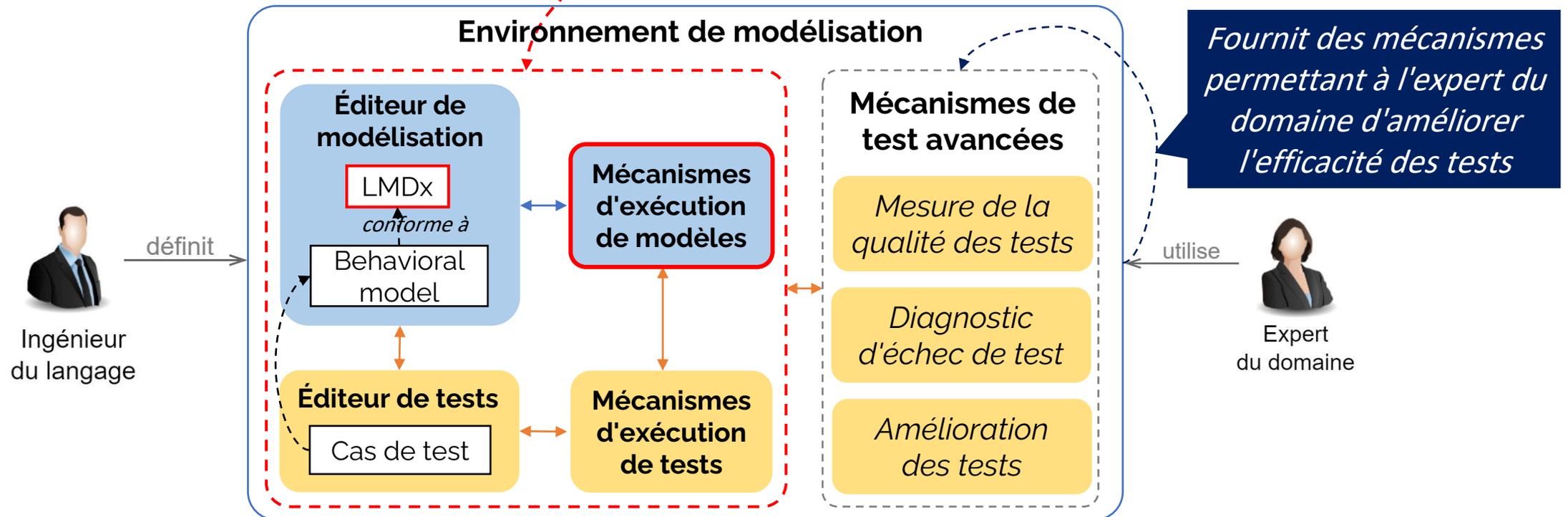


Énoncé des problèmes

Fournir des mécanismes pour améliorer l'efficacité des tests :

- Évaluer si les cas de test écrits sont assez bons
- Diagnostiquer les défauts lorsque les cas de test échouent sur un modèle
- Améliorer la qualité des cas de test écrits

