j-POST: a Java Toolchain for Property-Oriented Software Testing

Model-Based Testing / ETAPS'08

Yliès Falcone Laurent Mounier Jean-Claude Fernandez
Jean-Luc Richier

Vérimag & LIG, Universities of Grenoble

March 30 2008, Budapest, Hungary

Testing a policy ?

Notion of policy :

- Expected behavior of nowadays systems
- Specific domains, *e.g.* security

Issues when testing a policy :

- Availability and faithfulness of the model
- Discrepancy between the policy and the system

Testing a policy ?

Notion of policy :

- Expected behavior of nowadays systems
- Specific domains, *e.g.* security

Issues when testing a policy :

- Availability and faithfulness of the model
- Discrepancy between the policy and the system

Proposed Approach

- No functional specification of the system
- policy = set of requirements expressed by **logical formula**
- **property-oriented** test generation based on the structure of each formula φ

Running example : a travel agency application

The Travel application

- Management of mission of employees in an organization
- Client/Server principle
- Users request to perform operations on the application

Running example : a travel agency application

The Travel application

- Management of mission of employees in an organization
- Client/Server principle
- Users request to perform operations on the application

A security policy for Travel

"It is impossible to create a mission in Travel before being connected"
Formalization :

- elementary operation
- use of a logical formalism

$$(\neg \mathit{mission\_creation}(\mathit{user}, \mathit{mission}))\ \mathcal{U}\ \mathit{connection}(\mathit{user})$$

Outline

Test Generation : general principle

Aimed to produce a test case relatively to a logical formula

About the test case

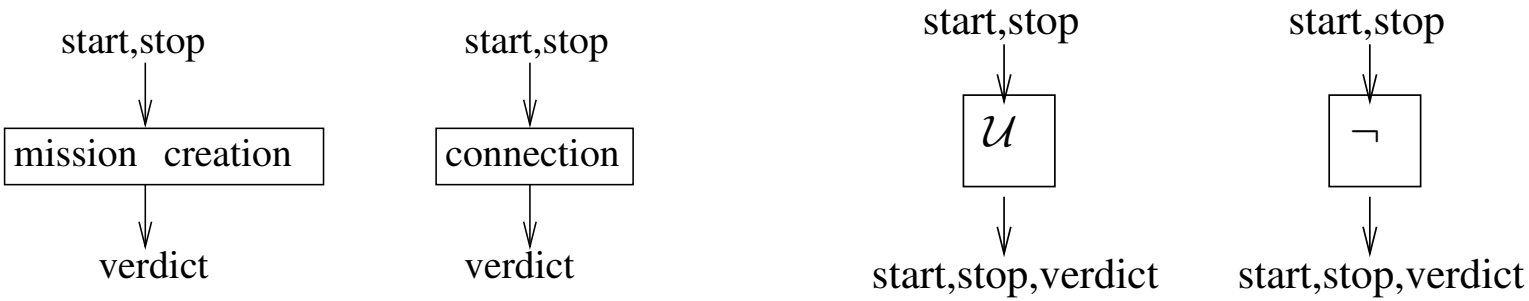
A set of communicating test processes (typed process algebra)

- test modules : evaluation of elementary predicates
  e.g. : to connect, perform an operation
- test controllers : associated to operators of the formal requirement

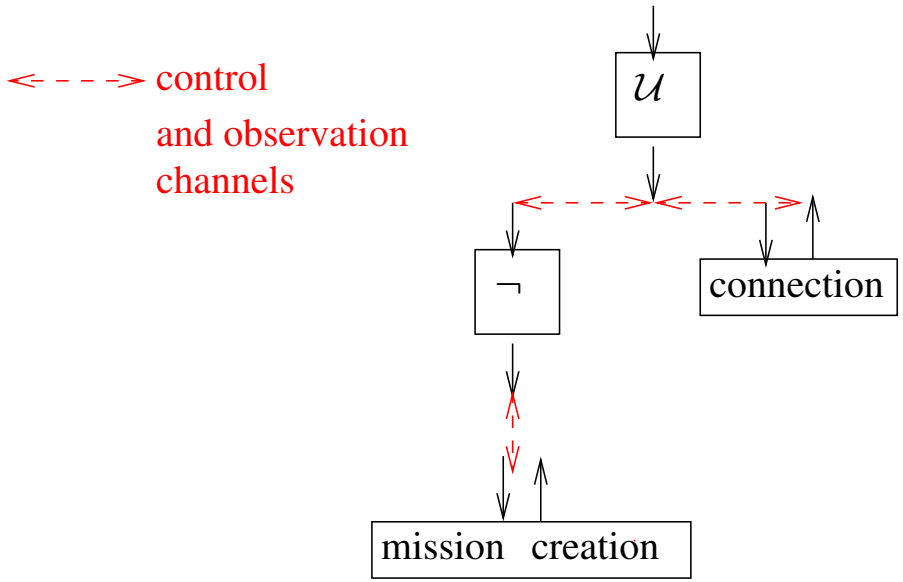
$\rightarrow$ Defined compositionaly following the syntax of the formula

