s@movar
CNRS Institut TELECOM

Network Protocol Testing:

Formal, Active and Passive!

Stephane Maag

ADVCOMP 2014

August 24 - 28, 2014 - Rome, Italy

Introduction

- **What is TESTING ??!**
  - It is Not:

    "I've searched hard for defects in this program, found a lot of them and repaired them. I can't find any more, so I'm confident there aren't any. " (Hamlet, 1994)

  - It is :

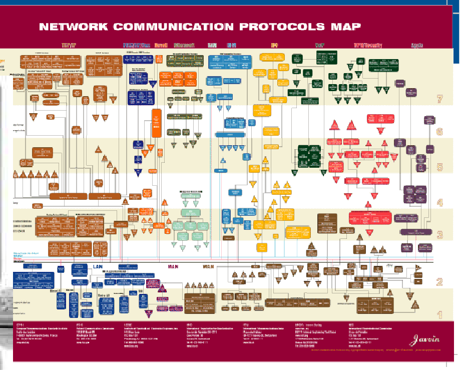
    "A process of analyzing an item to detect the differences between existing and required conditions, and to evaluate the features of the item." (ANSI/IEEE Standard 1059)

- **To summarize** (briefly!):
  - Meets the requirements,
  - Works as expected,
  - Is implemented with the same characteristics.

➔ Many stages or phases are needed …

What we want to Test??

- ITU, ITU-T (*ITU-T X.224,…*)
- ISO (*ISO 7498,…*)
- IEEE (*IEEE 802.11,…)*
- ETSI (ISDN, SS7)
- IETF (*routing protocol,…*)
- …



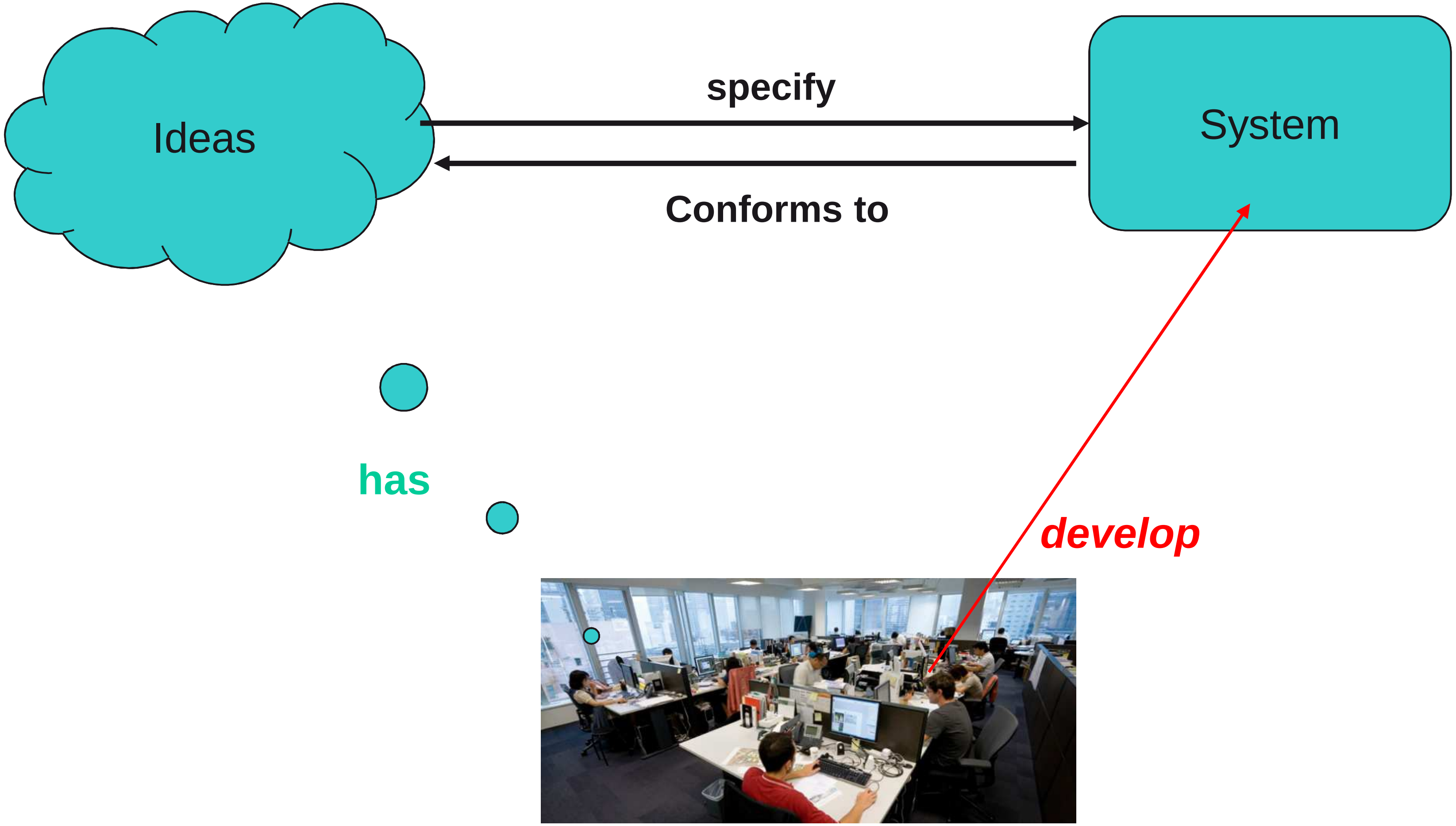
Our targets:

- Protocols
- Network devices (and its embedded components)
- Communicating systems (clouds, WS, IMS based,…)
- Information Systems
- Etc.

- But, in our presentation: **not** the Software or programs!

➔ we focus on **black box testing.**

Why to test??