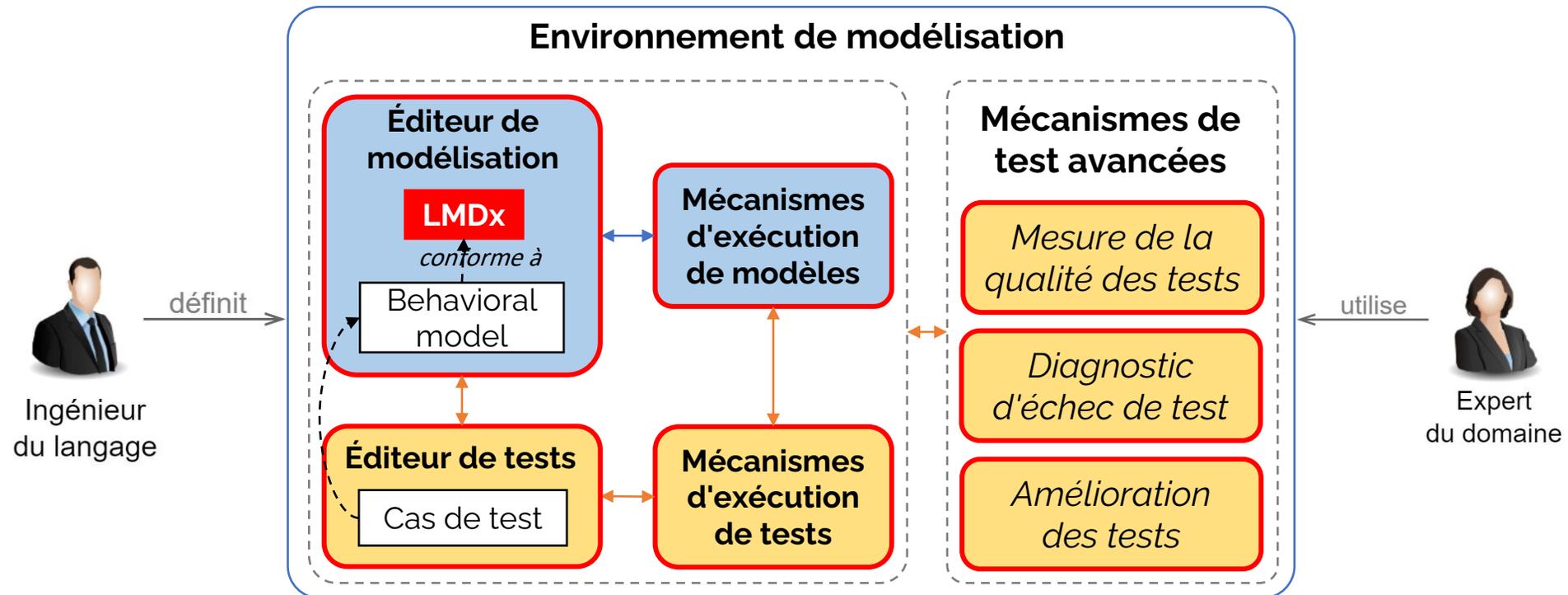
Défi #3 : dépendance aux environnements de test car ils doivent manipuler directement les cas de test et leur système sous test



Énoncé des problèmes:

Diversité et hétérogénéité des LMDx

- Nouveau LMDx \Rightarrow nouveaux concepts de domaine, nouveaux mécanismes d'exécution
- Chaque fois qu'un nouveau LMDx est conçu, un nouveau cadre de test doit être créé à partir de zéro