$\varphi = (\neg mission\_creation())\ \mathcal{U}\ connection()$

Inputs : test modules and controllers :



Test case :



Compositional test generation

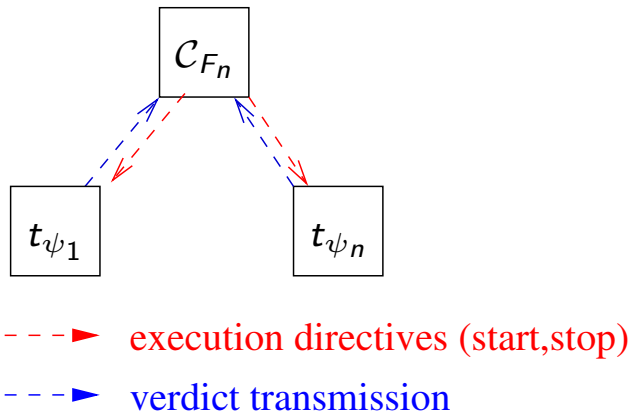
Test generation function

Produce $t_\varphi = GenTest(\varphi)$

- test module t_{p_i} associated to each predicate p_i
- each $\psi = F^n(\psi_1, \cdots, \psi_n) \longrightarrow$ test process t_ψ
  $\hookrightarrow t_\psi$: parallel composition of t_{ψ_1}, ..., t_{ψ_n} and $\mathbb{C}_{F^n}$ called a test controller for the operator F^n.

Roles of the controllers :

- execution management for sub-tests
- combination of verdicts



Principle of test execution

Produce a verdict for the initial requirement

$\hookrightarrow$ Parallel execution of the test case and the SUT with synchronisation on common action

Principle of test execution

Produce a verdict for the initial requirement
$\hookrightarrow$ Parallel execution of the test case and the SUT with synchronisation on common action

Issues for execution :

- Generated test case contains lots of possible executions
  - Non-determinism inside test modules
  - Parallel execution of test modules
- Information in the test case comes (only) from the formula (no additional behavior)
  e.g. : insert a disconnection after being connected

Principle of test execution

Produce a verdict for the initial requirement

↪ Parallel execution of the test case and the SUT with synchronisation on common action

Issues for execution :

- Generated test case contains lots of possible executions
  - Non-determinism inside test modules
  - Parallel execution of test modules
- Information in the test case comes (only) from the formula (no additional behavior)
  e.g. : insert a disconnection after being connected

↪ Problem of test selection

↪ A classical solution : use a test objective during test generation (e.g. TGV)

Principle of test execution

Produce a verdict for the initial requirement

↪ Parallel execution of the test case and the SUT with synchronisation on common action

Issues for execution :

- Generated test case contains lots of possible executions
  - Non-determinism inside test modules
  - Parallel execution of test modules
- Information in the test case comes (only) from the formula (no additional behavior)
  e.g. : insert a disconnection after being connected

↪ Problem of test selection

↪ A classical solution : use a test objective during test generation (e.g. TGV)

Not applicable directly : no complete specification

Test Objectives (1) : definition

The test objective guides the test execution

- Features : local priorities and inserable behavior
- Reduce choices between actions

Test Objectives (1) : definition

The test objective guides the test execution

- Features : local priorities and inserable behavior
- Reduce choices between actions

Definition (Behavioural test objective)