Ideas

specify

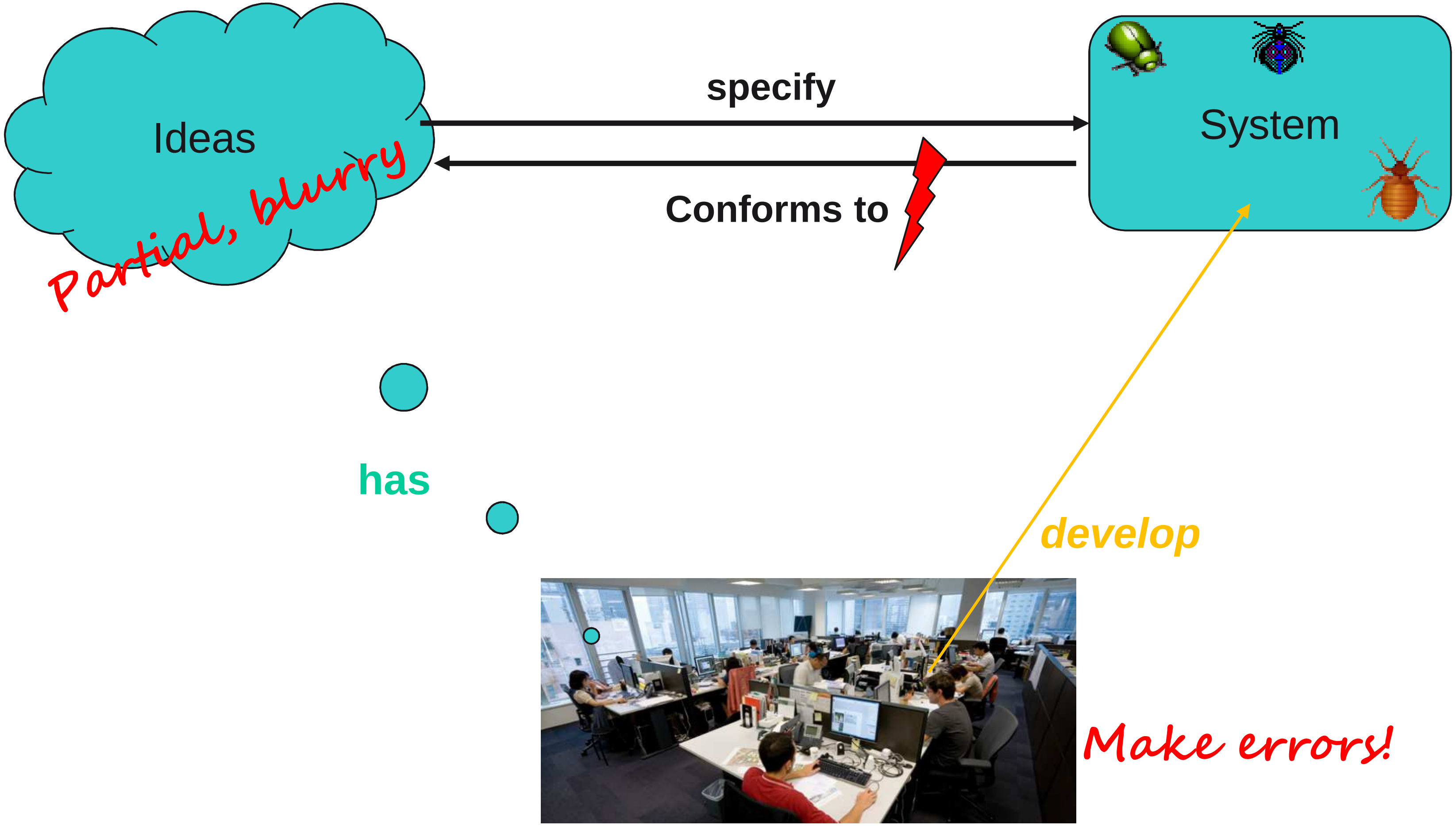
Conforms to

System

has

develop

From the *Ideas* to the *System*



Ideas

Partial, blurry

specify

Conforms to

System

has

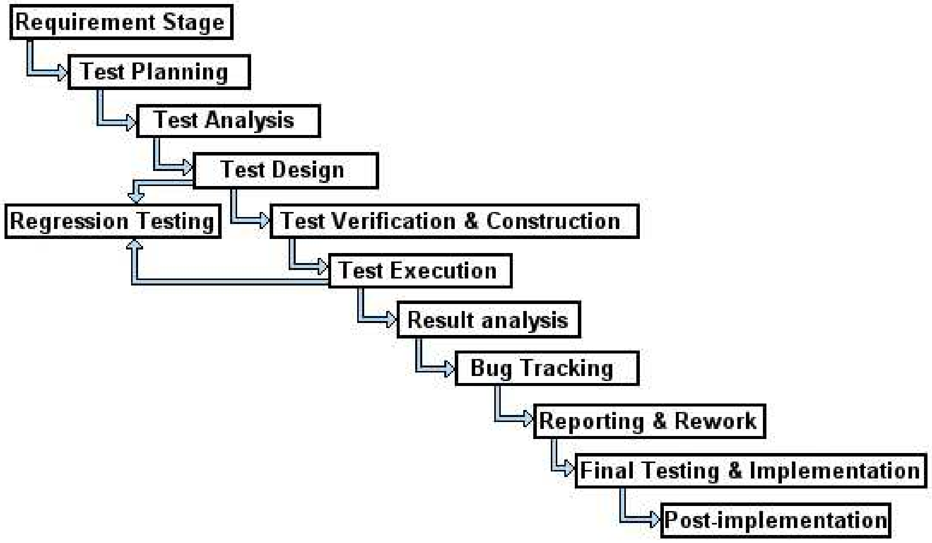
develop

Make errors!

Why to test??

- Depending on “What” and “When” we test, the reasons for testing are numerous!

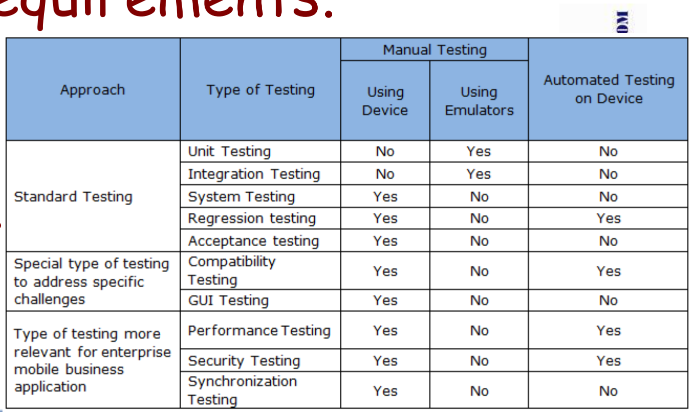
- This could be for:
  - Checking the design
  - Bug tracking
  - Conformance
  - “Security”
  - Interoperability
  - Performance
  - Reporting
  - Trust, confidence
  - … thousands of reasons …
  - Making money?



How to test??

- Depending on "What", "Why" and "When" we test, hundreds of techniques are today used! [Hierons et al 2009]

- In our presentation, we introduce **formal** ways of testing network protocols through **active** and **passive** methods for **functional** requirements.
  - Formal description of the protocol requirements.
  - Active testing architectures.
  - Passive testing by formal monitoring.



Approach	Type of Testing	Manual Testing		Automated Testing on Device
		Using Device	Using Emulators	
Standard Testing	Unit Testing	No	Yes	No
	Integration Testing	No	Yes	No
	System Testing	Yes	No	No
	Regression testing	Yes	No	Yes
	Acceptance testing	Yes	No	No
Special type of testing to address specific challenges	Compatibility Testing	Yes	No	Yes
	GUI Testing	Yes	No	No
Type of testing more relevant for enterprise mobile business application	Performance Testing	Yes	No	Yes
	Security Testing	Yes	No	Yes
	Synchronization Testing	Yes	No	No

“Formal”, i.e.??

Did you already test your owns?!

1. You develop your system,
2. You check some well chosen functionalities,
3. If all pass, you then decide it is tested.

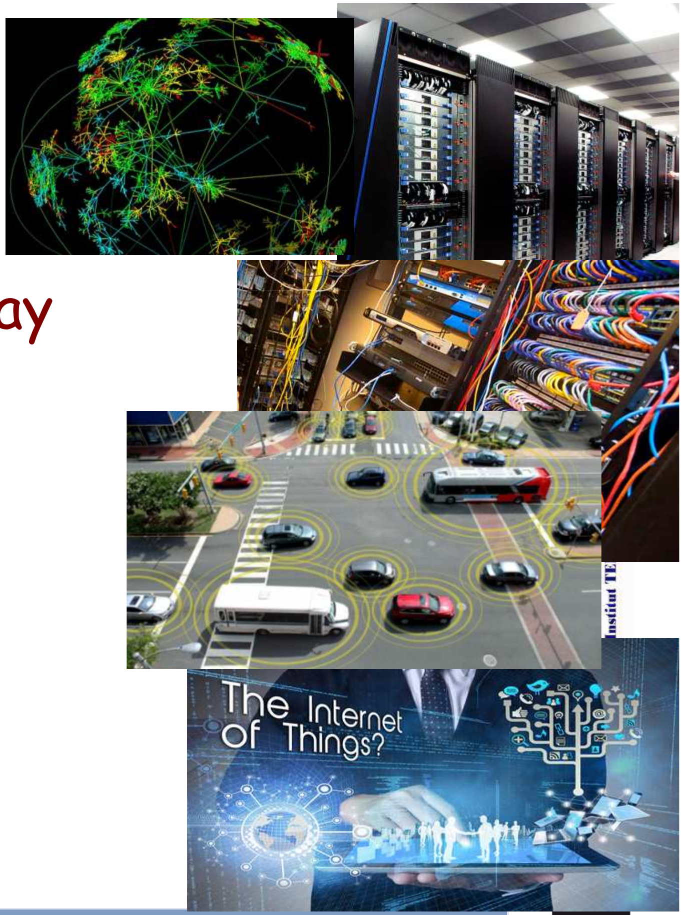
➔ Ancestral way !!