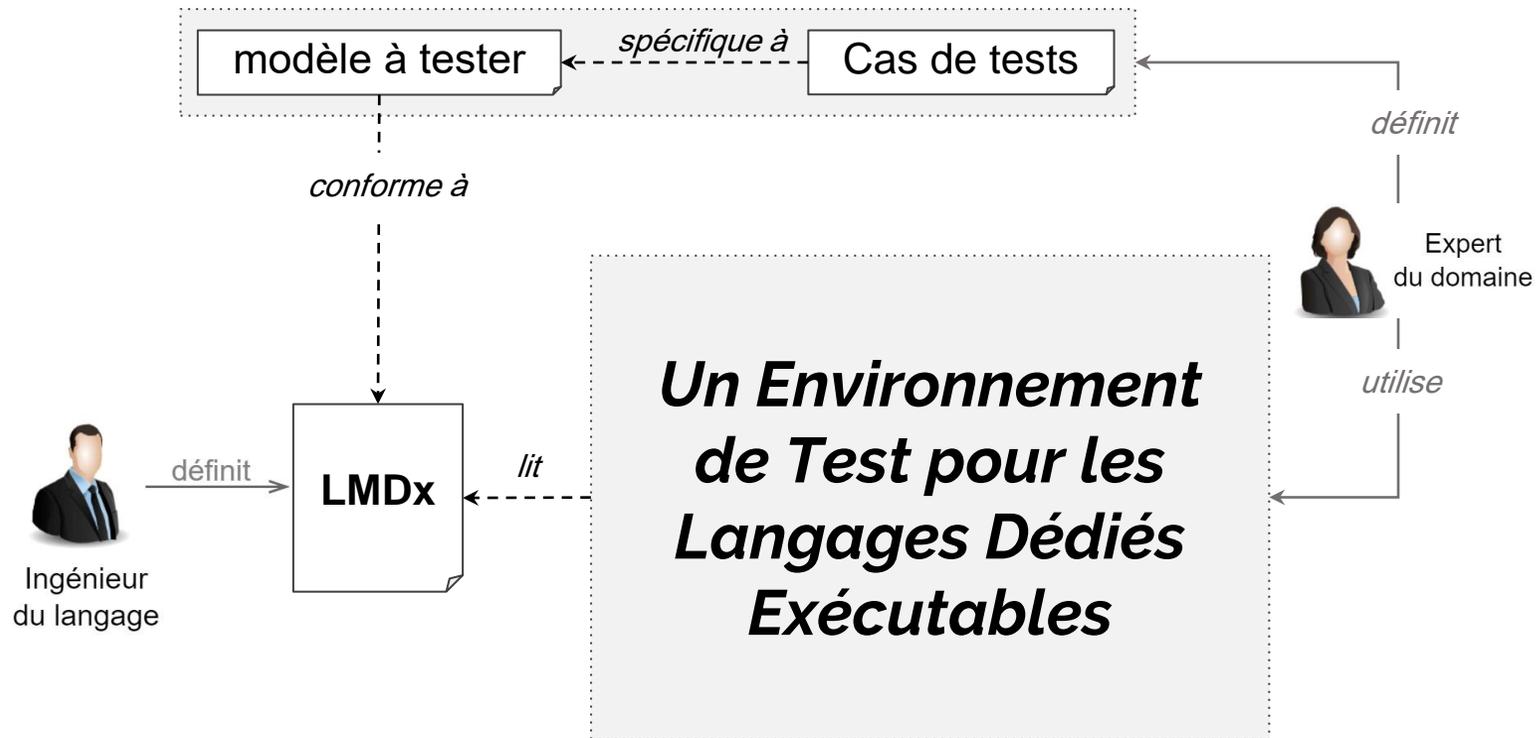


Solution: une approche *systematique* pour fournir un support de test pour chaque LMDx donné