A test objective O relatively to a test case t which semantics can be expressed by a LTS $(Q^{S_t}, Act^{S_t}, \rightarrow_{S_t}, q_0^{S_t})$ is :

- a deterministic LTS $(Q^O, Act^O, \rightarrow_O, q_0^O)$
- complete wrt. Act^{S_t} ($i.e.$ $\forall q \in Q^{S_t}, \forall a \in Act^{S_t}, \exists q' \in Q^O \cdot q \xrightarrow{a}_O q'$)
- Q^O contains two sink states $Accept^O$ and $Reject^O$
- $Act^O \subseteq Act^{S_t} \cup Act^{com}$

Test Objectives (2) : test selection

Priorities for actions :

- For $a \in Act_O$, $prio(a)$
- $\forall E \subseteq Act^* \cdot$
  $prioritary(E) = \{a \in E \mid prio(a) = max\{prio(act) \mid act \in E\}\}$

Test Objectives (2) : test selection

Priorities for actions :

- For $a \in Act_O$, $prio(a)$
- $\forall E \subseteq Act^* \cdot$
  $prioritary(E) = \{a \in E \mid prio(a) = max\{prio(act) \mid act \in E\}\}$

Test selection via a test objective

For $a \in Act \cdot prioritary(O(o)) = \{a\}$,

$$\frac{a \in T(t) \qquad t \xrightarrow{a}_{S_t} t' \qquad o \xrightarrow{a}_{O} o'}{(t,o) \xrightarrow{a}_{S_t \times O} (t',o')}\ 1$$

$$\frac{a \notin T(t) \qquad inserable(a) \qquad o \xrightarrow{a}_{O} o'}{(t,o) \xrightarrow{a}_{S_t \times O} (t,o')}\ 2$$

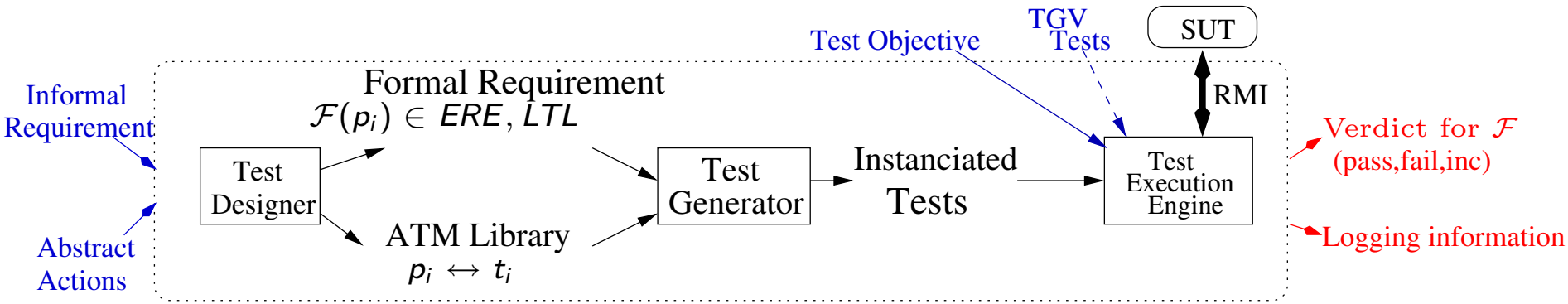
$$\frac{o \xrightarrow{a}_{O} Reject_O}{(t,o) \xrightarrow{a}_{S_t \times O} Inc}\ 3$$

Outline

Overview

What is it ?

A toolchain dedicated to Property-Oriented Software Testing.



Features

Several phases of testing are addressed :

- design (manual)
- generation (automatic)
- execution (automatic)

Features

Several phases of testing are addressed :

- design (manual)
- generation (automatic)
- execution (automatic)

Support for several logics :

- Extended Regular Expressions
- Linear Temporal Logic

Features

Several phases of testing are addressed :

- design (manual)
- generation (automatic)
- execution (automatic)

Support for several logics :

- Extended Regular Expressions
- Linear Temporal Logic

Using open formats for internal representations (tests modules, test controllers) :

- interaction with other tools (e.g. : TGV)
- logical formalism (plugin)