➔ MC Gaudel, ***Testing Can Be Formal, Too***

Proceedings of the 6th International Joint Conference CAAP/FASE on Theory and Practice of Software Development, Pages 82-96, Springer-Verlag, 1995

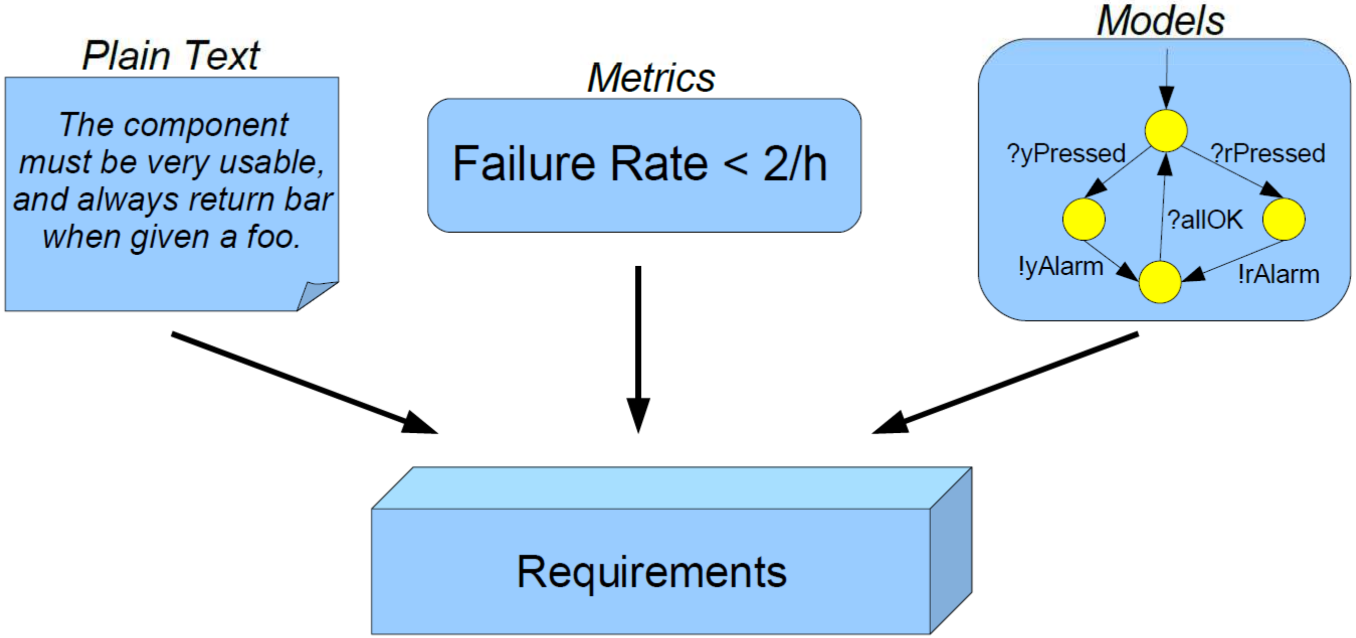
“Formal”, i.e.??

- Indeed, with current complex systems, “manual” testing is not “efficient”!
- Automate the Testing process becomes crucial.
  - Designing the protocol in a “smart” way
  - Designing the Ideas in a “clear” way
  - Defining Testing architectures
  - Test suites generation/execution
  - Testing verdicts generation
  - Diagnosis, reactions,
  - …

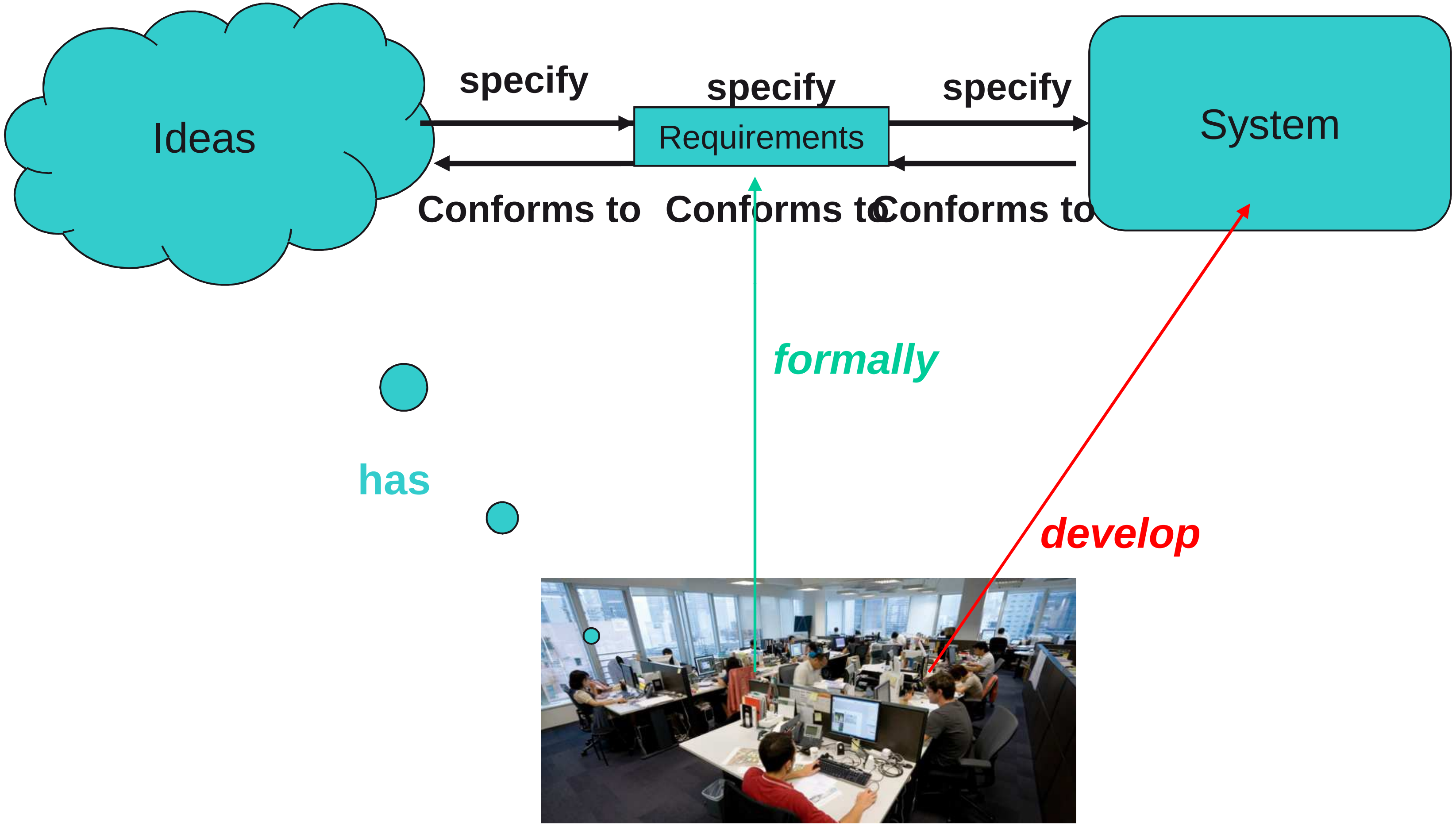


“Formal”, i.e.??

- The Ideas of a system is denoted in *Requirements.*
- To give the requirements of a system, metrics are not enough, further documents are needed.