La proposition

Utilisateurs

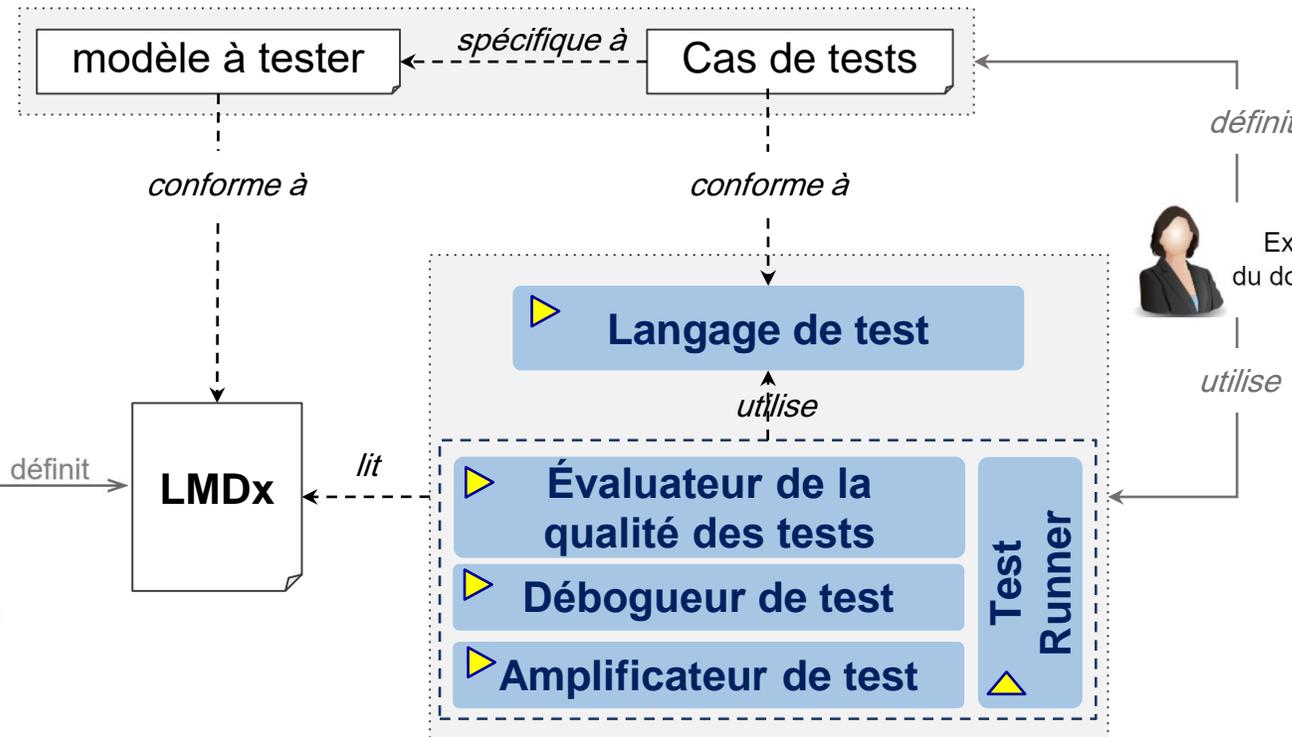
- Permettre aux ingénieurs du langage de fournir un support de test pour leurs xDSLs
- Permettre aux experts du domaine de tester efficacement les modèles comportementaux le plus tôt possible



Un Environnement de Test pour les Langages Dédiés Exécutables

Calcul de la **couverture du modèle** et analyse de la **mutation du modèle** de manière générique pour les LMDs

