Temporal Properties for Testing

Frédéric Dadeau – FEMTO-ST/DISC

Master 2 Informatique

UE Spécification et Vérification Automatique de Modèles

Mercredi 8 Janvier 2020

Syllabus

Goal of this lecture:

- Provide an overview of how to use temporal properties in a software validation process involving models.

At the end of this lecture, you will be able:

- to explain the possible usages of temporal properties in the software validation process
- to express properties using temporal patterns
- to translate temporal pattern properties into automata
- to implement monitors that detect the violation of a property or evaluate its coverage

Prerequisites

You should already be able to:

- write a temporal property using LTL (using appropriate symbols)
- transform an LTL property into a Büchi automaton
- explain how to model-check a temporal property
- give an example of an indecidable problem ☺

- explain what is all of this used for...

Agenda

1. Dwyer's temporal property patterns
2. Principles of Model-Based Testing
3. Use of properties for testing
4. Your assignment for the next two weeks

Agenda

1. Dwyer's temporal property patterns
   - what is it?
   - how does it work?
   - examples
2. Principles of Model-Based Testing
3. Use of properties for testing
4. Your assignment for the next two weeks

Temporal Property Patterns - what is it?

- 20 years ago, Matt Dwyer, Georges Avrurnin and James Corbett defined a set of temporal patterns, aiming to simplify the writing of temporal properties [DAC99].

- Property = Pattern + Scope
  - Pattern: describes occurrences or orderings of events
  - Scope: describes the observation window on which the pattern is supposed to hold

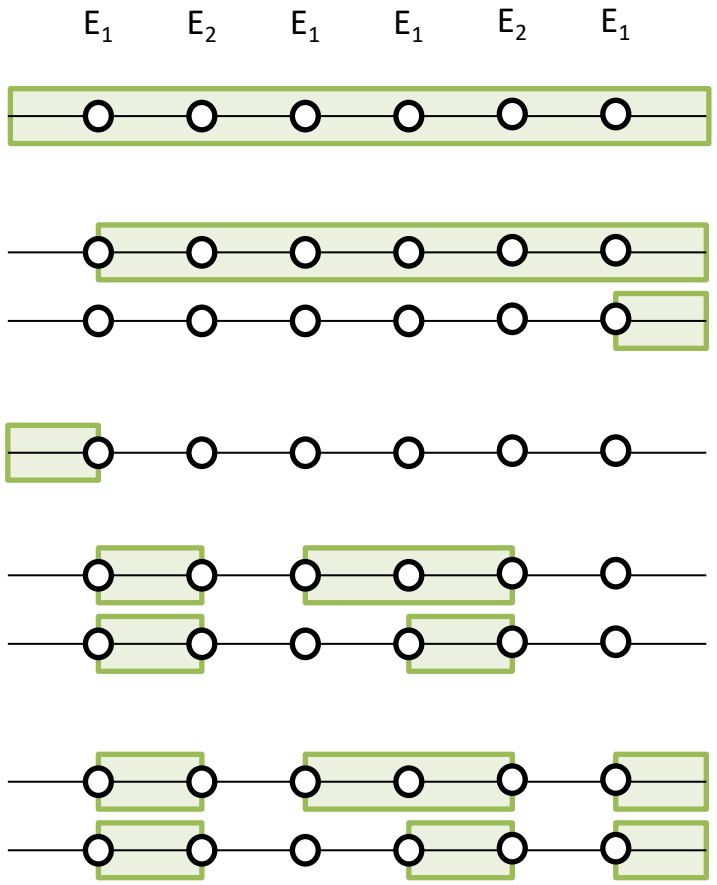
- 5 scopes + 8 patterns → 88% of temporal properties in various case studies.

- Easier to understand and practice by industrials…

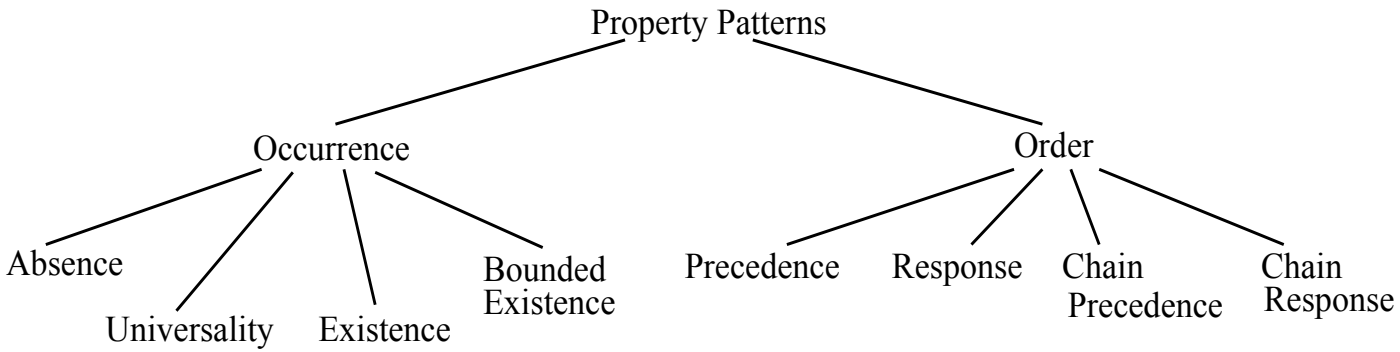
[DAC99] **Patterns in Property Specifications for Finite-state Verification**, *Matthew B. Dwyer, George S. Avrunin and James C. Corbett*. Proceedings of the 21st International Conference on Software Engineering, May, 1999.

Temporal Property Patterns - Scope

- globally
- after E_1
  - after last E_1
- before E_1
- between E_1 and E_2
  - between last E_1 and E_2
- after E_1 until E_2
  - after last E_1 until E_2



Temporal Property Patterns - Patterns



Usually defined using keywords

- always P
- never E
- eventually E at least/at most/exactly k times
- E_1 [directly] precedes E_2
- E_1 [directly] follows E_2

Temporal Property Patterns - Events

Events: left unspecified in the original paper

- Basic idea: what ever you want/can observe
- Can be (a combination of):
  - operation invocation (name, possibly with parameters and return values)
  - guard/precondition
  - postcondition
- At least you can use “keywords” representing abstract actions

Temporal Property Patterns - Examples

Requirement: « In order to buy tickets, I need to be authenticated. »

- Property 1

  never BUY_TICKET_OK
  before LOGIN_OK

- Property 2

  never BUY_TICKEY_OK
  after LOGOUT_OK
  until LOGIN_OK

- Property 3

  eventually BUY_TICKET_OK at least 0 times
  between LOGIN_OK
  and LOGOUT_OK

Syntax makes temporal pattern look like they're easy. However, usage needs some serious thinking... Nevertheless, it is still easier than using LTL or CTL... No?

Agenda

1. Dwyer's temporal property patterns
2. Principles of Model-Based Testing
3. Use of properties for testing
4. Your assignment for the next two weeks

Model-Based Testing – what is it?

Model-Based Testing (MBT) aims to automate conformance testing by using a formal model of the system which:

- → describes the inputs and/or
- → characterizes the outputs

Unlike other approaches (e.g. white-box testing using code coverage criteria), it especially targets (business) logics faults in the application.

