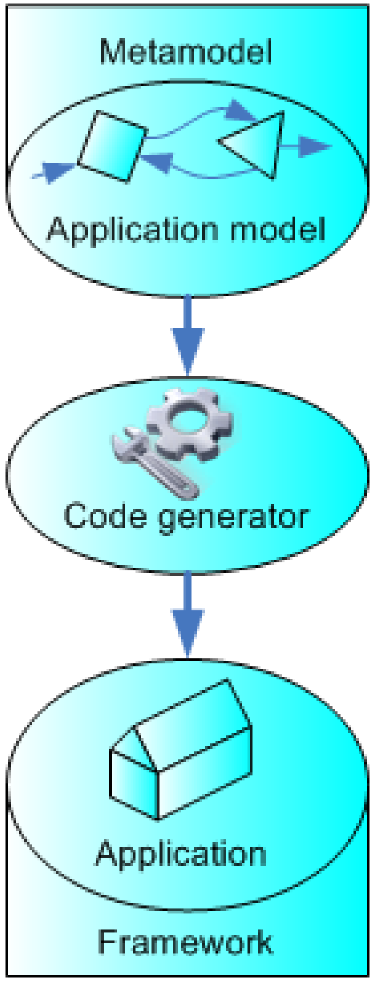
Using Model-Based Testing for Testing Application Models in the Context of Domain-Specific Modelling

Olli-Pekka Puolitaival

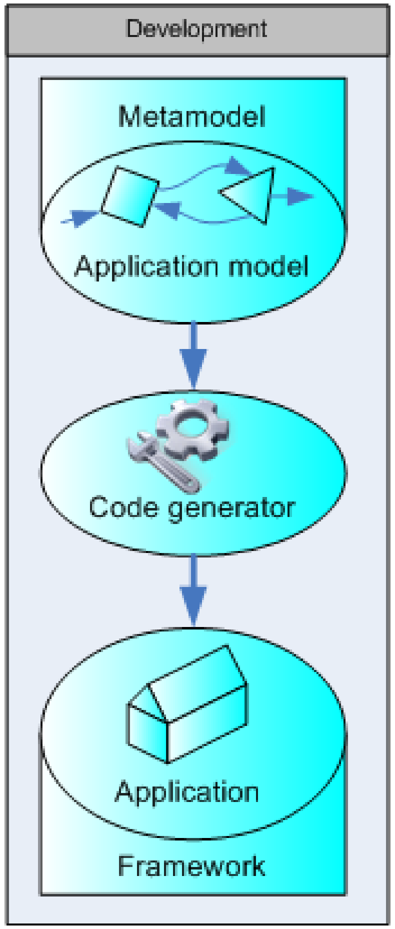
Business from technology

Index

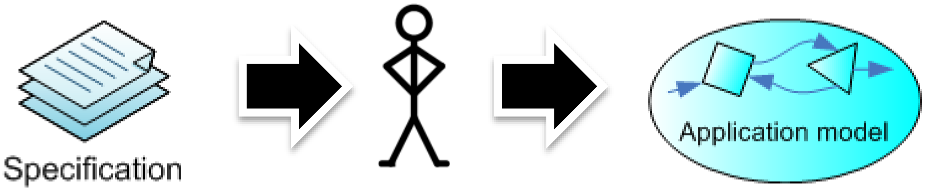
Domain-Specific Modeling



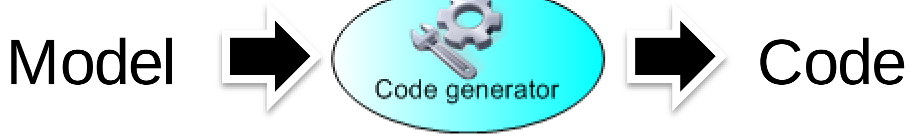
Where the Bug Lures?



1. Application model:



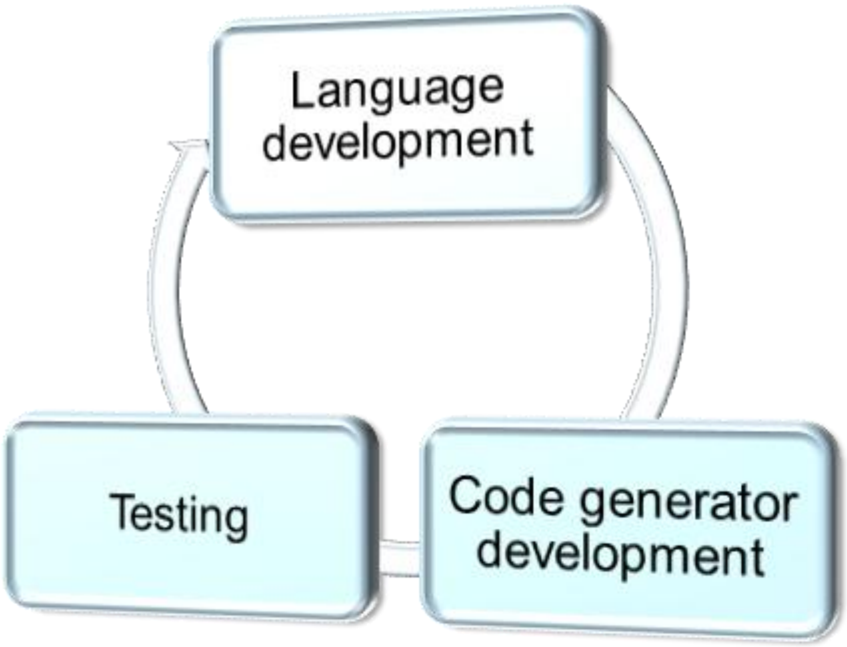
2. Code generator:



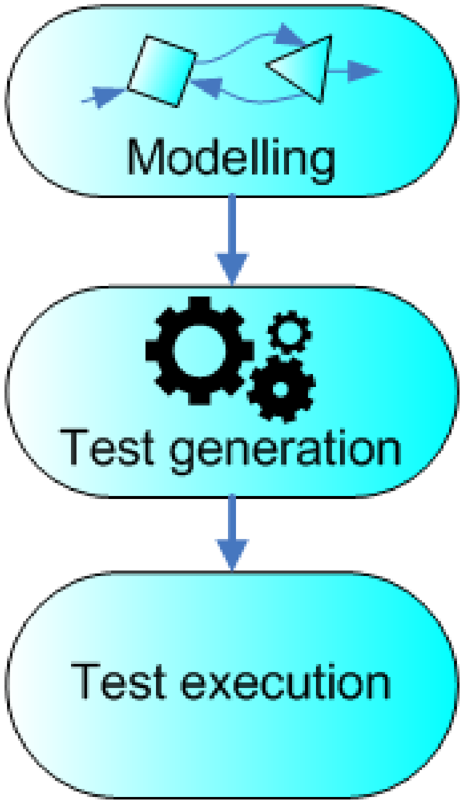
3. Framework:

Testing in the Context of DSM

- Iterative and incremental language development
  - Code generator produces same fault many places and therefore those are easy to find out
- Problems
  - Ad-hoc testing
  - Test coverage?
  - Test maintenance?
- More systematic and automated testing is required

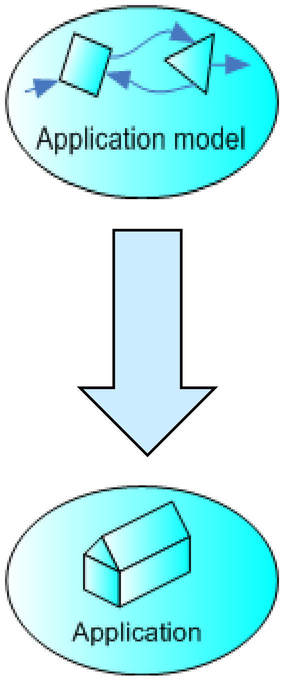


Model-Based Testing (MBT)



- Model-Based Testing (MBT) is about generating test cases from a model
- Idea:
  1. Make a high level behaviour model of system under testing
  2. MBT tool generates test suite automatically
  3. Run tests in your test execution environment
- Benefits of MBT
  - Reduces maintenance effort
  - Increases test coverage