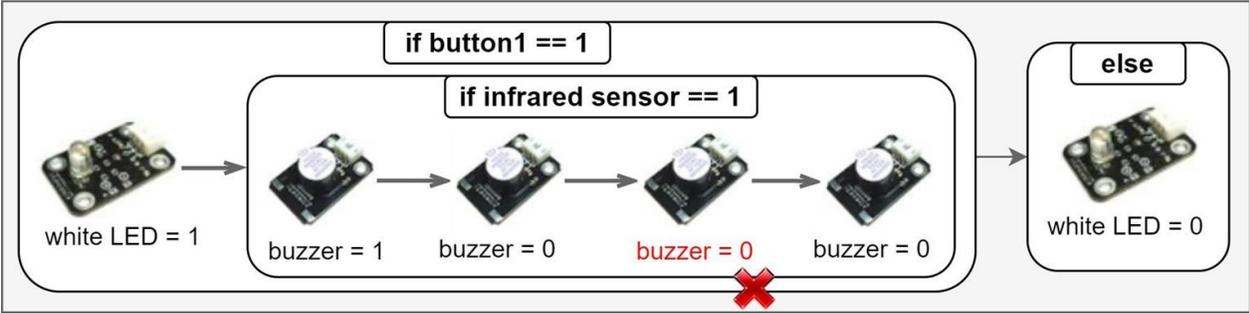
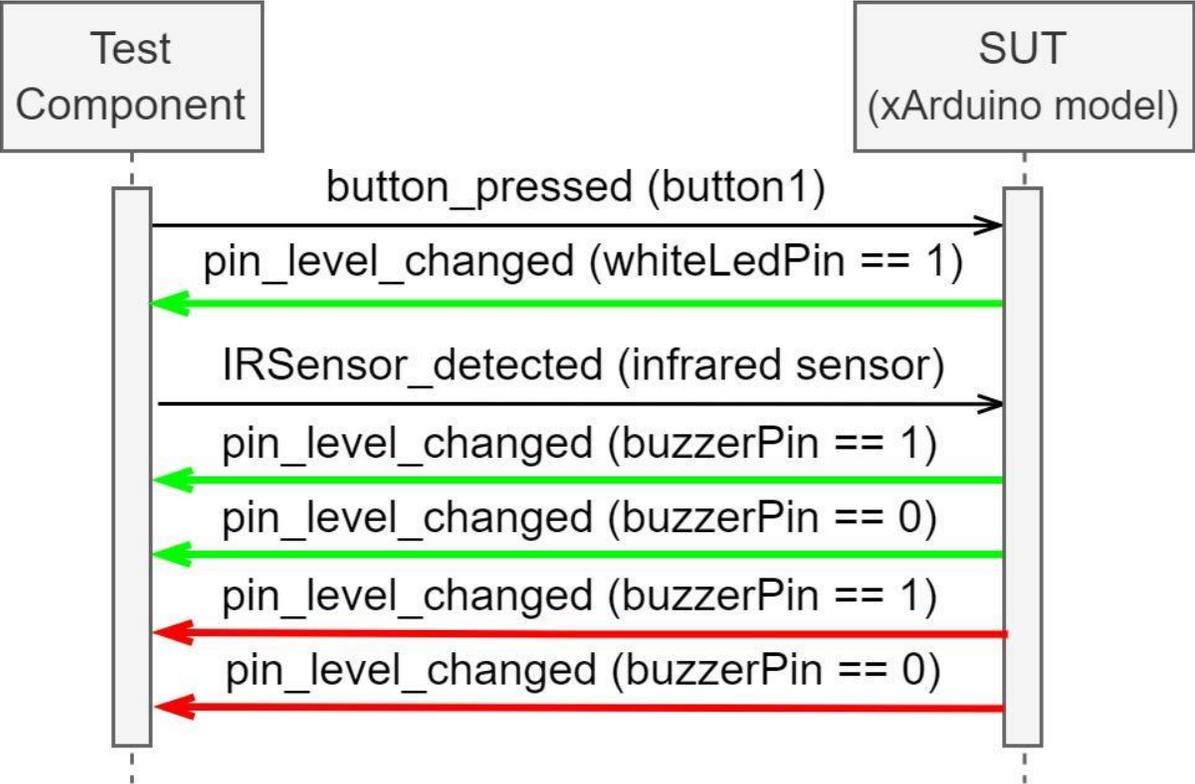
Amplification
automatique des tests pour améliorer les tests de **régression** des modèles



Adaptation du langage standard de description des tests (**TDL**) [1] au test de modèles exécutables

Mécanismes de débogage pour les tests qui **échouent** : débogage conjoint manuel modèle-test, localisation automatique des erreurs de modèle.