- ➔ 15% of total faults in an application[1]

And sometimes, it is the only applicable technique…

- No access to the code,
- Other V&V tools/approaches can not be applied

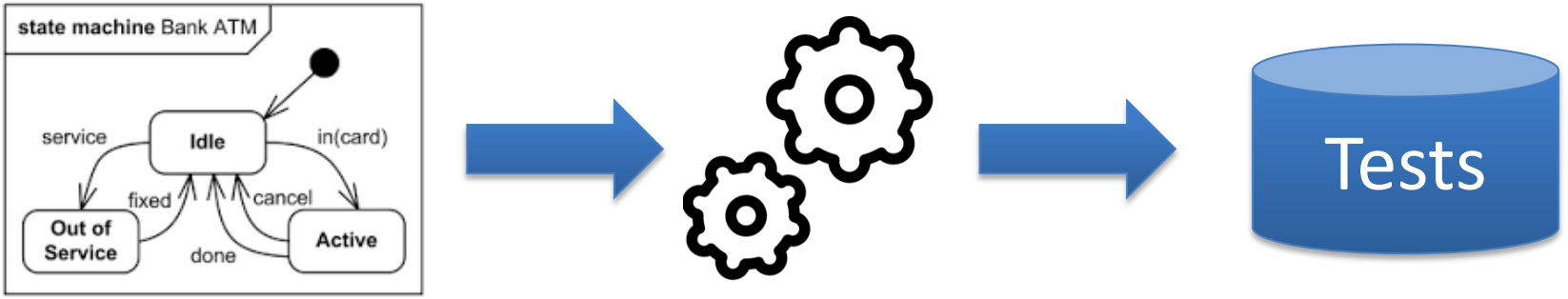
[1]Source: H. Hemmati. *How effective are code coverage criteria?* Int. Conf. on Quality Reliability and Security, 2015

Model-Based Testing – what is it?

```
/**
 * @requires s != null;
 * {|
 *    @requires (\exists int i; 0 <= i < s.length(); s.charAt(i) == c);
 *    @ensures \result >= 0 && \result < s.length() &&
 *      (\forall int i; 0 <= i < s.length(); s.chatAt(i) == c ==> \result <= i);
 *  also
 *    @requires (\forall int i; 0 <= i < s.length(); s.charAt(i) != c);
 *    @ensures \result < 0;
 * |}
 */
int findIndex(char c, String s) {
   int r = -42;
   for (int i=0; i <= s.length(); i++) {
      if (s.charAt(i) == c) {
         r = i;
      }
   }
   return r;
}
```

Is there any fault in this Java function?

Model-Based Testing – what is it?

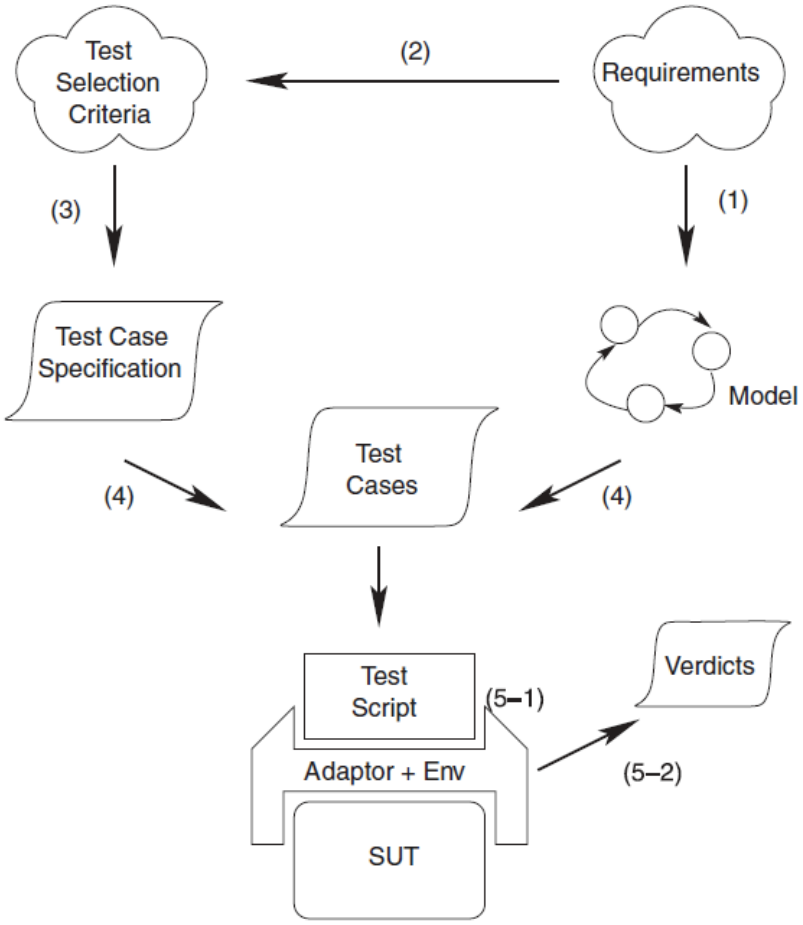


Model-Based Testing: testing based on or involving models

Test case: *A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.*

ISTQB Glossary 2015 – International Software Testing Qualifications Board - http://www.istqb.org – MBT extension

Model-Based Testing – what is it?



(1) Model design

(2) Choice of the appropriate test selection criteria

(3) Transformation of TS criteria into an « operational form » (algorithms)

(4) Test cases generation

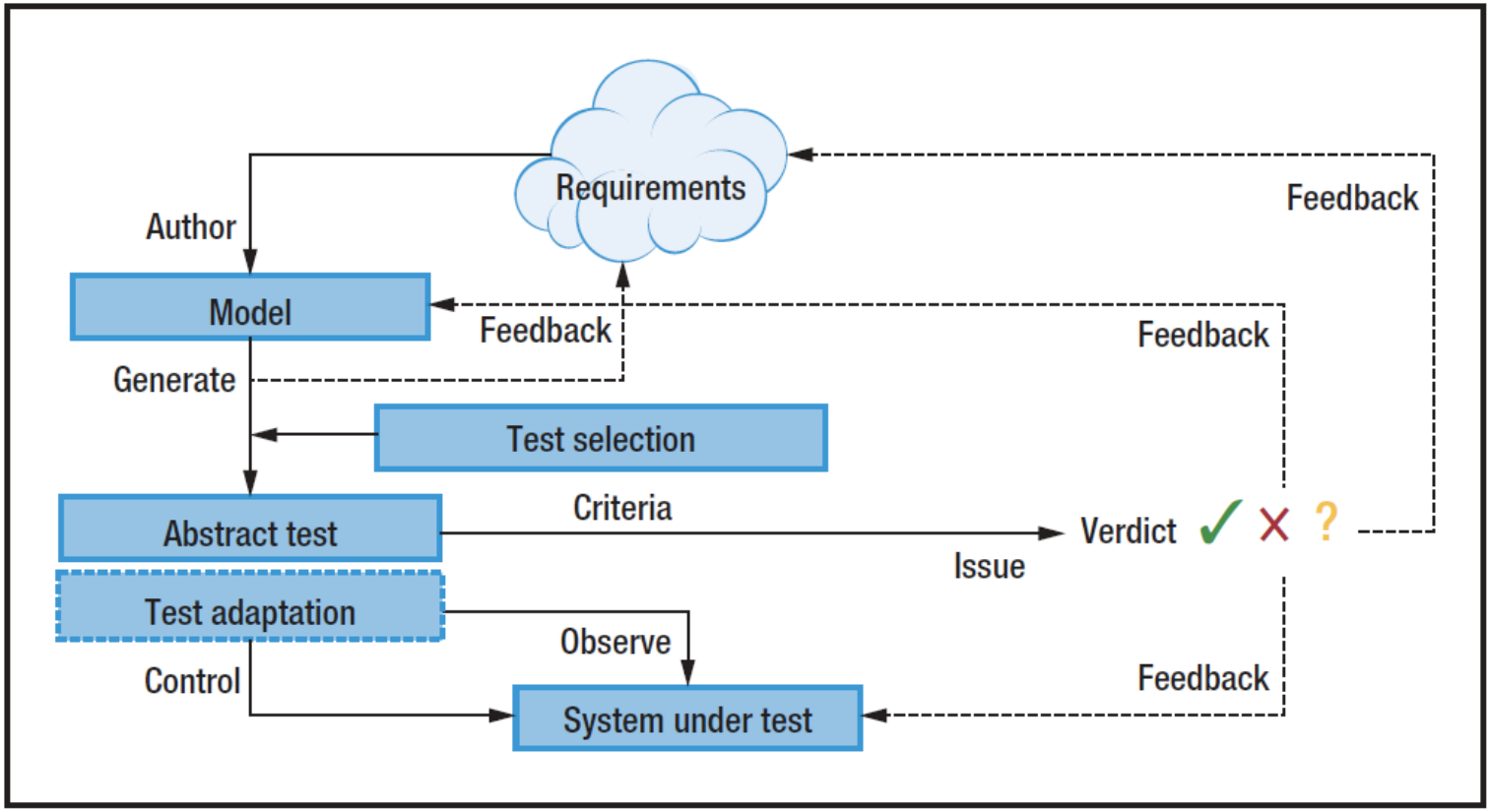
(5) Test execution

(5-1) Concretization of test cases

(5-2) Establishment of the test verdict

Source: M. Utting, A. Pretschner, B. Legeard
Taxonomy of MBT approaches. STVR, 22-5, 2012

Model-Based Testing – what is it?