Outline

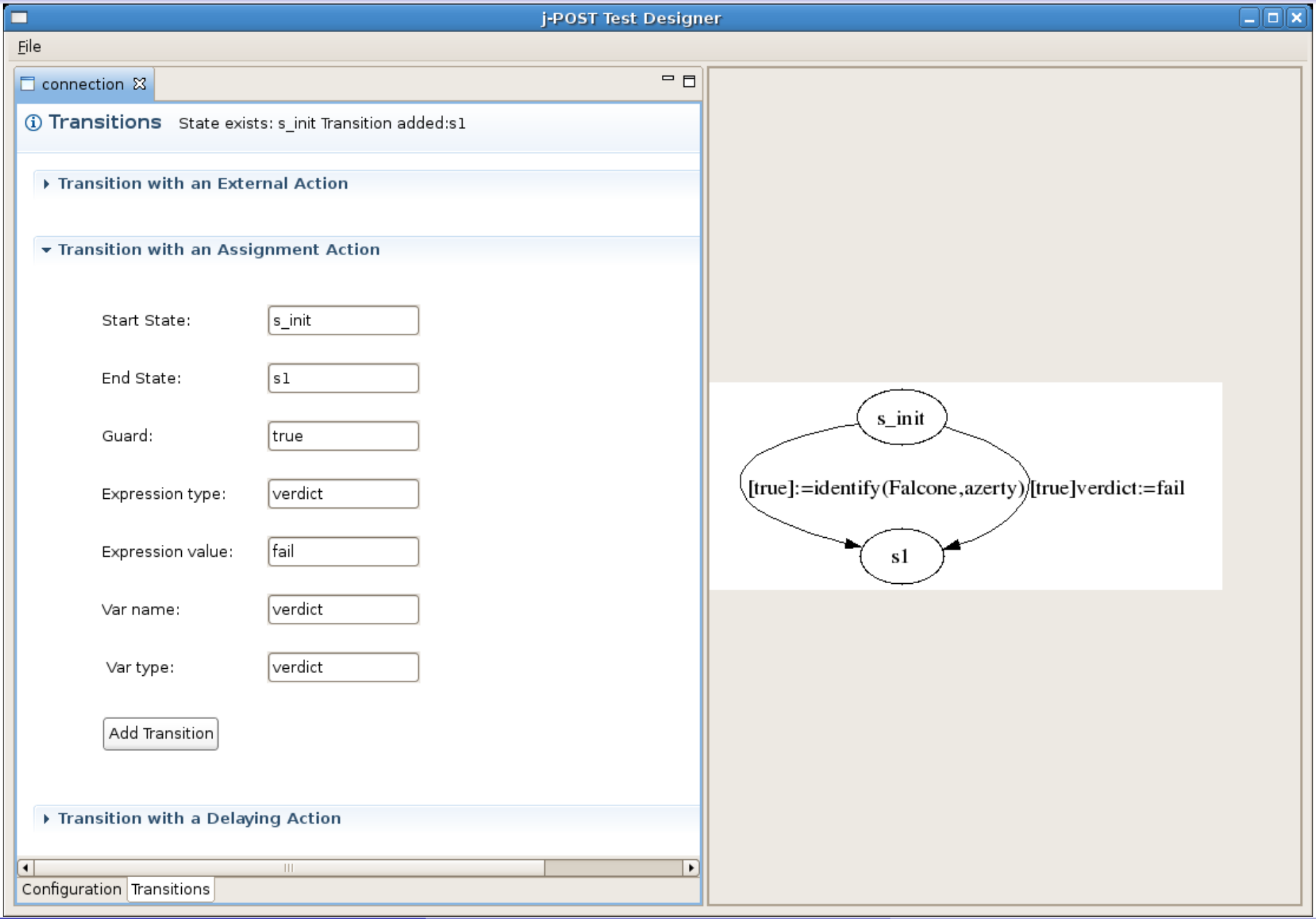
Functionalities of the test designer

About abstract test modules

- extended LTS
- Actions : internal, communication (internal to the tester), with the SUT
- A typed process algebra as an underlying formalism