Exemple de scénario de test pour le modèle Arduino



Outillage au sein du GEMOC Studio



Outil de définition et d'exécution des cas de test

The screenshot displays the GEMOC Studio interface. On the left, a project tree shows the structure of the test suite. The main editor shows the TDL code for `TestSuite4reactive.tdlan2`. A blue callout bubble points to the import statements, stating: *une bibliothèque TDL générée pour l'Arduino LMD*. Another blue callout bubble points to the test case definition, stating: *définir les données de test en utilisant les concepts Arduino fournis par la bibliothèque*. On the right, the 'Test Results' window shows a table of test outcomes.

```
Package TestSuite4reactive {
  Import all from common ;
  Import all from xArduinoTypes ;
  Import all from xArduinoEvents ;
  Import all from testConfiguration ;

  //test data
  InfraRedSensor IRSensor(_name = "infraredSensor");
  DigitalPin whiteLedPin (_name = "whiteLedPin", level =?);
  PushButton button1 (_name = "button1");
  DigitalPin buzzerPin (_name = "buzzerPin", level =?);

  //test cases
  Test Description test1 uses configuration reactiveArduinoConfiguration{
    tester.reactiveGate sends button_pressed (button = button1)
      to arduino.reactiveGate;
    arduino.reactiveGate sends pin_level_changed
      (pin = whiteLedPin (level = '1')) to tester.reactiveGate;
    tester.reactiveGate sends IRSensor_detected (sensor = IRSensor)
      to arduino.reactiveGate;
    arduino.reactiveGate sends pin_level_changed
      (pin = buzzerPin (level = '1')) to tester.reactiveGate;
    arduino.reactiveGate sends pin_level_changed
      (pin = buzzerPin (level = '0')) to tester.reactiveGate;
    arduino.reactiveGate sends pin_level_changed
      (pin = buzzerPin (level = '1')) to tester.reactiveGate;
    arduino.reactiveGate sends pin_level_changed
      (pin = buzzerPin (level = '0')) to tester.reactiveGate;
  }
```

Test case	Result	Description
test1	FAIL	
Message#1	PASS	PASS
Message#2	PASS	PASS
Message#3	PASS	PASS
Message#4	PASS	PASS
Message#5	PASS	PASS
Message#6	FAIL	There is no received e
Message#7	FAIL	There is no received e
test2	PASS	



Outillage au sein du GEMOC Studio



Outil pour le calcul de la couverture du modèle et la localisation automatique des défauts

The screenshot displays the GEMOC Studio interface with several key components:

- Project Explorer (1):** Shows the project structure, including 'testCoverage.xmi' and 'testReport.xmi' under the 'testSuite.tdlan2' folder.
- Coverage View (2):** Displays a table of coverage results for various meta-classes and model elements across four tests and a test suite.
- Fault Localization View (4):** Shows SBFL information for different model elements, including coverage metrics and a ranking table.
- SBFL Technique (5):** A dropdown menu showing the selected technique 'phi'.

Coverage Table (3):

Meta-Class	Model Element	Test 1	Test 2	Test 3	Test 4	Test Suite
IntegerModuleGet		C	C	C	C	C
BinaryBooleanExpression	equal	C	C	C	C	C
IntegerConstant	1	-	-	-	-	-
IntegerModuleGet		C	C	C	C	C
Block		NC	C	NC	C	C
ModuleAssignment		NC	C	NC	C	C
IntegerConstant	0	-	-	-	-	-
		61.11	72.22	88.89	100.0	100.0

SBFL Information Table (6):

Meta-Class	Model Element	Test 1	Test 2	Test 3	Test 4	NCF	NUF	NCS	NUS	NC	NU	NS	NF	Susp	Rank
ModuleAssignment		C	C	C	C	2	0	2	0	4	0	2	2	0.0	6
If		C	C	C	C	2	0	2	0	4	0	2	2	0.0	6
Block		NC	C	NC	C	2	0	0	2	2	2	2	2	1.0	1
ModuleAssignment		C	C	C	C	2	0	0	2	2	2	2	2	1.0	1
Delay	MilliSecond	C	C	C	C	2	0	0	2	2	2	2	2	1.0	1
ModuleAssignment		NC	C	NC	C	2	0	0	2	2	2	2	2	1.0	1
Delay	MilliSecond	NC	C	NC	C	2	0	0	2	2	2	2	2	1.0	1
BinaryBooleanExpression	equal	C	C	C	C	2	0	2	0	4	0	2	2	0.0	6
IntegerModuleGet		C	C	C	C	2	0	2	0	4	0	2	2	0.0	6
BinaryBooleanExpression	equal	C	C	C	C	2	0	2	0	4	0	2	2	0.0	6
IntegerModuleGet		C	C	C	C	2	0	2	0	4	0	2	2	0.0	6
Block		NC	C	NC	C	1	1	1	1	2	2	2	2	0.0	6
ModuleAssignment		NC	C	NC	C	1	1	1	1	2	2	2	2	0.0	6
		PASS	PASS	FAIL	FAIL										