Source: Methods for Testing & Specification (MTS); Model-Based Testing (MBT); Requirements for Modelling Notations, ES 202 951 v.1.1.1 European Telecommunications Standards Institute, 2011

Model-Based Testing – what is it?

In MBT approaches, models are used to:

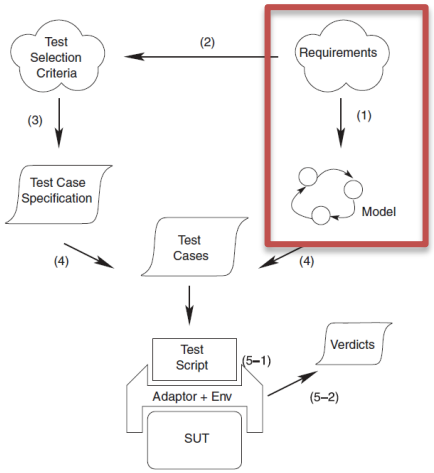
- compute test cases: test data and/or operation sequences
- predict the test oracle, and thus establish the test verdict

→ automate test generation & execution

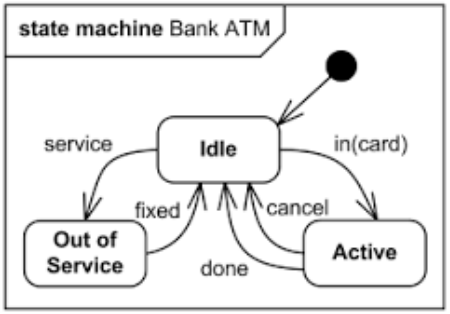
Scientific challenges:

- how to model the SUT?
- how to exploit the model to derive the tests?
- how to bridge the gap between the (abstract) model and the (concrete) SUT?
- how to establish the conformance between the model and the SUT?

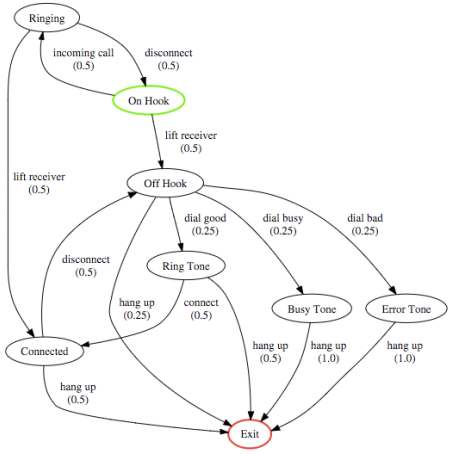
MBT challenges – Model Design



- Transition-based notations: (Extended-)Finite State Machines, Mealy Machines, statecharts, (Input-Output) Labelled Transitions Systems
- Pre-Post or Input Domains (a.k.a. State-based) notations: B (generalized substitutions), Z, JML (contracts), OCL (pre/post), SpecExplorer
- Operational notation: process algebra such as CSP, VHDL, Petri Nets, HLPSL
- Stochastic notation: markov chains



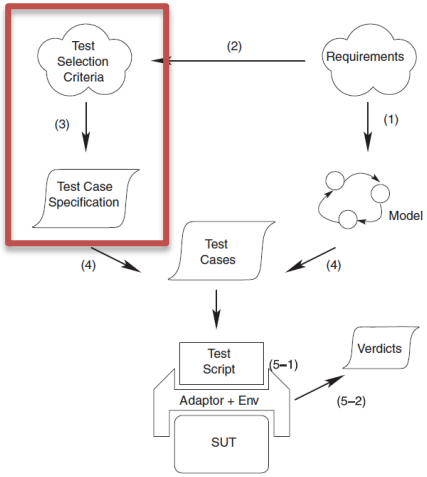

```
if (p_Code=CODE::FAUX)then
    ---@REQ:COUNTER_DECREASED
    carte.nbEssaisCode = carte.nbEssaisCode-1 and
    afficherMessage(MESSAGE::CODE_ERRONE) and
    if (carte.nbEssaisCode <= 0) then
        ---@REQ:CARD_BLOCKED
        afficherMessage(MESSAGE::CARTE_BLOQUEE) and
        restituerCarte()
    else
        ---@REQ:CARD_NOT_BLOCKED
        afficherMessage(MESSAGE::ENTRER_CODE)
    endif
else
    ---@REQ:OK
    statut = STATUTDAB::ATTENTE_SAISIE_MONTANT and
```




```
role alice(
  A,B     : agent,
  K       : symmetric_key,
  Hash    : hash_func,
  SND,RCV : channel(dy))
played_by A def=

  local
    State   : nat,
    Na,Nb   : text,
    K1      : message

  init
    State := 0

  transition

  1. State  = 0 /\ RCV(start) =|>
     State':= 2 /\ Na' := new()
                /\ SND({Na'}_K)

  2. State  = 2 /\ RCV({Nb'}_K) =|>
     State':= 4 /\ K1' := Hash(Na.Nb')
                /\ SND({Nb'}_K1')
                /\ witness(A,B,bob_alice_nb,Nb')

end role
```

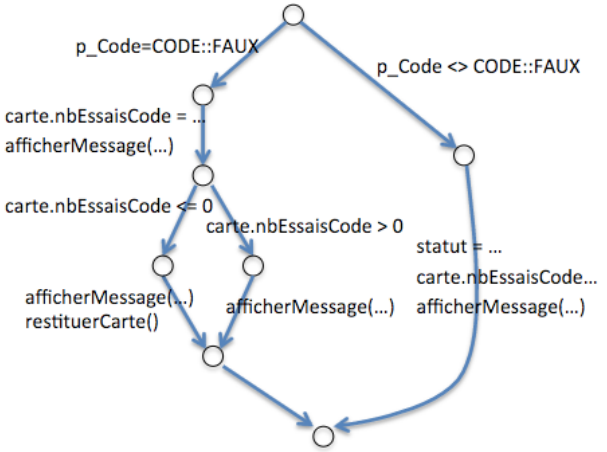
```
role bob (A,B: agent,
          Ka: public_key,
          Kb: public_key,
          Snd,Rcv: channel(dy))
played_by B
def=

  local State   : nat,
        Na,Nb : text
  const sec_k2 : protocol_id
  init  State := 0

  transition
   1. State = 0 /\ Rcv({Na'.A}_Kb)
      =|>
      State' := 1
      /\ Nb' := new()
      /\ Snd({Na'.Nb'}_Ka)
      /\ witness(B,A,nb,Nb')
      /\ secret(Nb',sec_k2,{A,B})
   2. State = 1 /\ Rcv({Nb}_Kb)
      =|>
      State' := 2
      /\ request(A,B,na,Na)
end role
```

MBT challenges – Model exploitation



Test generation criteria:

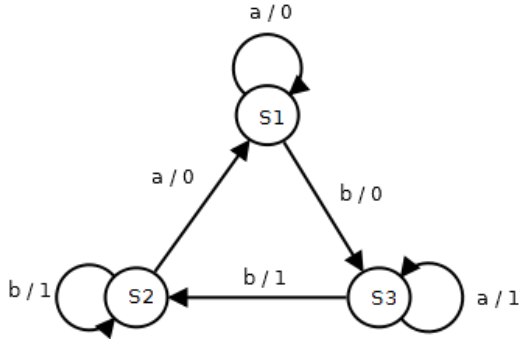
- structural coverage: nodes, arcs, etc.
- data coverage: boundary values, pairwise, etc.
- requirement coverage
- test case specification: test purpose, scenario
- stochastic coverage: usage model
- fault-based




```
1if (p_Code=CODE::FAUX)then
2    ---@REQ:COUNTER_DECREASED
3    carte.nbEssaisCode = carte.nbEssaisCode-1 and
4    afficherMessage(MESSAGE::CODE_ERRONE) and
5    if (carte.nbEssaisCode <= 0) then
6        ---@REQ:CARD_BLOCKED
7        afficherMessage(MESSAGE::CARTE_BLOQUEE) and
8        restituerCarte()
9    else
0        ---@REQ:CARD_NOT_BLOCKED
1        afficherMessage(MESSAGE::ENTRER_CODE)
2    endif
3else
4    ---@REQ:OK
5    statut = STATUTDAB::ATTENTE_SAISIE_MONTANT and
```

Test generation technologies :

- model-checking
- theorem proving
- symbolic execution
- random
- machine learning/AI



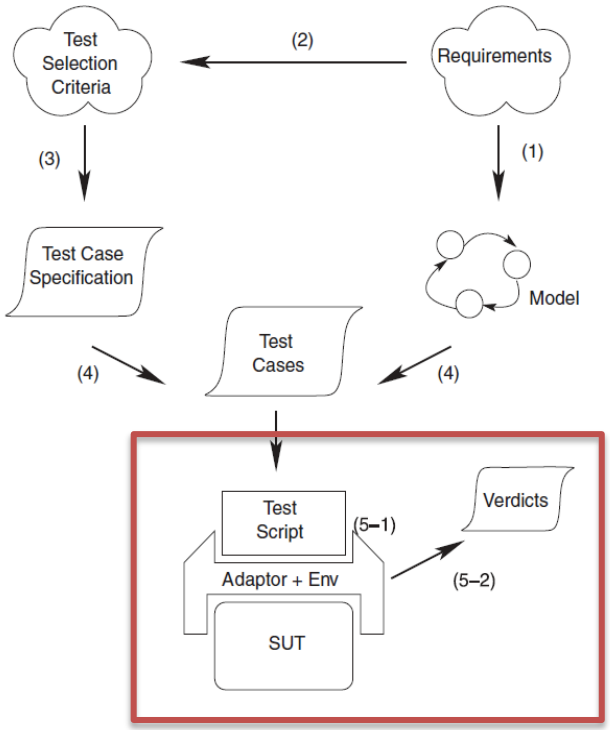
MBT challenges – Establishing conformance (oracle problem)

More or less intrusive techniques:

- analyze execution traces (logs)
- assertions in the tests cases (on public/observable data/artifacts) on expected results
- instrumentation of the code, notably with runtime assertion checking (e.g. JML-RAC)

Conformance relationships:

- simulation, bi-simulation (trace equivalence), ioco (trace inclusion), etc.



MBT challenges - Conformance relationship

Reasonable compromise: **ioco** defined on IOLTS

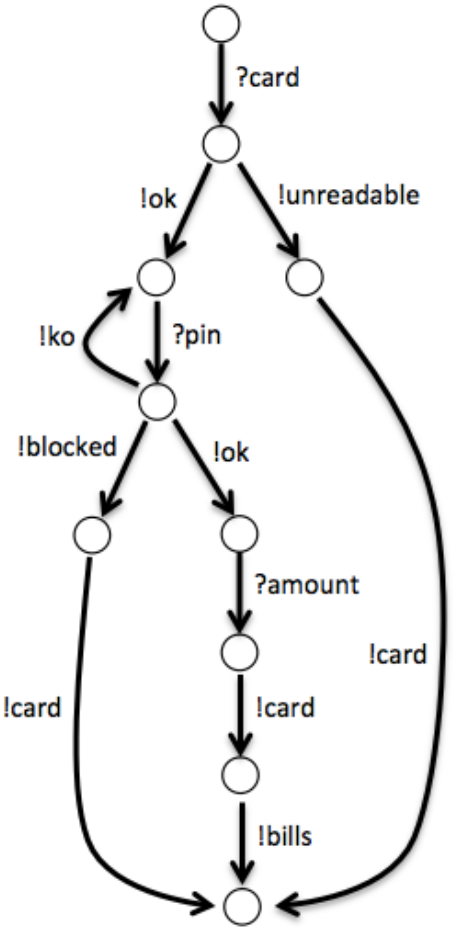
IOLTS = < Q, A, →, q0 >

- Q = set of states
- A = $A_i \cup A_o \cup \{\tau\}$ with
  - A_i = input actions (prefixed by ?)
  - A_o = output actions (prefixed by !)
  - τ = internal action
- → ⊆ Q × A × Q
- q0 = initial state

δ : quiescence (observation of no output) : deadlock/livelock

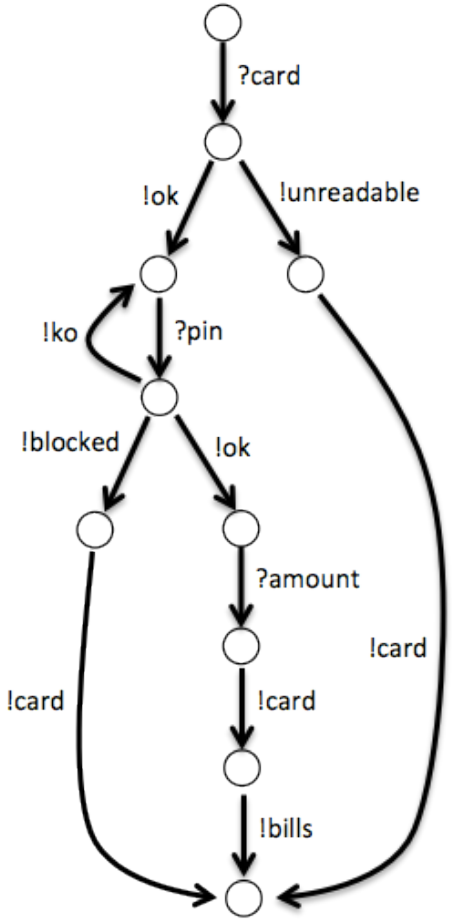
IUT ioco S: ∀σ : Straces(S) : out(IUT after σ) ⊆ out(S after σ)

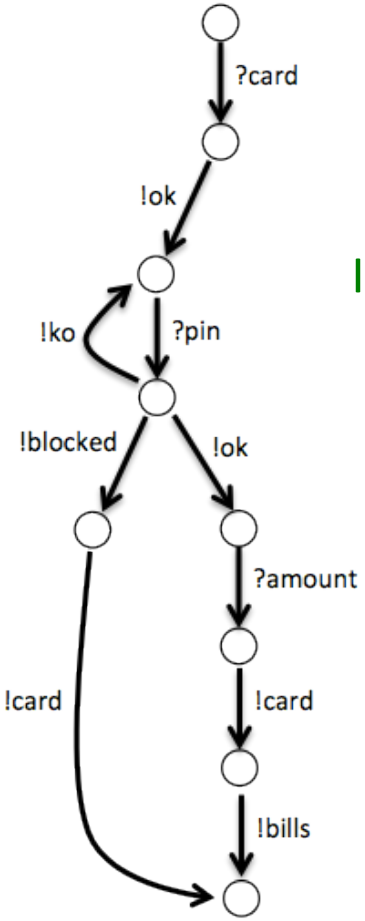
After each suspended trace (ie. an execution up to a quiescence), IUT exhibits only outputs and quiescences present in S.