Ideas and System - Revisited



Ideas

specify

specify

Requirements

specify

System

Conforms to

Conforms to

Conforms to

formally

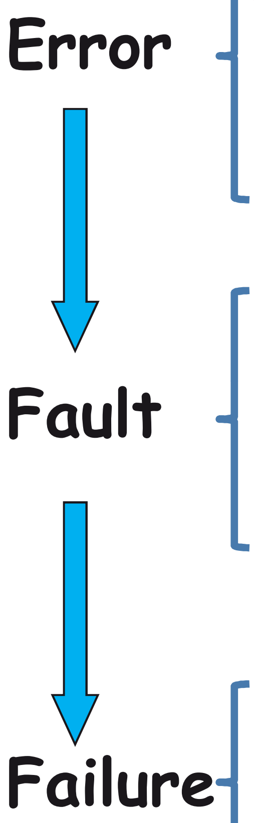
has

develop

Terminology

- **Error** means that a human being writes code which has a **Fault**.
- A **Fault** is a piece of code which, when being executed, may lead to a **Failure**.
- **Failure** is an observable behavior of a system which does not conform to a **requirement**.
- **Testing** means running a system with the ability to detect failures.

To improve the quality



Error

Organize, assess and improve the development process:
V-Model, XP, Agile ...
TQM, Six Sigma, ...
CMM, SPICE, ...

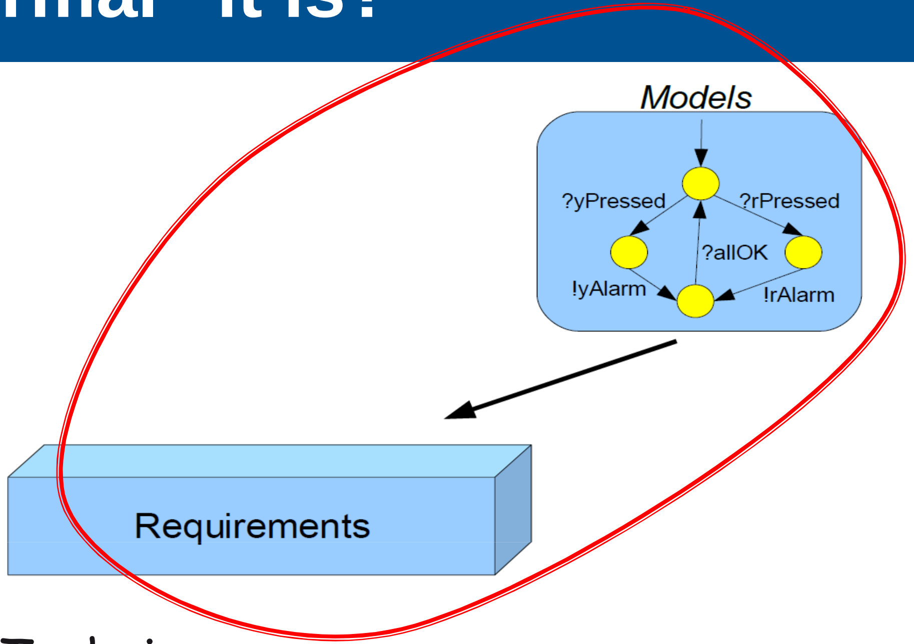
Fault

Static/Dynamic analysis
SW testing
Reviews, reporting,…

Failure

Organize, assess and improve the testing process:
Certified Tester, TMM, TPI, CTP, STEP,…

How “Formal” it is?



- <u>Formal Description Techniques</u>
  - Based on mathematical concepts, graph theory, logics or algebra.
  - To specify the functional properties (qualitiative) of a system according to its environment.
  - Are conceived to describe composed distributed systems.

  - **Many semantics**: state machines, LTS, temporal logic, process algebra (CCS), Petri nets, ...
  - **Many languages**: SDL, VHDL, Lotos, CASL, B, Z, LTL, … *UML, SysML?*

Advantages of these FDT

- Standardized and stable definitions (international consensus),
- No ambiguities
- Precise
- Controlled evolution,
- Scalable, application to complex realtime systems,
- Important user community,
- Reduce the development cost:
  - Fast error fixing – to react asap!
- Abstraction:
  - Implementation independent
    - ❑ FDT $\neq$ programming language

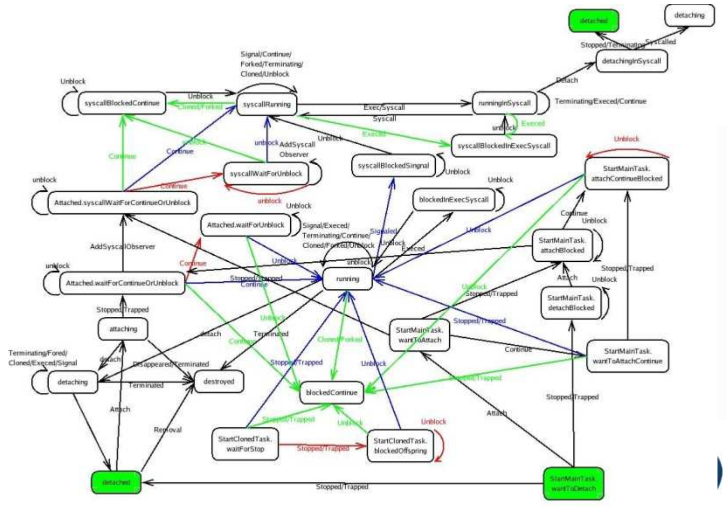
$$s_0 = \perp$$

$$s_1 = \beta \vee (\alpha \wedge \mathbf{EX} \overbrace{\perp}^{s_0}) = \beta$$

$$s_2 = \beta \vee (\alpha \wedge \mathbf{EX} \overbrace{\beta}^{s_1})$$

$$s_3 = \beta \vee (\alpha \wedge \mathbf{EX} \overbrace{(\beta \vee (\alpha \wedge \mathbf{EX}\beta))}^{s_2})$$