<https://github.com/lowcomote/Testing4DSLs>



Outillage au sein du GEMOC Studio



Outil de co-débugage des cas de test ayant échoué et de leur modèle sous test

The screenshot displays the GEMOC Studio IDE interface. The main editor shows a test case with the following code:

```
8 InfraRedSensor IRSensor(_name = "infraredSensor");
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 reactiveArduinoConfig
15 tester.reactiveGate sends button_pressed (button = button1)
16 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
20 to arduino.reactiveGate;
21 arduino.reactiveGate sends pin_level_changed
22 (pin = buzzerPin (level = '1')) to tester.reactiveGate;
23 arduino.reactiveGate sends pin_level_changed
24 (pin = buzzerPin (level = '0')) to tester.reactiveGate;
25 arduino.reactiveGate sends pin_level_changed
26 (pin = buzzerPin (level = '1')) to tester.reactiveGate;
27 arduino.reactiveGate sends pin_level_changed
28 (pin = buzzerPin (level = '0')) to tester.reactiveGate;
29 }
```

The left sidebar shows the Project Explorer with the following structure:

- run-test-reactive [Executable model with GEMOC Java engine]
 - Gemoc debug target
 - Model debugging
 - [Message] TestSuite4reactive.tdlan2//@packagedElement4/
 - [TestDescription] TestSuite4reactive.test1#executeTestCase()
 - [Package] TestSuite4reactive#main()
 - Engine : TestSuite4reactive.tdlan2 => TestSuite4reactive
- event_basedTesting [Executable model with GEMOC Java engine]
 - Gemoc debug target
 - Model debugging
 - [ModuleAssignment] sensorAlarm_withBug.model_oqHloNFg
 - [If] sensorAlarm_withBug.model_kSBuENFgEey60oAITKsoDC
 - [If] sensorAlarm_withBug.model_TeGOuLvnEey7OoLBzoKkPA
 - [InfraRedSensor] board.IR_sensorPin.infraredSensor#detect()
 - Engine : sensorAlarm_withBug.model => board.buttonPin.b

The bottom console shows the following output:

```
Default MessagingSystem console
[interface org.etsi.mts.tdl.connection, interface org.etsi.mts.tdl.Element
[interface org.etsi.mts.tdl.GateReference, interface org.etsi.mts.tdl.El
Start test case execution: test1
Button button1 pressed
The level of whiteLedPin pin changed to 1
Sensor infraredSensor detected
The level of buzzerPin pin changed to 1
```

The right sidebar shows the Variables window with the following data:

Name	Value
self	org.imt.arduino
module	org.imt.arduino
name	buzzer
operand	org.imt.arduino
value	1
level ((DigitalPin) board.IR_sensorPin)	1
level ((DigitalPin) board.buttonPin)	1
level ((DigitalPin) board.buzzerPin)	1
level ((DigitalPin) board.whiteLedPin)	1

The bottom right corner shows the Multidimensional Timeline window with a diagram of execution states.