Testing Presuppositions

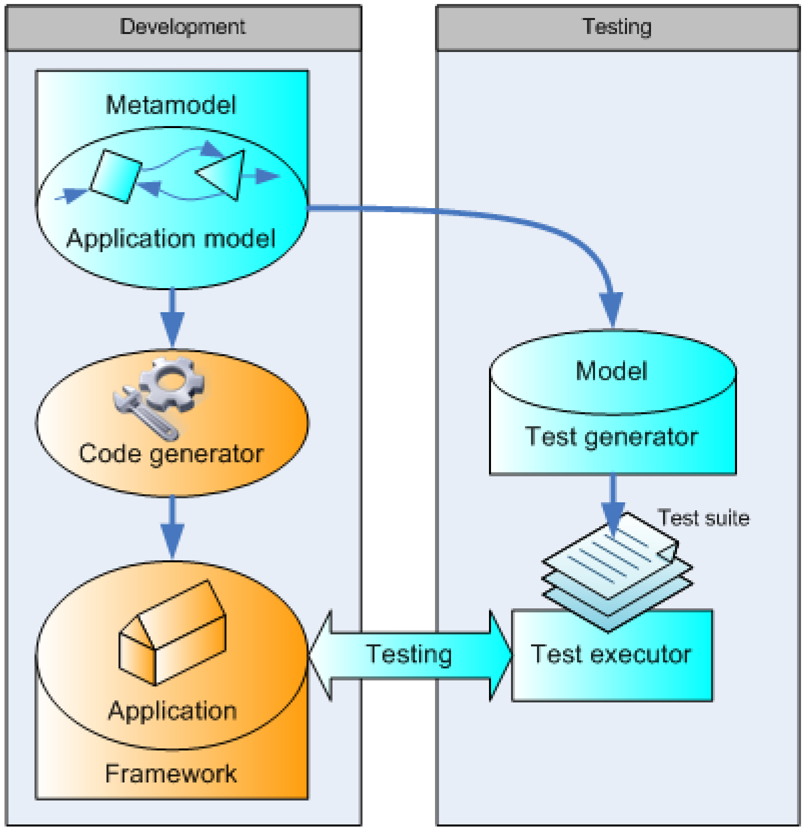


= Fact

= Testing / Verification

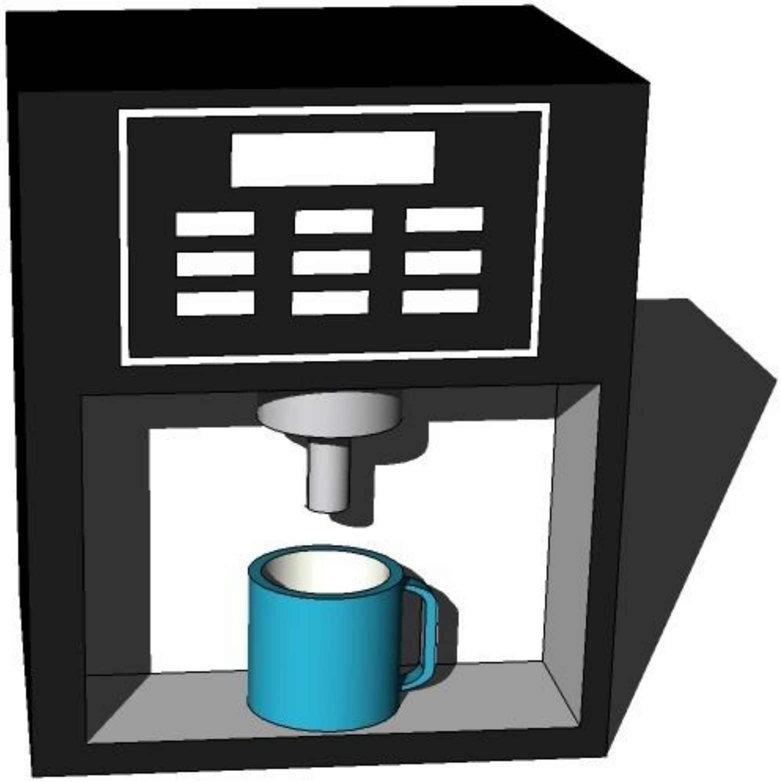
= System Under Testing

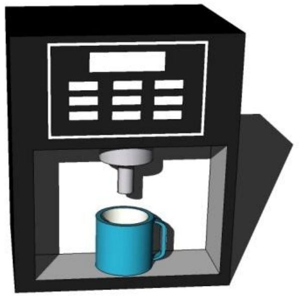
How to Find Bugs?



Case study:

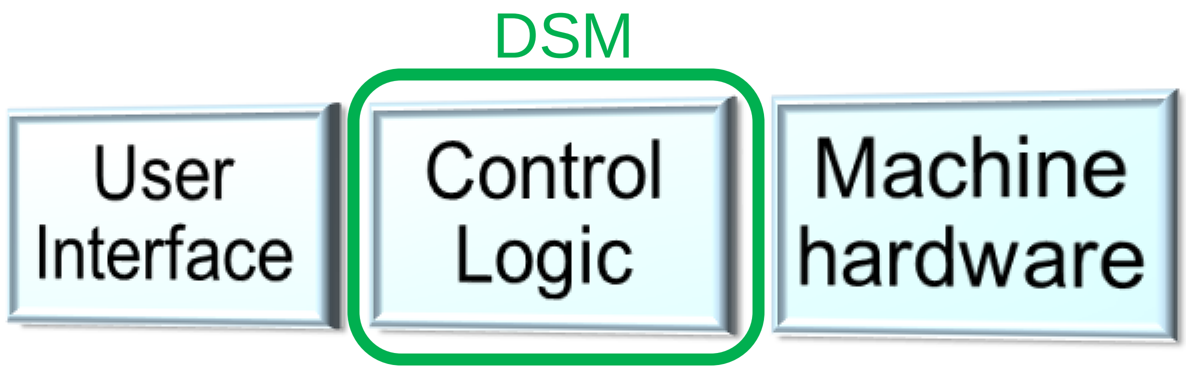
COFFEE MACHINE

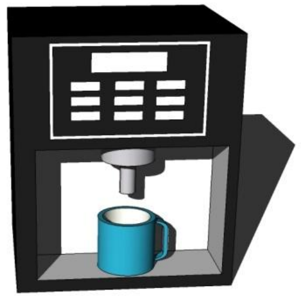




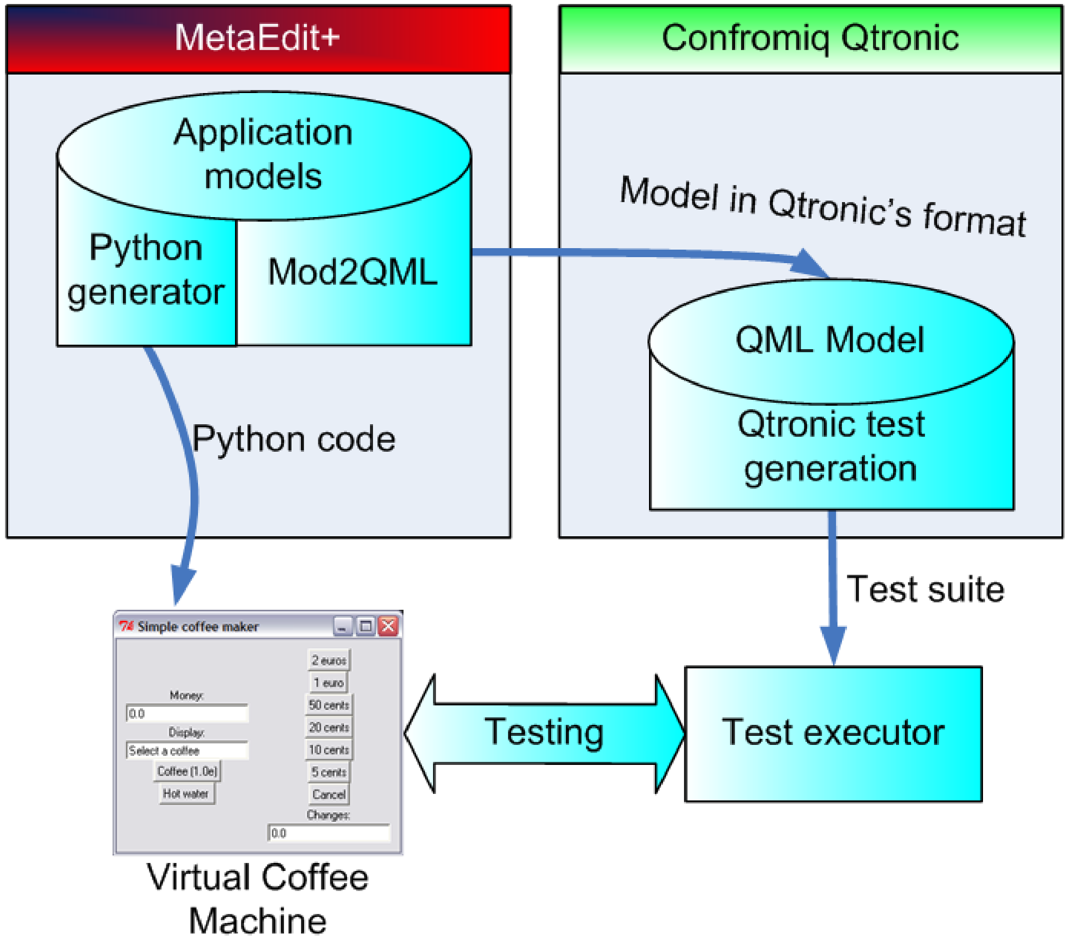
Demonstration Structure

- Architecture of the coffee machine:
  - User interface = User interface hardware
  - Control logic = Control logic code
  - Machine hardware = Make coffee in practice: add water, warm water, add coffee….
- **DSM is used for modelling control logic part**
- Also other parts are generated for demonstration