$\dots$

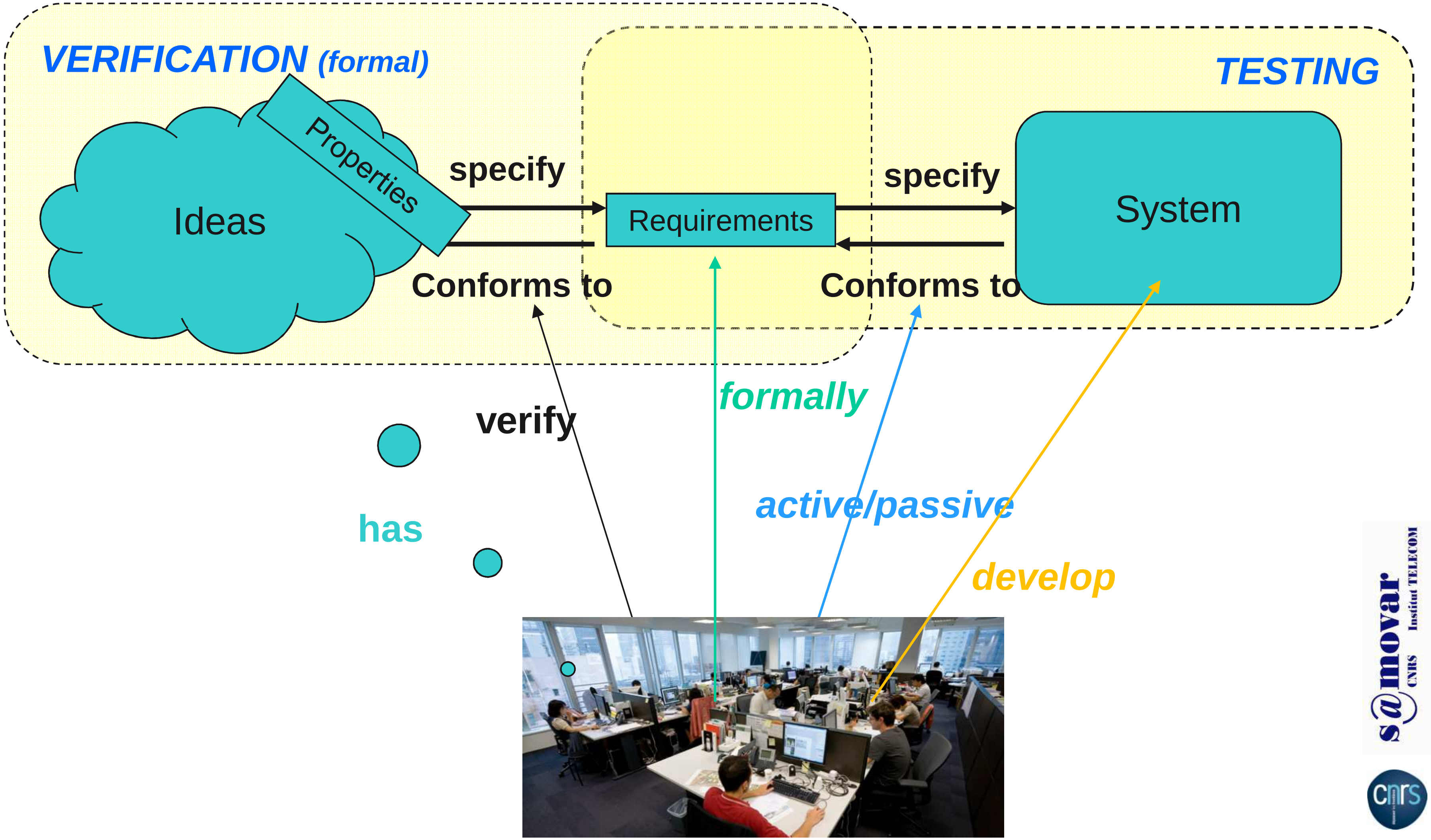


Who are using FDTs?

- **Industrials:**
  - To improve the quality, reliability, reusability, …
    - ❑ The Majors but not only! Airbus, Orange, CISCO, Google, IBM, Daimler, …

- **Universities and research centers**
  - Communicating distributed systems,
  - QoS, QoE, QoBiz
  - To make them evolve to target MANET, VANET, IoT, …

FDT for Testing

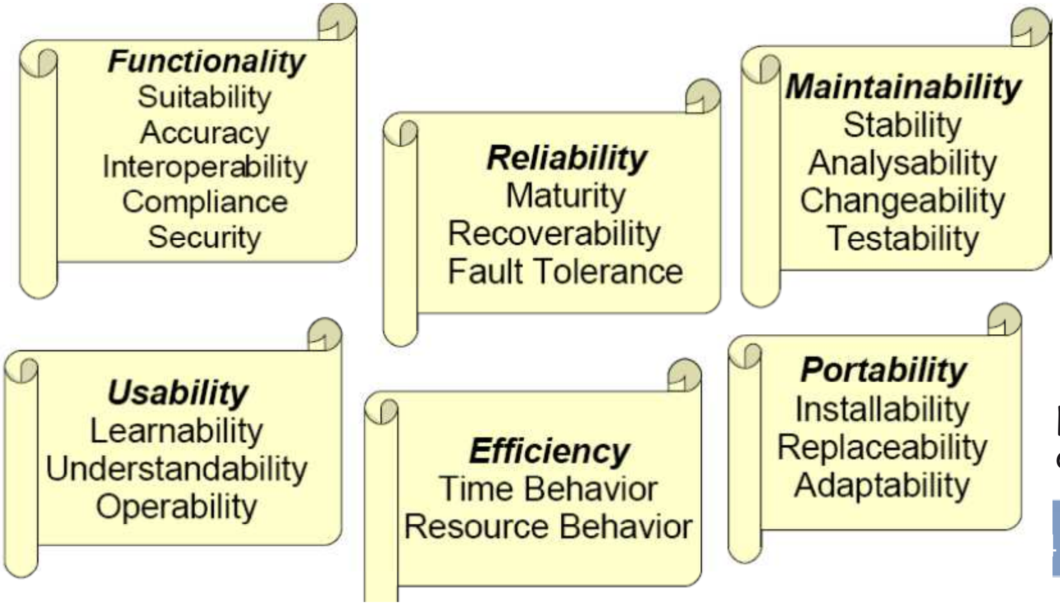


Testing ?

- ***Test* : multiple interpretations**
  - Inspection/review/analysis
  - Debugging
  - Conformance testing
  - Interoperability testing
  - Performance testing
  - Etc… etc…etc…

Various phases of testing

- **Unit testing:**
  - *Process to ensure the smooth functioning of a protocol module.*
- **Integration testing**
  - *Method to integrate multiple modules previously tested by ensuring that everything is good operation.*
- **Conformance testing**
  - *Method to compare the performance of a real system with its formal model.*
- **Interoperability testing**
  - *Process to ensure that a protocol interacts' properly 'with another protocol (which may also be the same NN' / NM).*
- **And many others**



Functionality
Suitability
Accuracy
Interoperability
Compliance
Security

Reliability
Maturity
Recoverability
Fault Tolerance

Maintainability
Stability
Analysability
Changeability
Testability

Usability
Learnability
Understandability
Operability

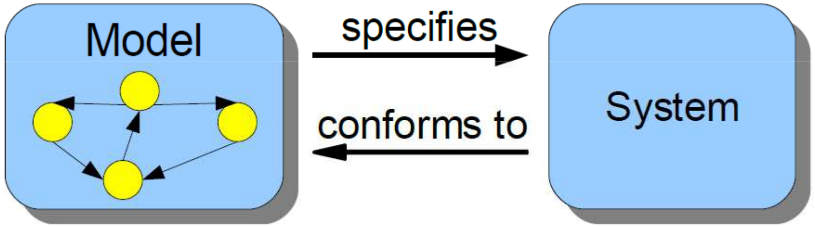
Efficiency
Time Behavior
Resource Behavior

Portability
Installability
Replaceability
Adaptability

[ISO 9126] gives a taxonomy of quality attributes.

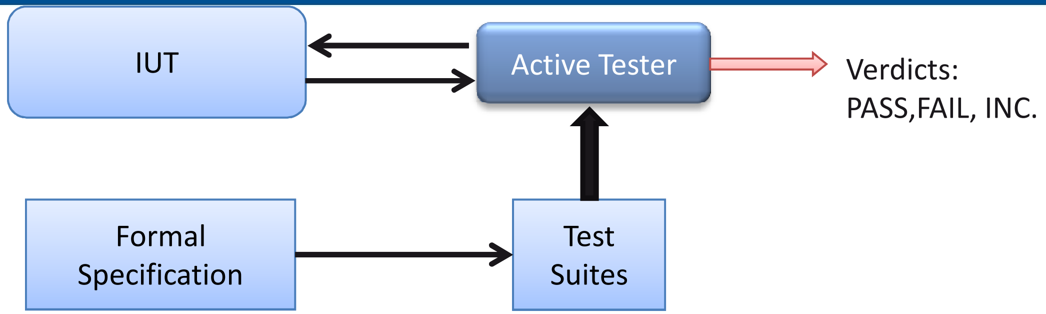
Conformance testing – MBT

- **Remind**: in here, we focus on *functional testing* (≠ non-functional) for protocols, *black box testing* (≠ white/grey box), i.e. **Model-based testing**.
- MBT means testing with the ability to detect failures which are non-conformities to a Model.



- MBT can also include:
  - automatic generation of test cases from the model
  - automatic execution of these test cases at the system (testing arch.)
  - automatic evaluation of the observed behavior, leading to a verdict (PASS,FAIL, INC,...)

Two main techniques


IUT

Active Tester

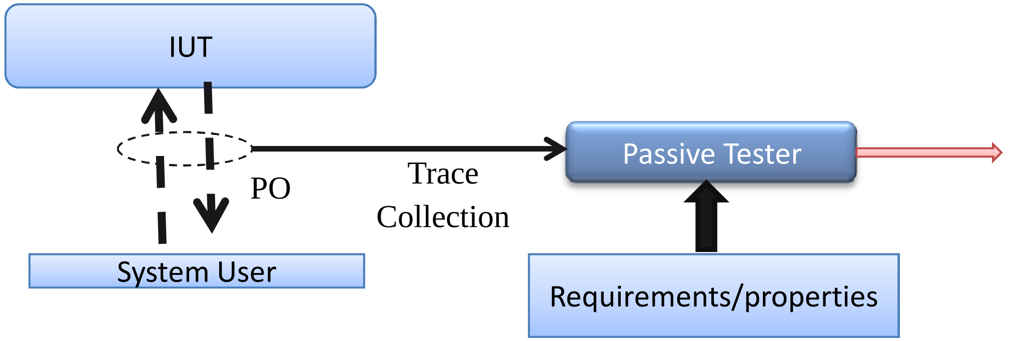
Verdicts:
PASS,FAIL, INC.