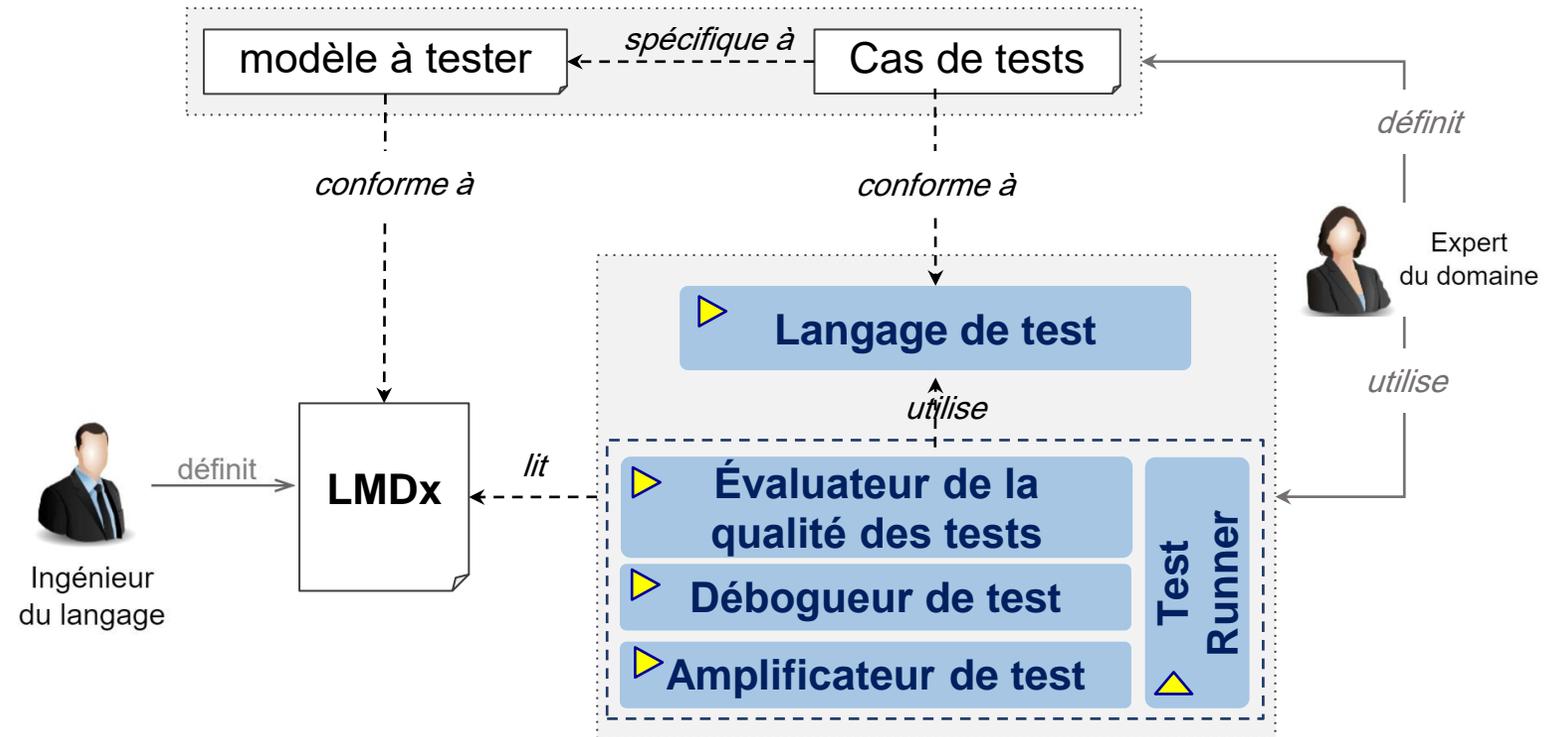
débugage pas-à-pas du cas de test qui a échoué

débugage pas-à-pas du modèle testé



<https://github.com/lowcomote/Testing4DSLs>

Conclusion



Publications: [JOT 2020](#), [SoSym 2021](#), [MODELS 2022](#), [SLE 2022](#)

En cours de révision: [JSS 2023](#) (SLE extension), SoSym 2023 (MODELS extension)

Un Environnement de Test pour les Langages Dédiés Exécutables

Faezeh Khorram

GDR-GPL 2022 Prix de thèse, l'accessit

NaoMod, LS2N, IMT Atlantique

Directeur de thèse

- Prof. Gerson SUNYE, Nantes Université, France

Encadrants

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

Soutenu le 12/12/2022