- J Tretmans. *Test generation with inputs, outputs and repetitive quiescence.* Software---Concepts and Tools, 1996

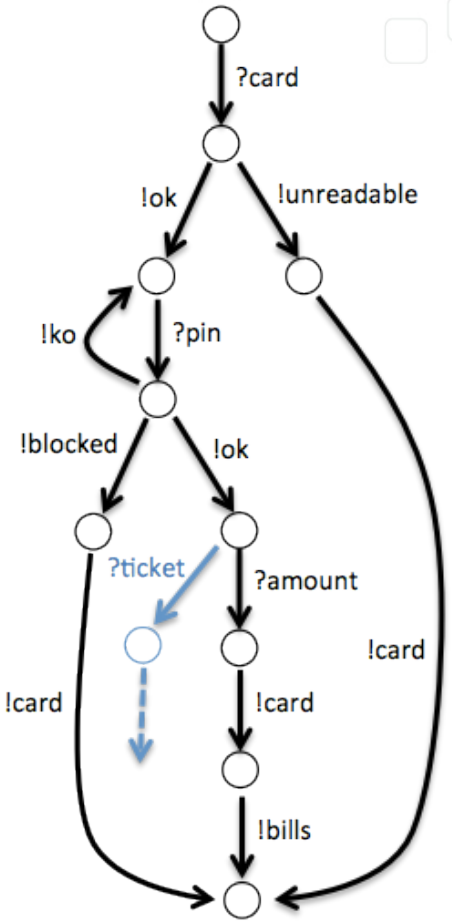
MBT challenges - Conformance relationship





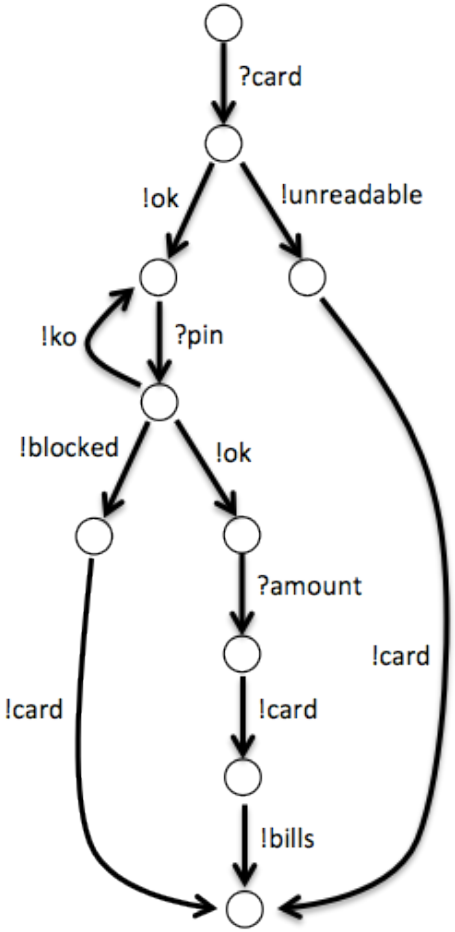
Implementation choice

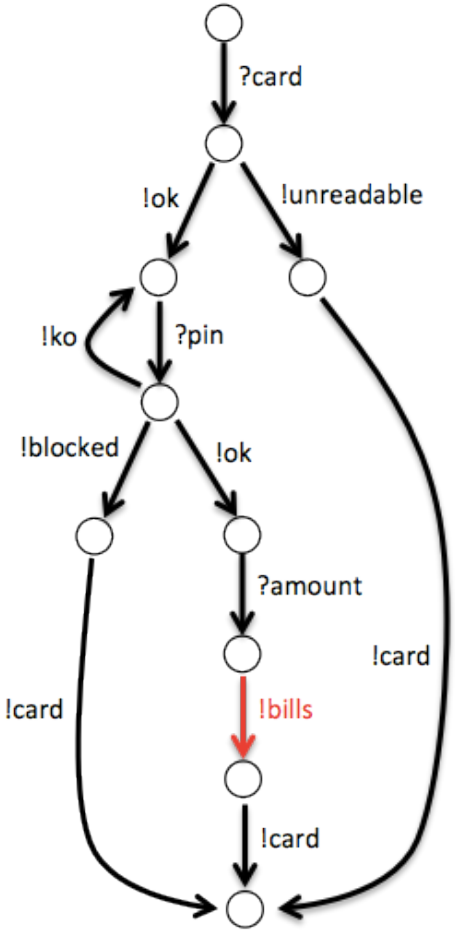
IOCO



Implementation of partial specification

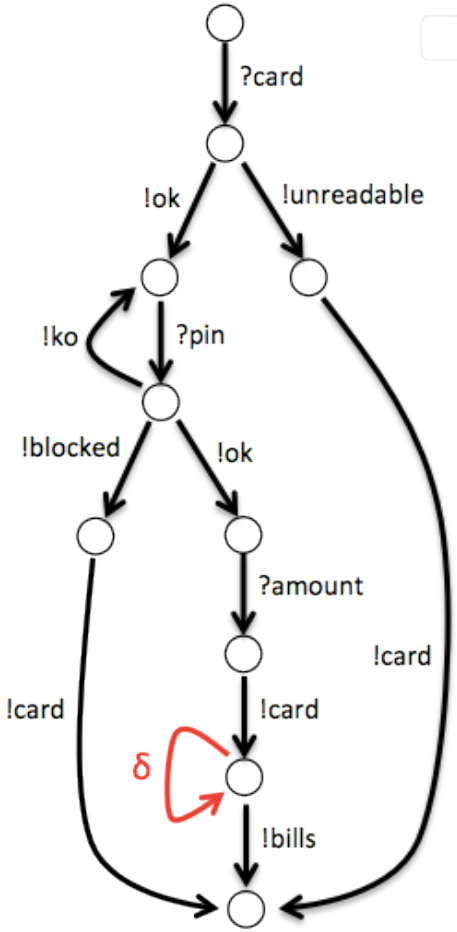
MBT challenges - Conformance relationship





Unexpected output

~~IOCO~~



Unexpected quiescence

Summary on Model-Based Testing

The fine art of modelling for testing…

Models are strongly related to the test objectives!

MBT **test models** should be:

- abstract:
  - to cover what is intended to be tested
- detailed and precise enough:
  - to compute the expected result (oracle) → test verdict assignment
- validated and verified[1]:
  - if the test « fails » what is wrong? the model or the SUT?
    (keep in mind that MBT is « back-to-back testing » - MBT model vs. SUT)

…a difficult trade-off!

[1]a must-read: M.-C. Gaudel. *Checking models, proving programs, testing systems*. Tests & Proofs'2011. LNCS.

Agenda

1. Dwyer's temporal property patterns
2. Principles of Model-Based Testing
3. Use of properties for testing
4. Your assignment for the next two weeks

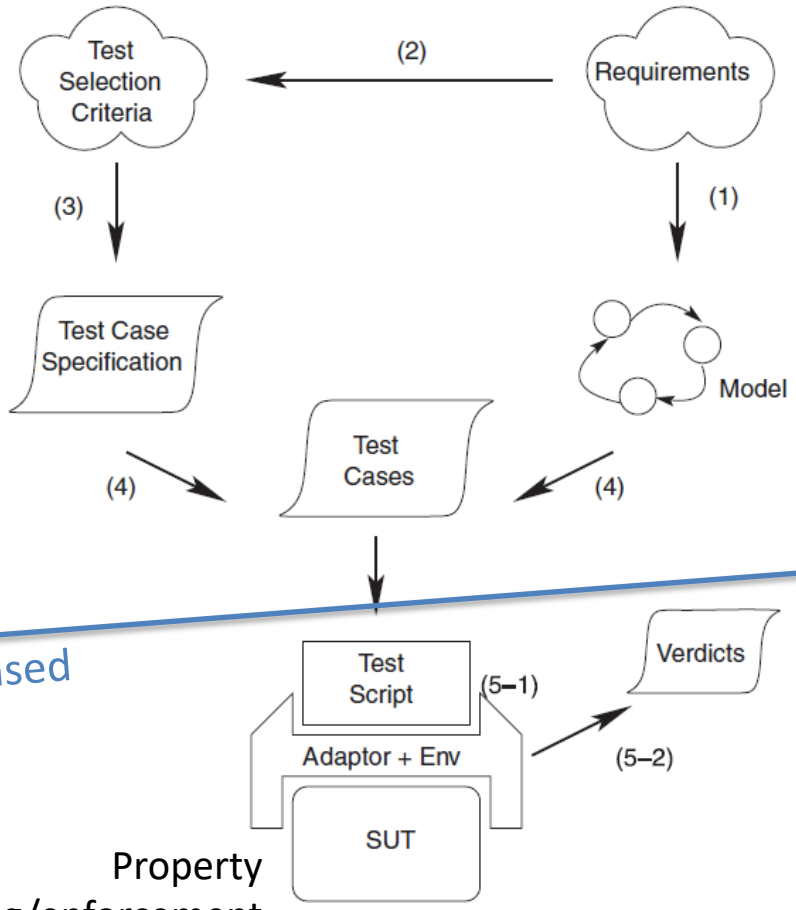
Property-based testing – where?