Formal Specification

Test Suites

Pros vs cons :

- ☺ Able to target a specific piece of the specification
- ☹ Automatic generation of the tests (complexity)
- ☹ May degrade (crash) the IUT functions


IUT

PO

Trace Collection

Passive Tester

System User

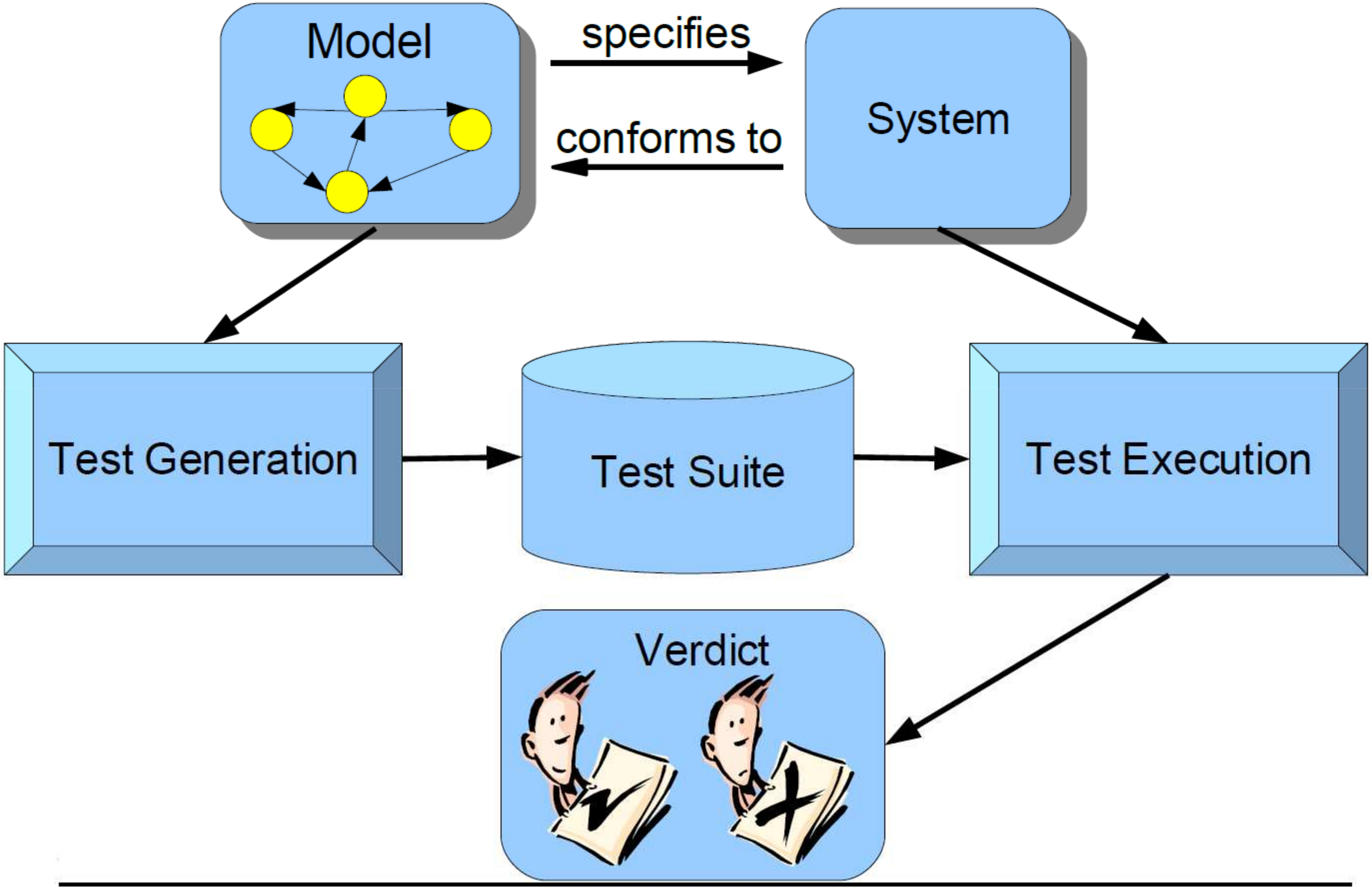
Requirements/properties

Verdicts:
PASS,FAIL,
INC, and others!

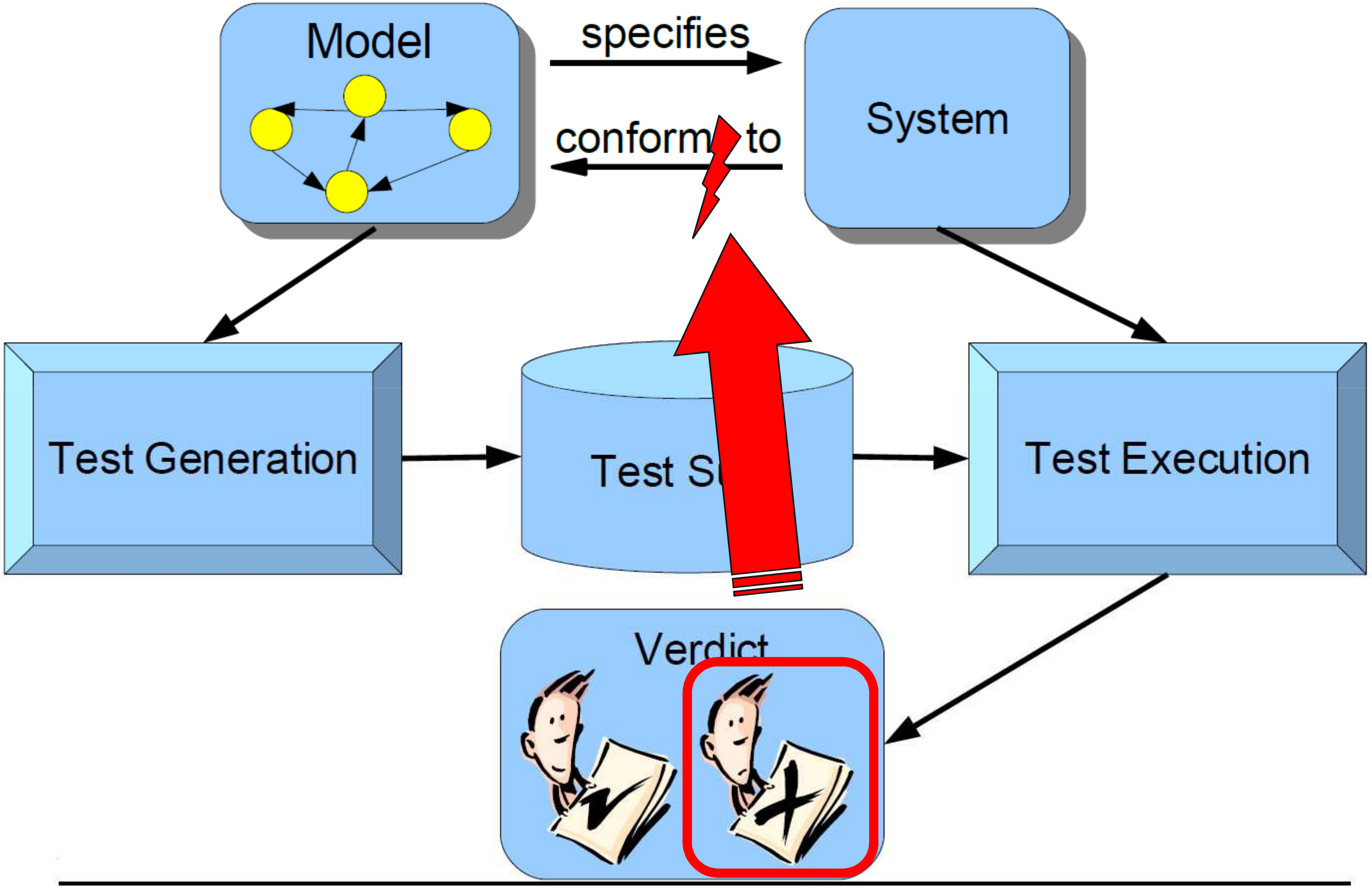
Pros vs cons :