Establishment of a test module library

- Edition of test modules
  - XML
  - GUI (based on Eclipse RCP)
- Visualization of test modules
  - Generation of a graph
  - GraphViz


j-POST Test Designer
File
connection
Transitions
State exists: s_init Transition added:s1
Transition with an External Action
Transition with an Assignment Action
Start State:
s_init
End State:
s1
Guard:
true
Expression type:
verdict
Expression value:
fail
Var name:
verdict
Var type:
verdict
Add Transition
Transition with a Delaying Action
Configuration
Transitions
s_init
[true]:=identify(Falcone,azerty)/[true]verdict:=fail
s1

Outline

Functionalities of the test generator

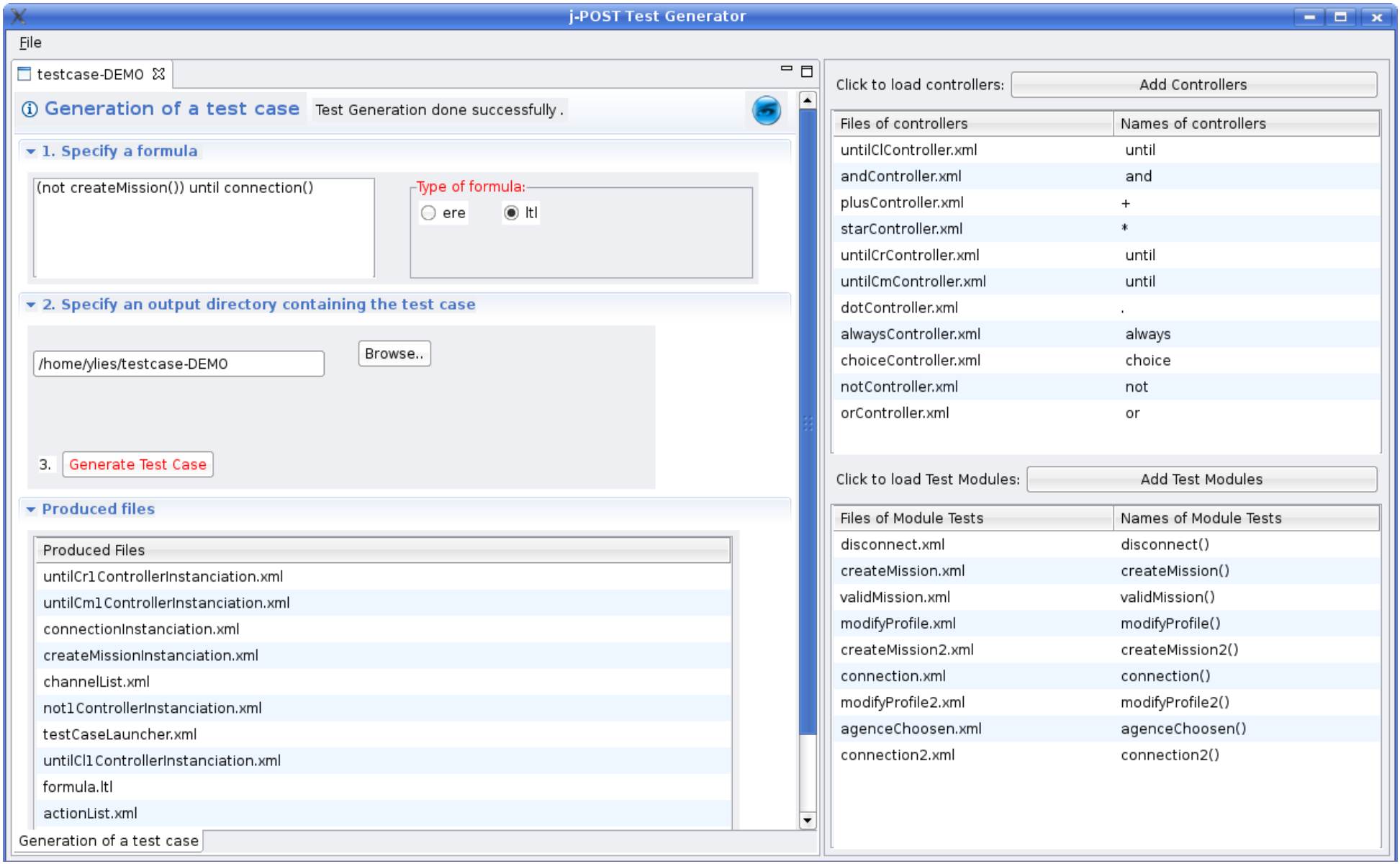
Implements the *GenTest* function

- Construction of a set of communicating tests
- Instantiation of test modules and generic controllers with information from the formula (e.g. value of parameters)