Property coverage (automaton)

Express a temporal (dynamic) requirement

Exploit property to drive the test gen. process

Verification & validation of the model



Model-based

Not necessarily model-based

Detect property violation → test failure

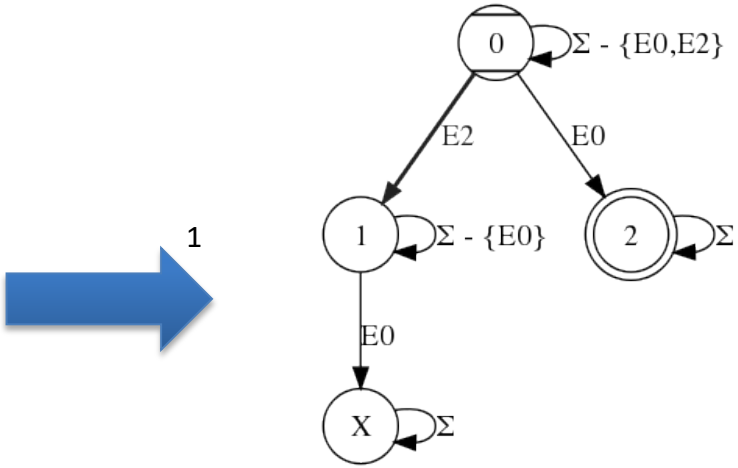
Property monitoring/enforcement

Property-based testing – How-to

Properties have to be written in a format that allows processing:

- measuring property coverage
- guide the test generation process
- monitoring the property to detect violation
- enforcing its satisfation

never BUY_TICKET_OK
before LOGIN_OK



[1]

Property automata have to be complete and (preferably) deterministic.

[1]Kalou Cabrera Castillos. **Génération automatique de tests à partir de propriétés temporelles et de modèles comportementaux.** Thèse de doctorat. Novembre 2013.

Property-based testing – Use of automata

Monitoring of the property on the model/system :

- composition of the model/system and the property automata
  - if both are expressed as automata
  - rarely the case in practice
- exploration of the property automaton
  - in parallel while the model/system is running, or
  - after the execution, on recorded logs/traces

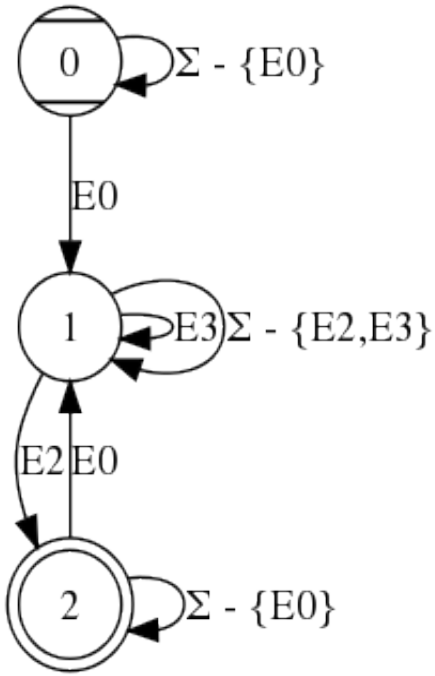
Monitoring vs. enforcing:

- monitoring consists in observing the execution of the system/model and reporting the violations of the property
- enforcing consists in preventing the violation of the property to happen (and also report violation attempts)

PBT – K. Cabrera PhD Thesis

- Existing automata coverage criteria are not appropriate

➔ all transitions are considered equally!



E0 = LOGIN_OK

E2 = LOGOUT_OK

E3 = BUY_TICKET_OK

Need to distinguish two different kinds of transition

⇒ α-transitions, labelled by events expressed in the property

⇒ Σ-transitions, the others

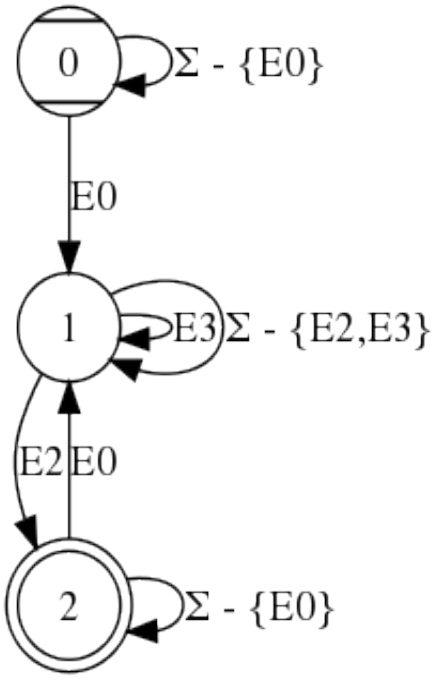
Also, the origin of all the transitions (scope/pattern) is known.

PBT – K. Cabrera PhD Thesis

- New coverage criteria for the property automata
  - alpha-transition coverage: coverage of the transitions labelled by events expressed in the property

Transitions to cover:

(0, E0, 1)
(1, E3, 1)
(1, E2, 2)
(2, E0, 1)



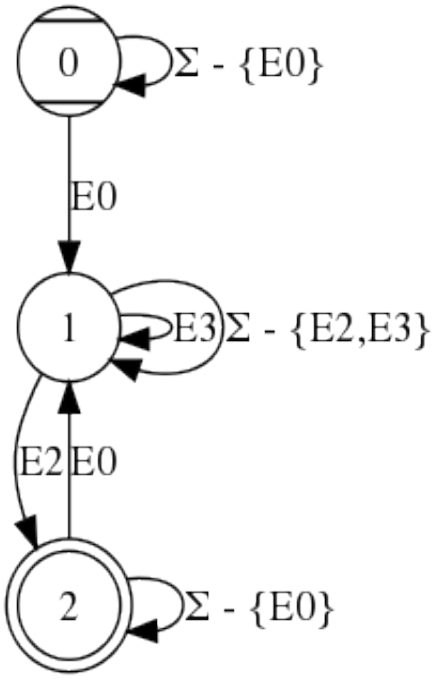
PBT – K. Cabrera PhD Thesis

- New coverage criteria for the property automata
  - alpha-transition-pairs coverage: coverage of the pairs of transitions labelled by events expressed in the property

Pairs of transitions to cover:

< (0, E0, 1) ; (1, E3, 1) >
< (1, E3, 1) ; (1, E2, 2) >
< (1, E2, 2) ; (2, E0, 1) >
< (2, E0, 1) ; (1, E3, 1) >
< (2, E0, 1) ; (1, E2, 2) >

Important: strict successions of α-transitions are not required (intermediate Σ-transitions are allowed)



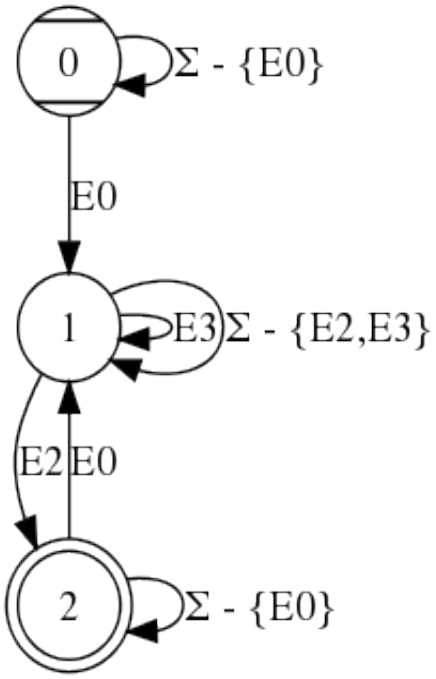
PBT – K. Cabrera PhD Thesis

- New coverage criteria for the property automata
  - k-pattern coverage: coverage of the iterations of the pattern

All pattern-loops have to iterated between 0 and k times.