- ☺ No interferences with the IUT
- ☺ No test cases generation
- ☹ Algorithms efficiency (complexity)

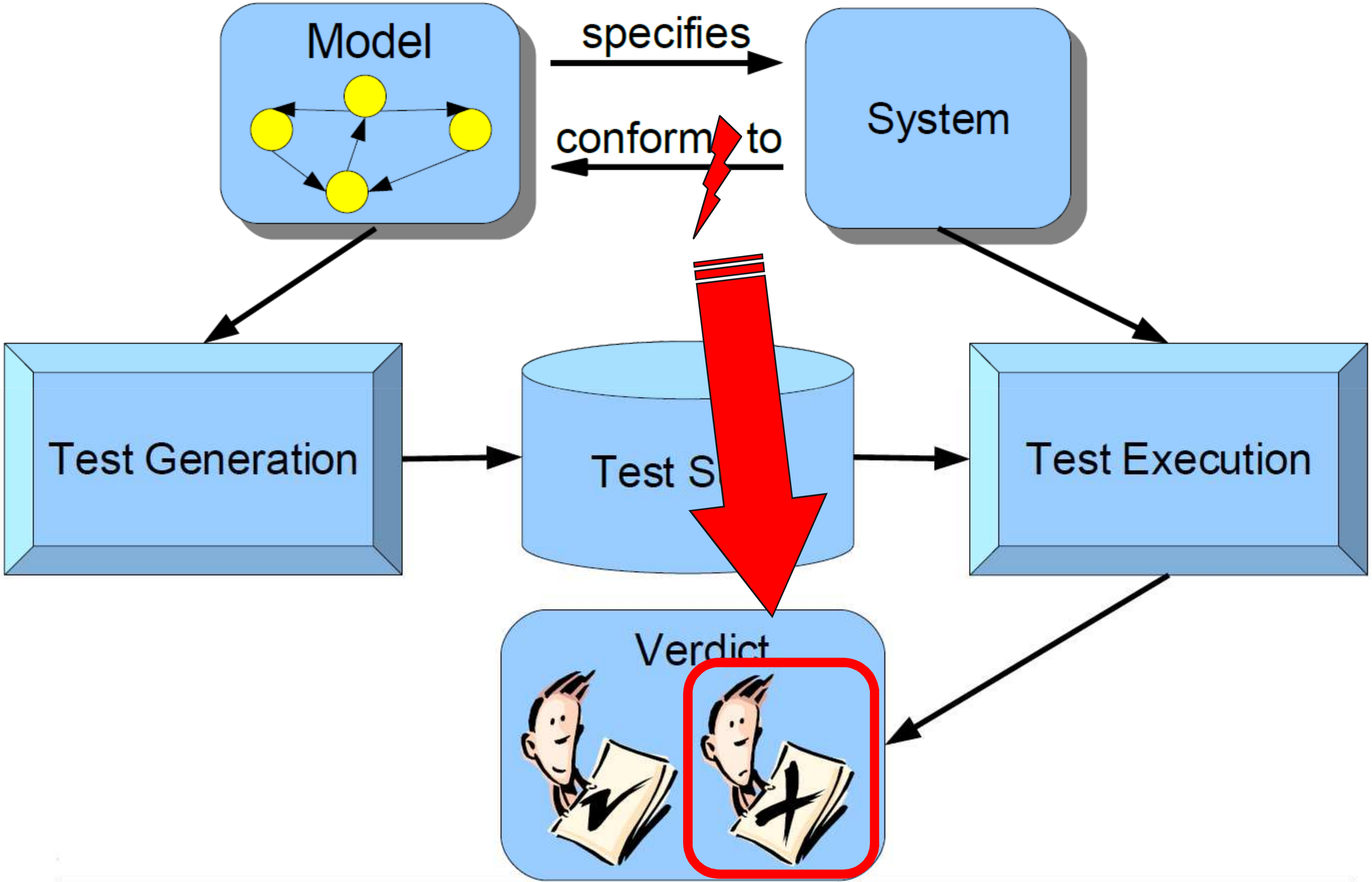
Active Testing



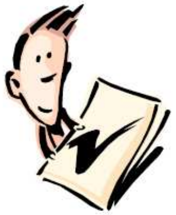
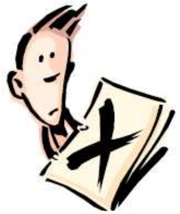
Soundness of Conformance Testing



Completness of Conformance Testing



Soundness and Completness

- A test system, which always says  is sound.
- A test system, which always says  is complete.
- We want test systems that are **sound and complete**!
- But … someone told …

Soundness and Completeness

- "*Testing can never be sound and complete*", dixit Dijkstra … of course, he is right (of course!).
- He refers to the fact, that the number of test cases in a sound and complete test suite is usually infinite (or at least too big).
- If that would not be the case, testing could prove the conformity of the system to the model (given some assumptions on the system).
- In practice, conformity is usually only semi-decidable by finding a failure with a (sound) test system.
- But still: completeness is a crucial property of a sound test system stating that it can *potentially* find all failures!

  ➔ **theoretically possible, but most of the time impossible in practice!**

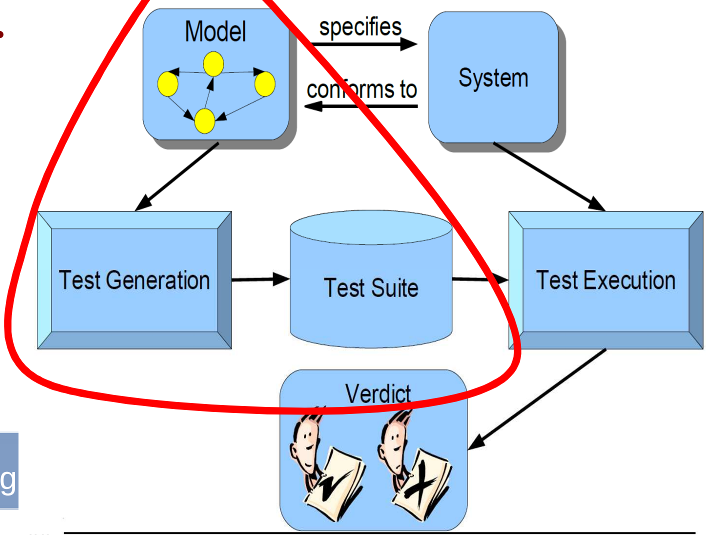
An *all-inclusive* definition of Active Testing

- Protocol testing consists in:
  - Dynamic verification of its behavior…
  - … According to a finite set of test cases …
  - … Aptly selected from an input domain (in practice infinite) …
  - … This compared to the specified expected behavior.

Automatic test cases generation

- For many years, several generation techniques! [Maag2010]
  - W, Wp, HSI, UIOv, DS, H, …
  - Many tools: TGV, Conformiq, JST, SmartTester, …
  - What about the **coverage test criteria**?
  - Outputs are test cases that are most of the time abstract ➔ need to be **concretized**.
  - One common notation: TTCN3

(+ETSI TDL)



Coverage test criteria

- Coverage is a measure of completeness.
- Coverage of 100% never means "complete test" but only the completeness regarding the selected strategy.
- ∃ many strategies and coverage metrics.
- No "best" but some better than others as appropriate.