Demo Setup

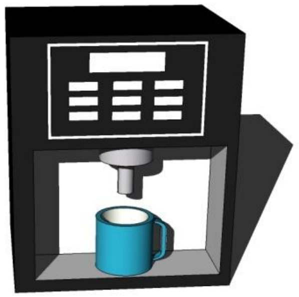


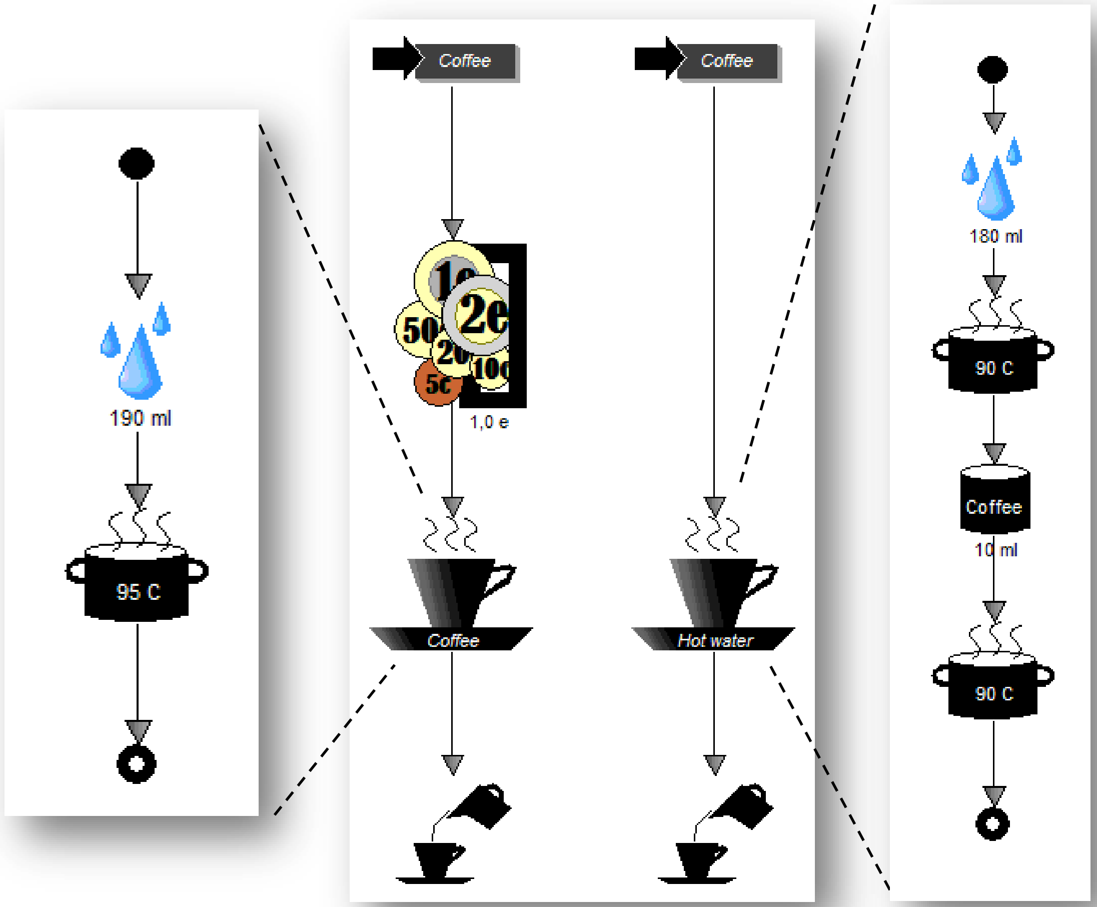
Money: 0.0

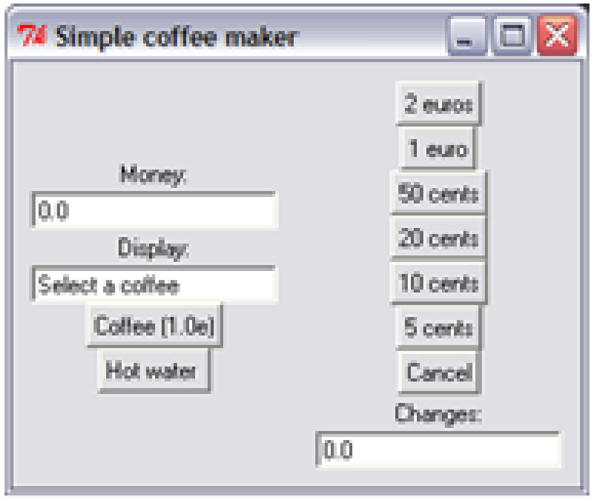
Display: Select a coffee

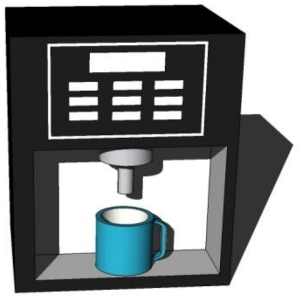
Changes: 0.0

DSML for coffee machine

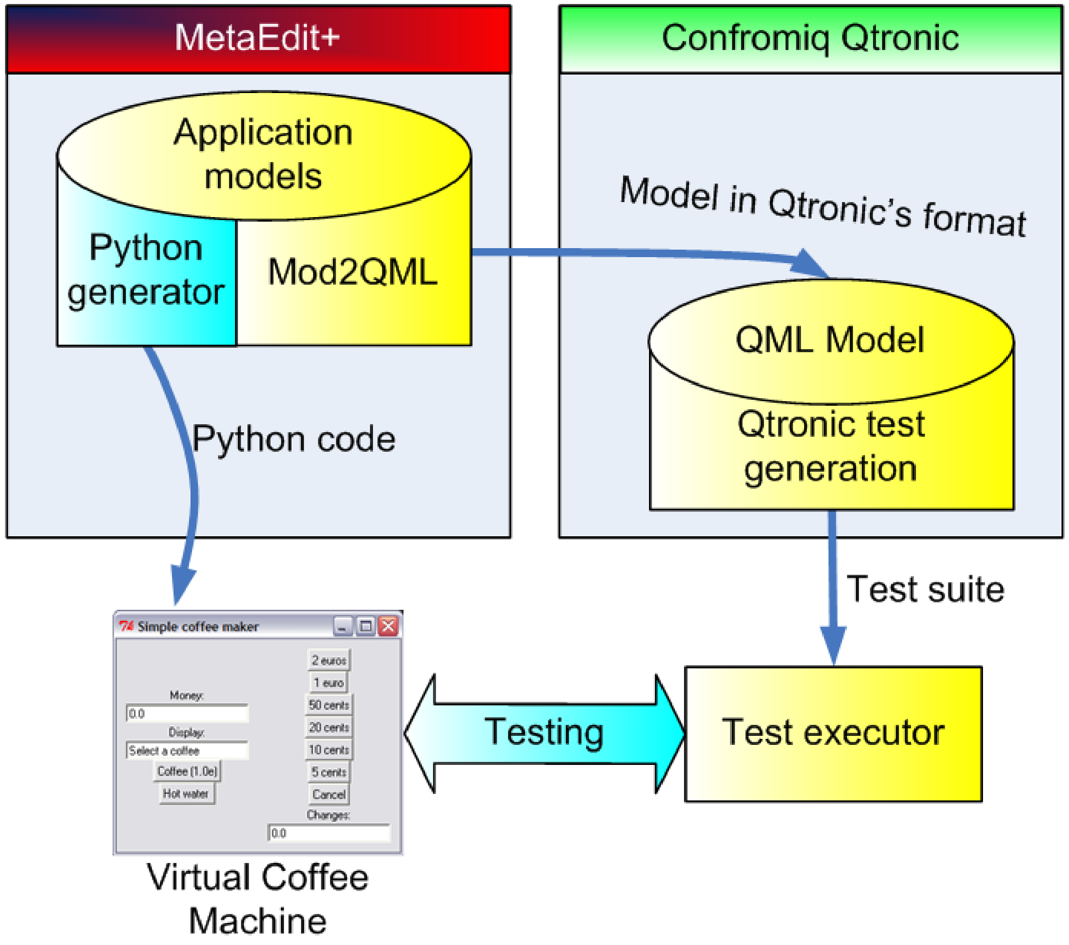




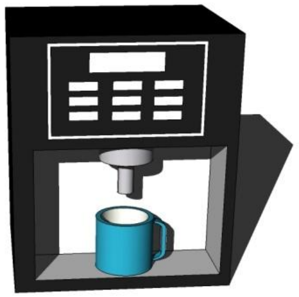


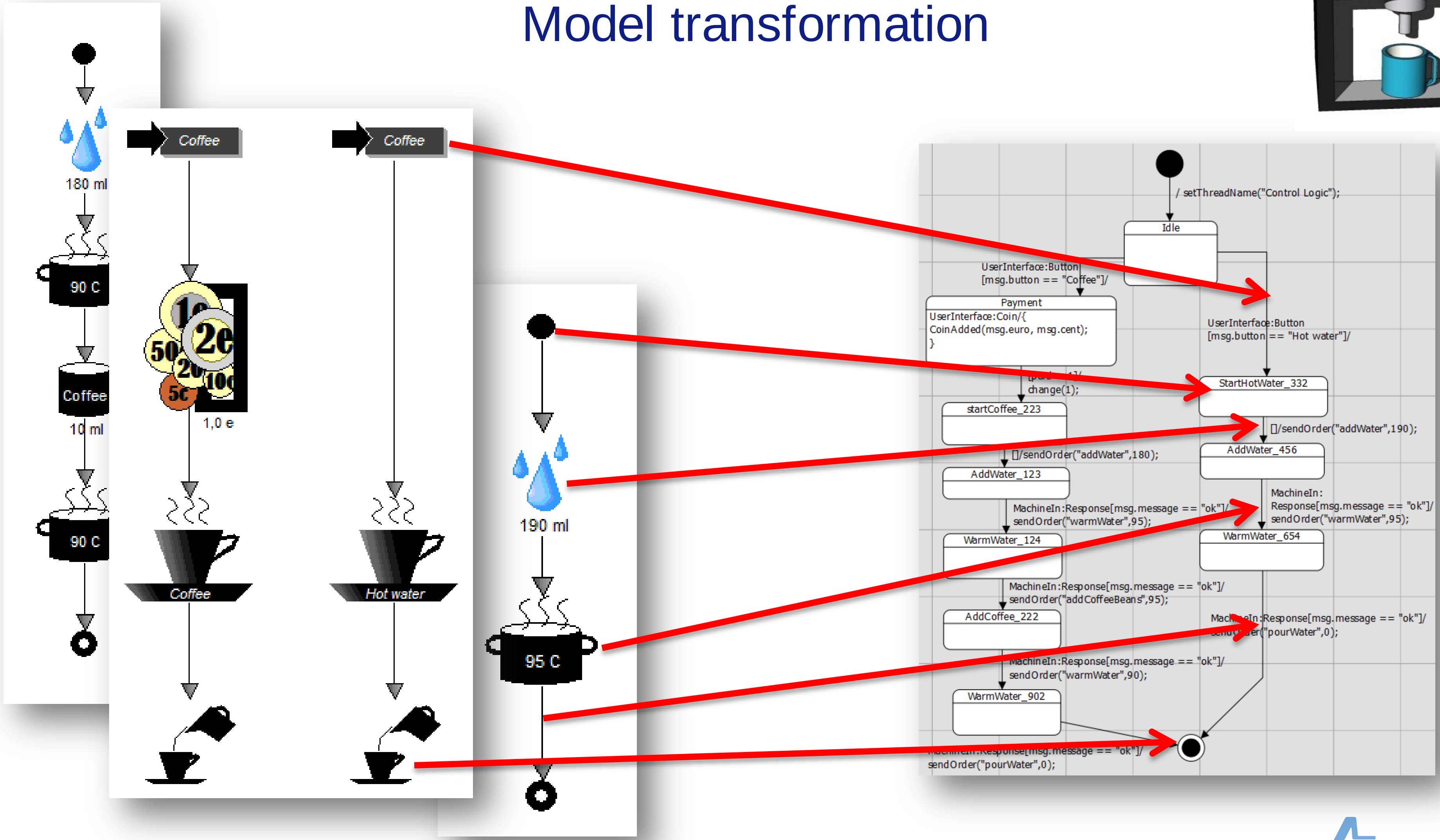


Model transformation

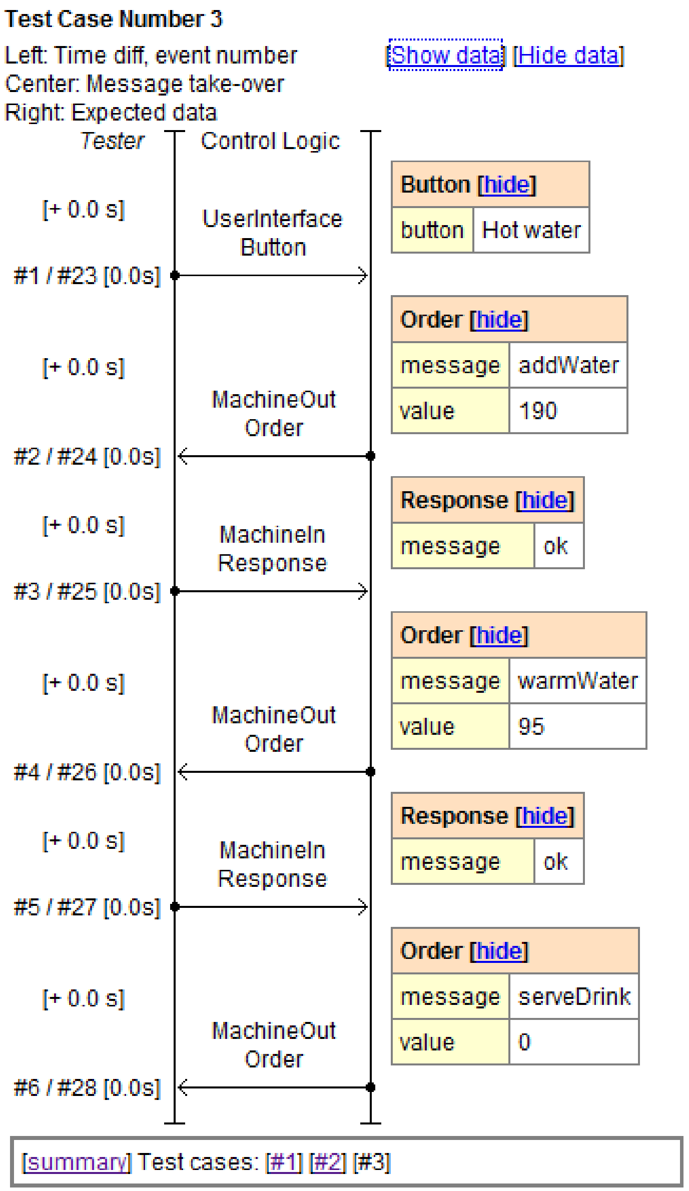


Model transformation




180 ml
90 C
Coffee
10 ml
90 C
Coffee
Coffee
1,0 e
Coffee
Hot water
190 ml
95 C
/ setThreadName("Control Logic");
Idle
UserInterface:Button
[msg.button == "Coffee"]/
Payment
UserInterface:Coin/{
CoinAdded(msg.euro, msg.cent);
}
change(1);
UserInterface:Button
[msg.button == "Hot water"]/
StartHotWater_332
startCoffee_223
[]/sendOrder("addWater",190);
[]/sendOrder("addWater",180);
AddWater_456
AddWater_123
MachineIn:
Response[msg.message == "ok"]/
sendOrder("warmWater",95);
MachineIn:Response[msg.message == "ok"]/
sendOrder("warmWater",95);
WarmWater_124
WarmWater_654
MachineIn:Response[msg.message == "ok"]/
sendOrder("addCoffeeBeans",95);
AddCoffee_222
MachineIn:Response[msg.message == "ok"]/
sendOrder("pourWater",0);
MachineIn:Response[msg.message == "ok"]/
sendOrder("warmWater",90);
WarmWater_902
MachineIn:Response[msg.message == "ok"]/
sendOrder("pourWater",0);