Applicable to « repeatable » patterns:

- precedes
- follows
- eventually at least n times (if n >= k)



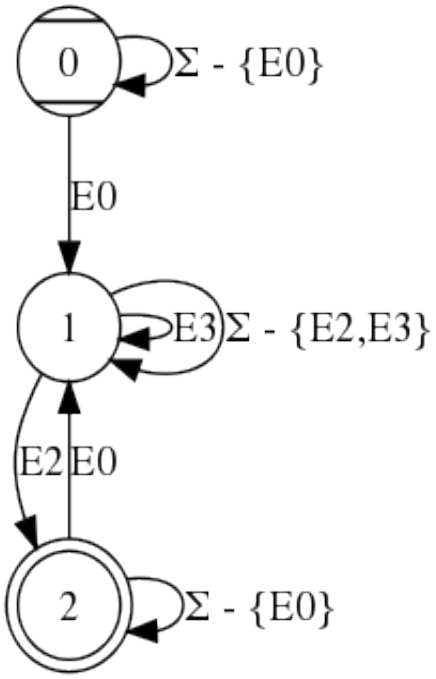
PBT – K. Cabrera PhD Thesis

- New coverage criteria for the property automata
  - k-scope coverage: coverage of the iterations of the scope

All scope-loops have to iterated between 1 and k times.

Applicable to « repeatable » scopes:

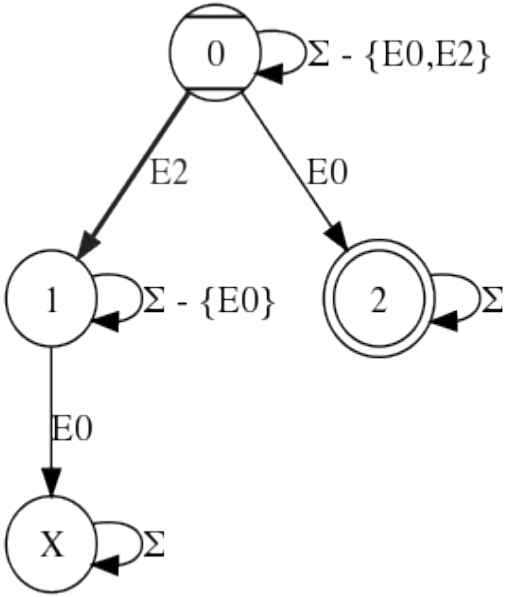
- between
- after… until…



Notice: interesting paths end on a final state of the automaton

PBT – K. Cabrera PhD Thesis

Case of transitions leading to the error state



never BUY_TICKET_OK
before LOGIN_OK

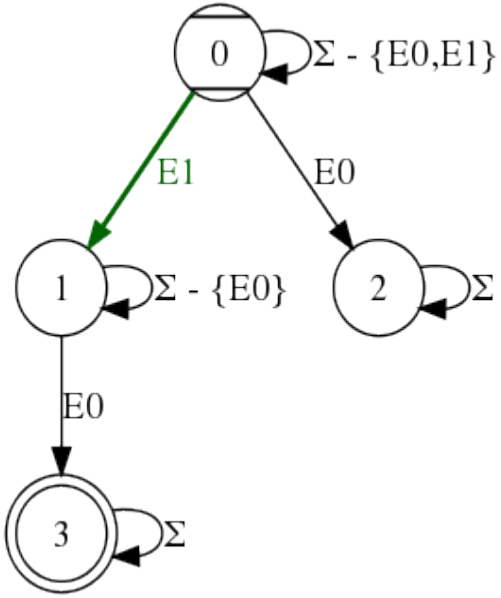
Can not be activated if we assume that the model satisfies the property (which is supposed to be the case)

New coverage criteria are inefficient…

➔ Specific criterion to test the robustness of the system w.r.t. the property

PBT – K. Cabrera PhD Thesis

Coverage criterion: robustness



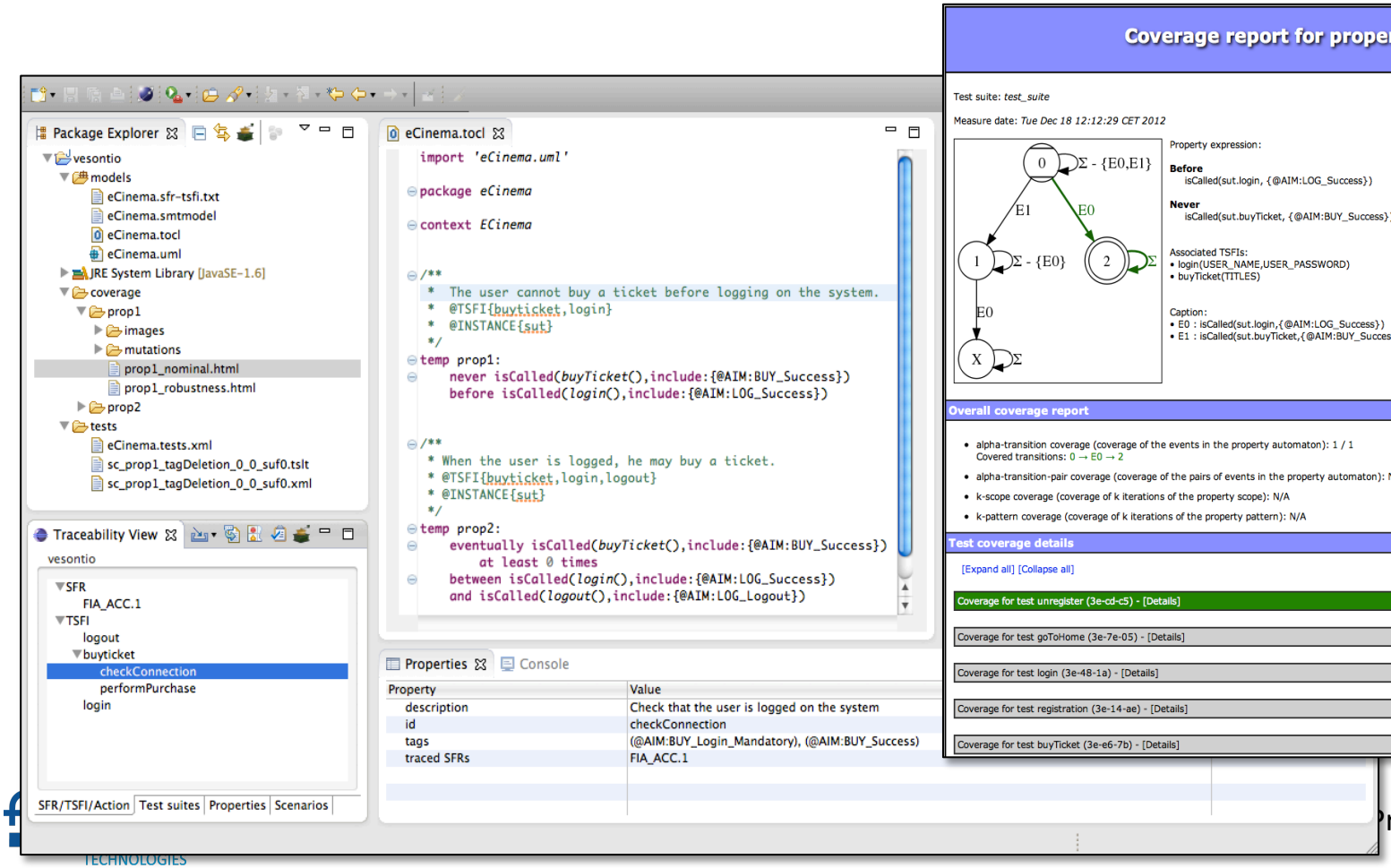
Modification of the automaton:
- the error state becomes the final state
- the event labelling the faulty transition is mutated/weakened to be made activable