Supported formalisms :

- (Future) Linear Temporal Logic
- Extended Regular Expressions

Extendable architecture via logic plugins


j-POST Test Generator
File
testcase-DEMO
Generation of a test case
Test Generation done successfully .
1. Specify a formula
(not createMission()) until connection()
Type of formula:
ere
ltl
2. Specify an output directory containing the test case
/home/ylies/testcase-DEMO
Browse..
3.
Generate Test Case
Produced files
Produced Files
untilCr1ControllerInstanciation.xml
untilCm1ControllerInstanciation.xml
connectionInstanciation.xml
createMissionInstanciation.xml
channelList.xml
not1ControllerInstanciation.xml
testCaseLauncher.xml
untilCl1ControllerInstanciation.xml
formula.ltl
actionList.xml
Generation of a test case
Click to load controllers:
Add Controllers
Files of controllers
Names of controllers
untilClController.xml
until
andController.xml
and
plusController.xml
+
starController.xml
*
untilCrController.xml
until
untilCmController.xml
until
dotController.xml
.
alwaysController.xml
always
choiceController.xml
choice
notController.xml
not
orController.xml
or
Click to load Test Modules:
Add Test Modules
Files of Module Tests
Names of Module Tests
disconnect.xml
disconnect()
createMission.xml
createMission()
validMission.xml
validMission()
modifyProfile.xml
modifyProfile()
createMission2.xml
createMission2()
connection.xml
connection()
modifyProfile2.xml
modifyProfile2()
agenceChoosen.xml
agenceChoosen()
connection2.xml
connection2()

Outline

Functionalities of the test execution engine

Goal : produce a verdict for the initial requirement

Functionalities of the test execution engine

Goal : produce a verdict for the initial requirement

Multithreaded execution

- Java Thread / test process
- Scheduling policy (test objective)
- Priority order between actions