Generated test cases

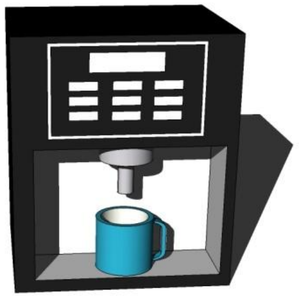

Test Case Number 3

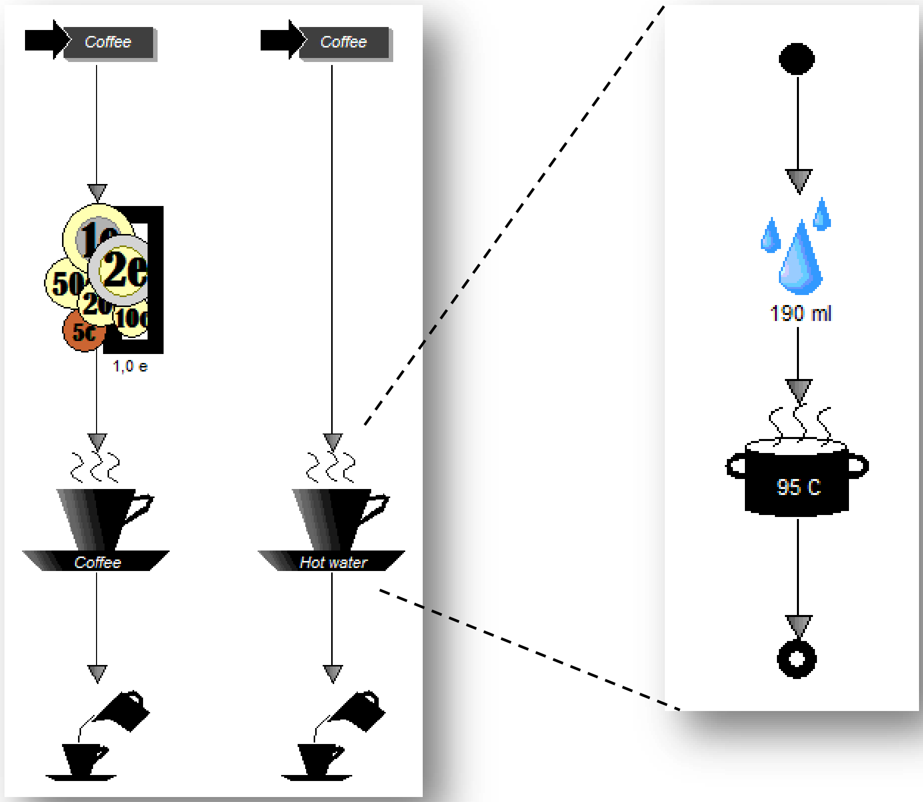
Left: Time diff, event number
Center: Message take-over
Right: Expected data

[Show data] [Hide data]

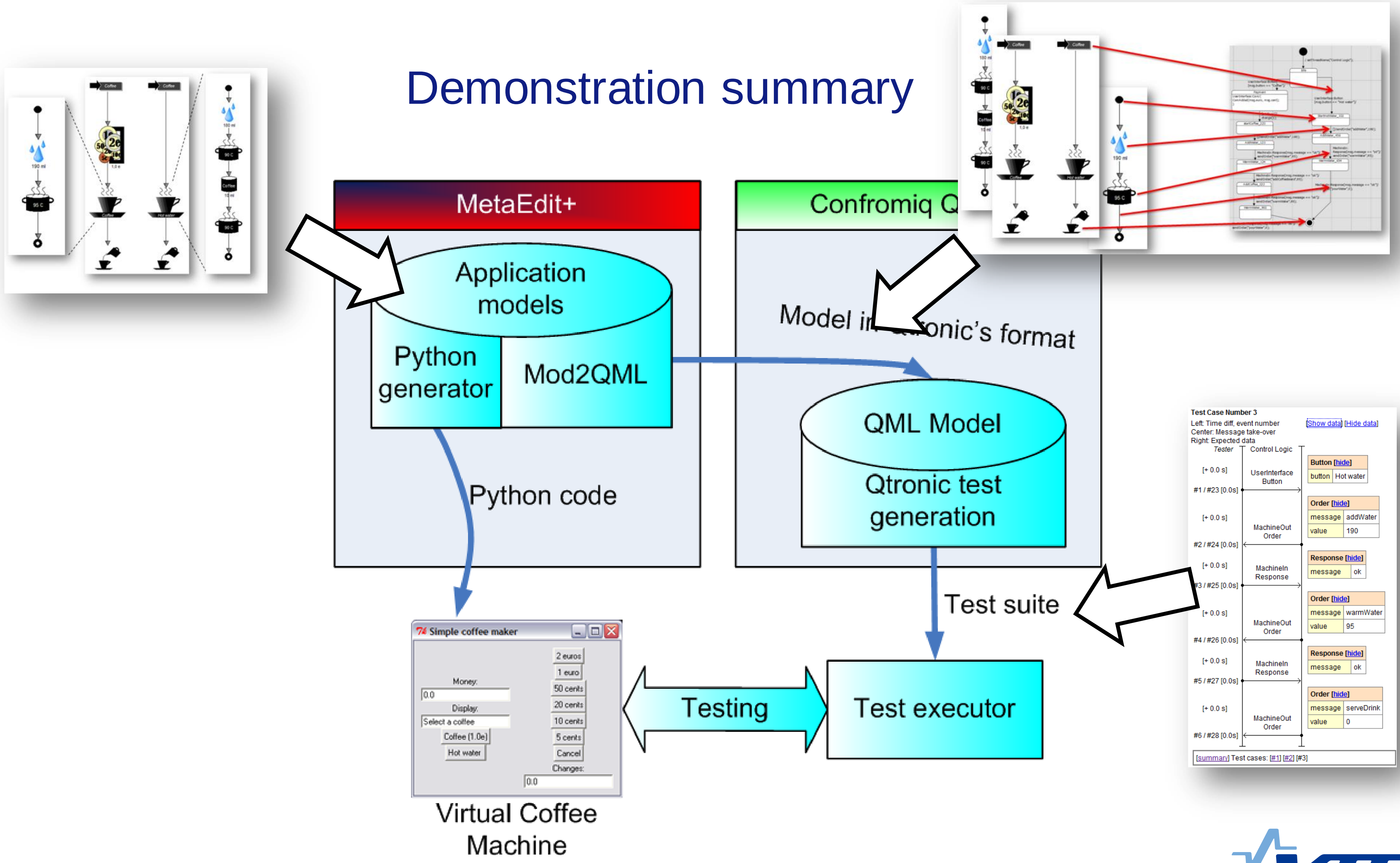
Tester	Control Logic	Expected data
[+ 0.0 s] #1 / #23 [0.0s]	UserInterface Button →	Button [hide]: button = Hot water
[+ 0.0 s] #2 / #24 [0.0s]	MachineOut Order ←	Order [hide]: message = addWater; value = 190
[+ 0.0 s] #3 / #25 [0.0s]	MachineIn Response →	Response [hide]: message = ok
[+ 0.0 s] #4 / #26 [0.0s]	MachineOut Order ←	Order [hide]: message = warmWater; value = 95
[+ 0.0 s] #5 / #27 [0.0s]	MachineIn Response →	Response [hide]: message = ok
[+ 0.0 s] #6 / #28 [0.0s]	MachineOut Order ←	Order [hide]: message = serveDrink; value = 0