Possible mutations:
- deletion/negation of predicates (pre/post)
- deletion/change of activated behavior

E1: BUY_TICKET_OK ➔ BUY_TICKET

PBT – K. Cabrera PhD Thesis

- Development of an Eclipse plug-in to support the approach



Agenda

1. Dwyer's temporal property patterns
2. Principles of Model-Based Testing
3. Use of properties for testing
1. Your assignment for the next two weeks
   - tutorial: write a model of an example application + property patterns
   - practical sessions: implement a monitor for property patterns, use it for model testing and model-based testing

Your assignment for the next two weeks – case study

Automated Teller Machine (ATM) - withdraw cash with a credit card

- System Under Test (SUT) = cash machine
- Test data = credit cards with associated bank accounts
- Control points = reader, pad (0-9 + cancel, delete, validate)
  → abstracted into « actions » (insert card, type PIN, etc.)
- Observation points = messages on the screen, card/bills ejected

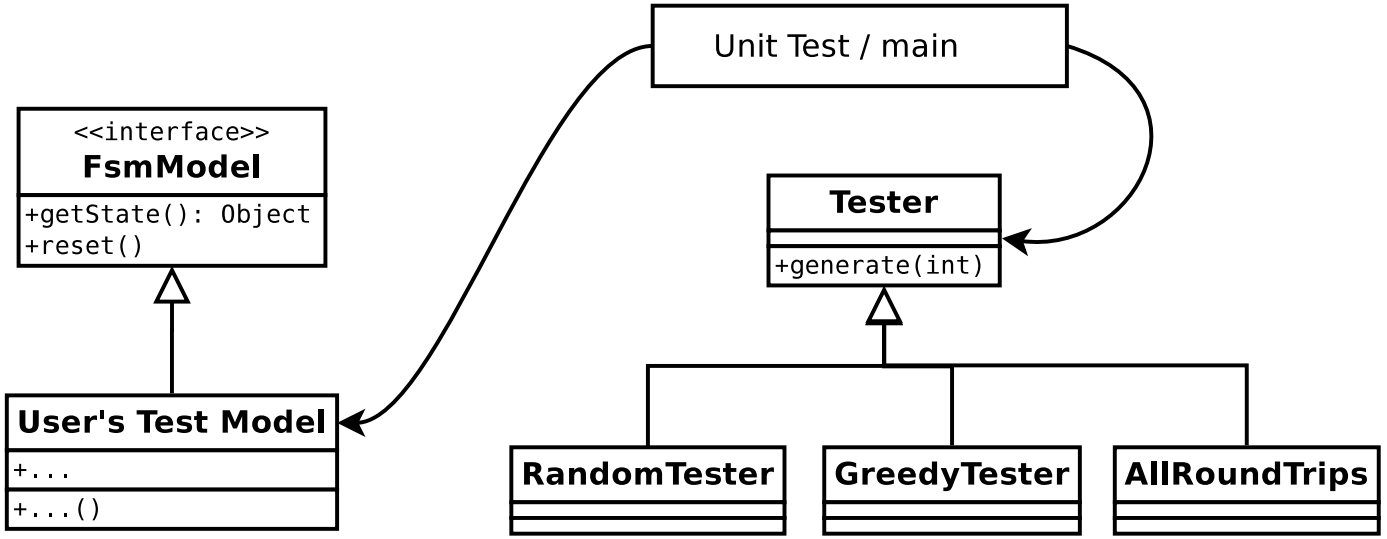
- Behaviours = « usual » behaviour of an ATM (functional testing)

A specification of the ATM will be given, along with Java binaries of the application.

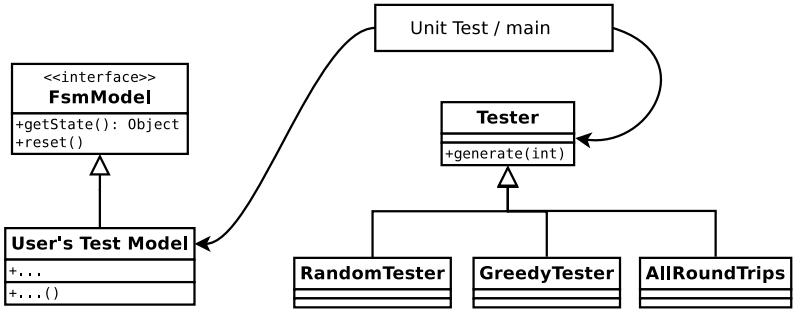
Your assignment for the next two weeks – supporting tools

Use of ModelJUnit - https://sourceforge.net/projects/modeljunit/

Library to perform model-based tests, by defining a Java class of a user model.



Your assignment for the next two weeks – supporting tools



Abstract !

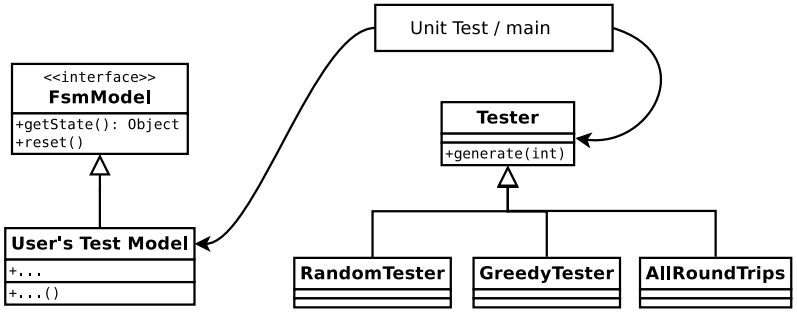
Write a model as a Java class that implements the two methods:

- getState() : Object → a method used to identify the current state of the FSM
- reset() → a method that (re-)initializes the automaton's variables (if any)

and, of course,

- additional methods that represent the transitions of the automaton, which have to be annotated by @Action (used by the tool to identify the FSM's transitions)
- possibly supporting methods if necessary

Your assignment for the next two weeks – supporting tools



It is possible to ask the tool to compute test cases and measure the states/transitions of the FSM covered by the tests

Apply the testers:

- RandomTester: tests a system by making random walks through the EFSM model.
- GreedyTester: improves the RandomTester by prioritizing, during the exploration, the transitions that have not been considered before
- AllRoundTrips: GreedyTester that stops after having visited N (parameter) times a state

Don’t worry: a code skeleton will be provided next week.

Finally, it is possible to link the test generator to the system under test → online testing

Your assignment for the next two weeks

What is expected of you today:

- work in groups of 3, with collaborative design/development tools

1. write a (Extended) Finite State Machine of the case study
2. write a set of temporal properties to express the requirements using temporal patterns (max. 3 properties/group – each group gets different properties)
3. translate these properties into their corresponding automata

For the next week (next Wednesday):

- develop or get a small automata library in Java (manage states, transitions, explore)

Your assignment for the next two weeks

What will be expected of you next week:

- use a simple Model-Based Testing tool (ModelJUnit)
  - implement your FSM (realized last week) in the tool
  - generate tests
  - execute tests on the implementation
- monitor the properties on the model by running the automaton in parallel and measure the property automata coverage
- monitor the properties on the implementation (e.g. using a proxy design pattern)

→ To be returned by the end of January.

Your assignment for the next two weeks

Summary: you have to test this ATM application regarding these properties

- formalize the properties
- generate and run tests and measure the coverage of the property (you are not expected to write a dedicated test generation algorithm – use the existing ones)
- make sure that your properties have been correctly formalized by monitoring their satisfaction at the model level.
- Produce some Java code and a small report that provides the considered properties and their expression as an automaton

Before we start...

Questions?