Functionalities of the test execution engine

Goal : produce a verdict for the initial requirement

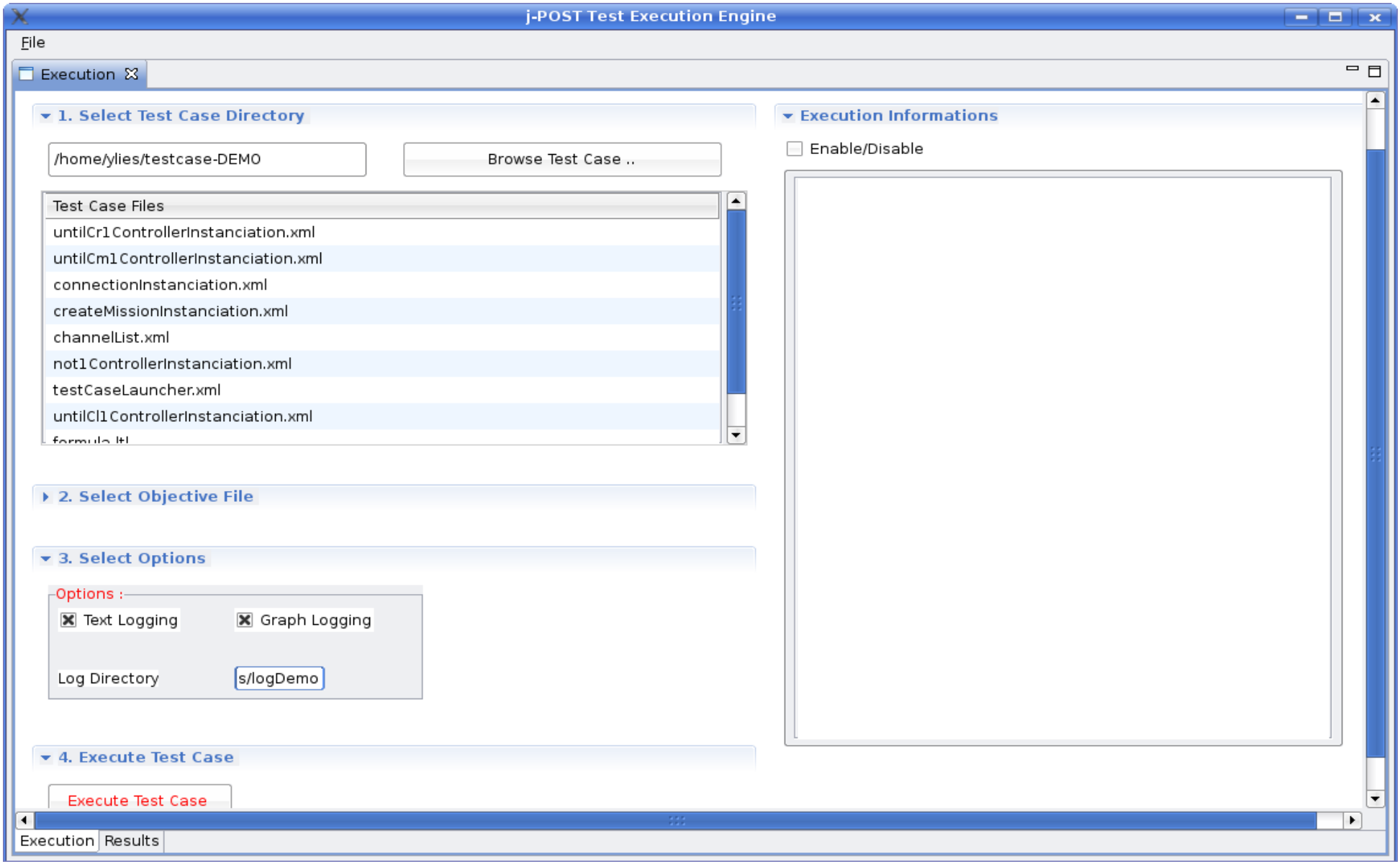
Multithreaded execution

- Java Thread / test process
- Scheduling policy (test objective)
- Priority order between actions

"on the fly" concretisation

Mapping between external abstract actions and the actions on the SUT

- Remote Method Invocation
- Web service invocation
- ...


j-POST Test Execution Engine
File
Execution
1. Select Test Case Directory
/home/ylies/testcase-DEMO
Browse Test Case ..
Test Case Files
untilCr1ControllerInstanciation.xml
untilCm1ControllerInstanciation.xml
connectionInstanciation.xml
createMissionInstanciation.xml
channelList.xml
not1ControllerInstanciation.xml
testCaseLauncher.xml
untilCl1ControllerInstanciation.xml
2. Select Objective File
3. Select Options
Options :
Text Logging
Graph Logging
Log Directory
s/logDemo
4. Execute Test Case
Execute Test Case
Execution Informations
Enable/Disable
Execution
Results

Outline

Conclusion

A Java toolchain dedicated to property testing

Extendable architecture

Proposed approach

- algebraic composition
- property-driven
- suitable for security policy testing

References

j-POST Web page : http ://www-verimag.imag.fr/~async/jpost.html

Theory

- "A Test Calculus Framework Applied to Network Security Policies". In FATES/RV'06 : Formal Approaches to TESting/Runtime Verification
- "A Compositional Testing Framework Driven by Partial Specifications". In TESTCOM/FATES'07 : TESTing of COMmunicating Systems/Formal Approaches to TESting
- "A Partial-specification Driven Compositional Testing Method". Vérimag Technical Report [TR-2007-4]

Implementation

- j-POST : a Java Toolchain for Property-Oriented Software Testing. Vérimag Technical Report [TR-2007-7]
- Experiments with j-POST on a Travel Agency Application

Perspectives

Assessment of quality for produced tests

- coverage criteria
- mutation

Perspectives

Assessment of quality for produced tests

- coverage criteria
- mutation

Using Abstraction/Concretisation in test modules

- Abstract types [Lestiennes,Gaudel'02]
  - Domain : finite set of values
  - Operations on abstract data types need corresponding concrete operations
- Issues :
  - Test execution can modify concrete domains
  - Coverage of concrete wrt. abstract domains