[summary] Test cases: [#1] [#2] [#3]





Demonstration summary



Test Case Number 3

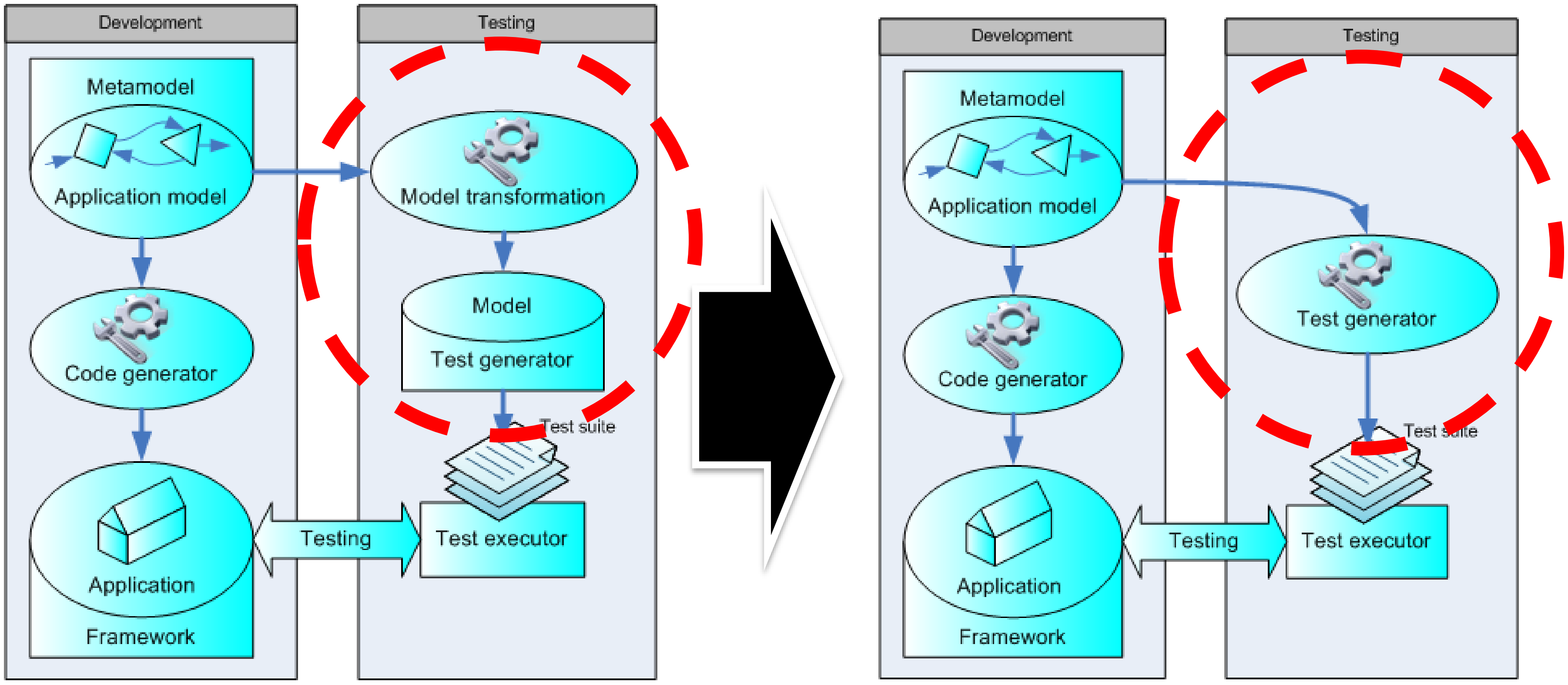
Left: Time diff, event number
Center: Message take-over
Right: Expected data

[Show data] [Hide data]

```
Tester              Control Logic
[+ 0.0 s]           UserInterface      Button [hide]
                    Button             button | Hot water
#1 / #23 [0.0s]  -->
[+ 0.0 s]           MachineOut         Order [hide]
                    Order              message | addWater
                                       value   | 190
#2 / #24 [0.0s]  <--
[+ 0.0 s]           MachineIn          Response [hide]
                    Response           message | ok
#3 / #25 [0.0s]  -->
[+ 0.0 s]           MachineOut         Order [hide]
                    Order              message | warmWater
                                       value   | 95
#4 / #26 [0.0s]  <--
[+ 0.0 s]           MachineIn          Response [hide]
                    Response           message | ok
#5 / #27 [0.0s]  -->
[+ 0.0 s]           MachineOut         Order [hide]
                    Order              message | serveDrink
                                       value   | 0
#6 / #28 [0.0s]  <--
```

[summary] Test cases: [#1] [#2] [#3]

Future Research: Passing model transformation



Future Research

- Language testing using application generation (see my presentation in OOPSLA 2008 DSL workshop)

- Adopting DSM in testing area:
    - Domains specific model-based testing
    - Test case visualization using DSM

Thank you!