Ex.: Branch coverage criteria

- Requires that each branch of implementation is performed by at least one test case.
- A test suite T satisfies the criteria for the implementation I iff for every branch B of I, ∃ a test case in T that causes the execution of B.
- NB: the branch coverage is not guaranteed by the states coverage.
- NB: branch coverage mandatory in the unit test.

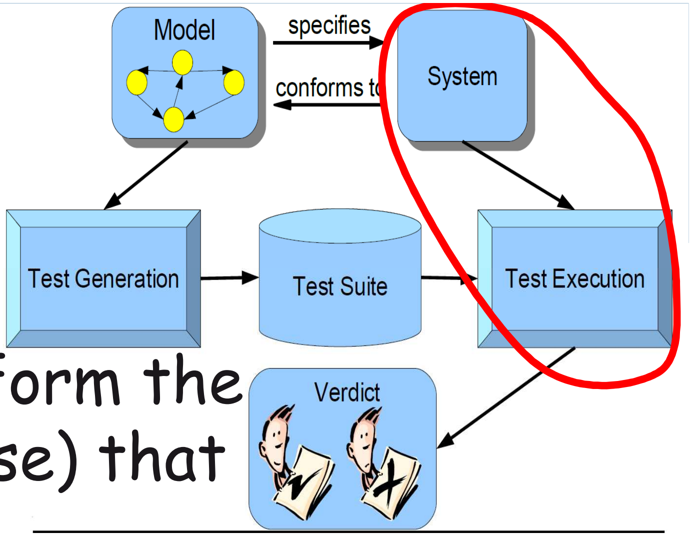
Test cases concretization

- Objective: Relate the abstract values of the model to concrete values of the implementation.

- Synthesized test cases describe sequences of actions that have an interpretation at the abstract level of specification.

- To run these tests on the implementation, we must *implement* these tests in terms of implementation through the interface I/O system.
  - Then **test cases execution** through a well chose **testing architecture**!

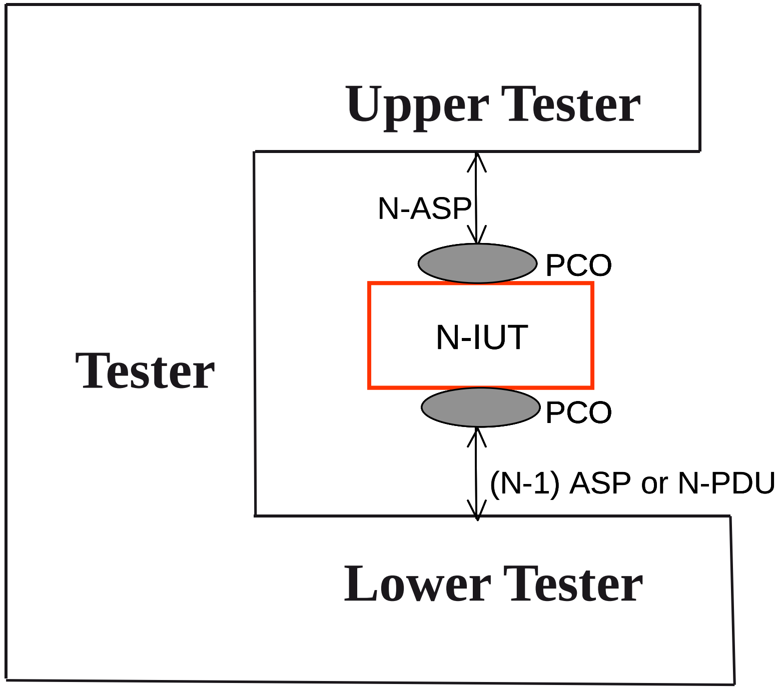
Test suites execution



- Objective: To force the IUT to perform the specific sequence of events (test case) that has been selected.
- Two requirements:
  - Put the system into a state from which the specified tests can be run (pre-condition),
  - Reproduce the desired sequence (known as the Replay problem)
    - ❑ tough issue, especially in the presence of concurrent processes, unreachable process, non-determinism (i.e. same input, different outputs!) and unstable context (wireless, mobile environment).

Testing architectures

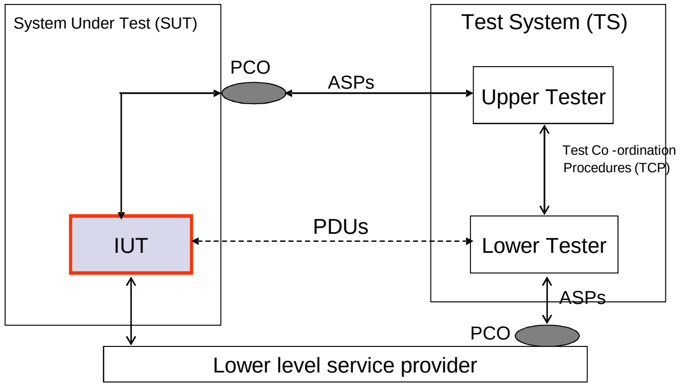
- Testing architectures defined by the ISO 9646
- Conceptually:
  - The tester is directly connected to the IUT and controls its behavior.
  - As presented here: only used when the test are performed locally by the human tester: optimal to detect failures!
  - But not directly useable for conformance testing since the communication between the upper and lower testers has to be done through the “environment” (lower layers).



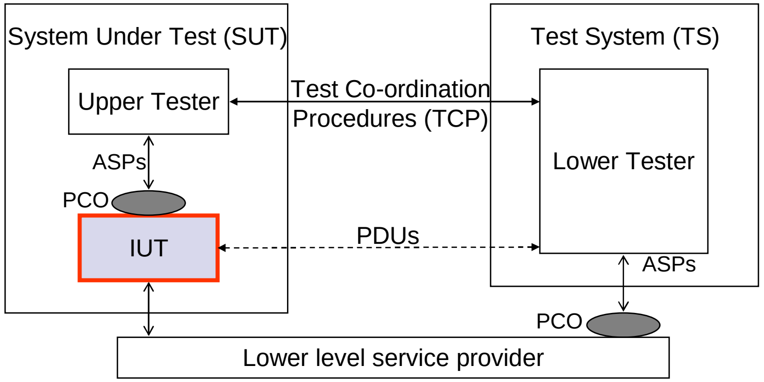
Testing architectures

- **ISO 9646 describes four main architectures:**
  - Local
    - ❑ Upper and lower testers are into the SUT.
    - ❑ The upper tester is directly controlled by the tester and its interface with the IUT is a PCO.
  - Distributed
    - ❑ The upper tester is into the SUT.
    - ❑ It is directly controlled by the tester and its interface with the IUT is a PCO.
  - Coordonnated
    - ❑ The upper tester is into the SUT but is implemented by the human tester.
    - ❑ It is directly controlled by the tester and its interface with the IUT is not directly observable.
  - Remote
    - ❑ No Upper Tester

Testing architectures

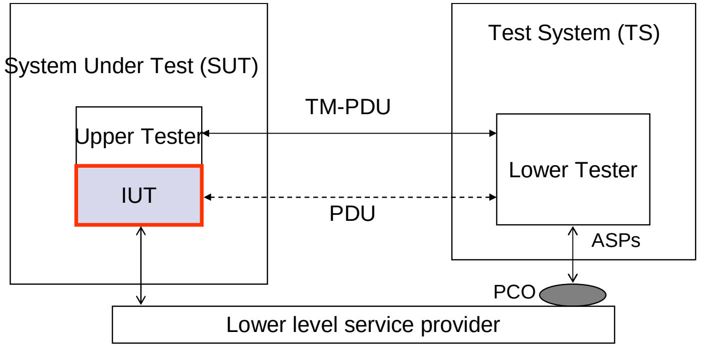


Local

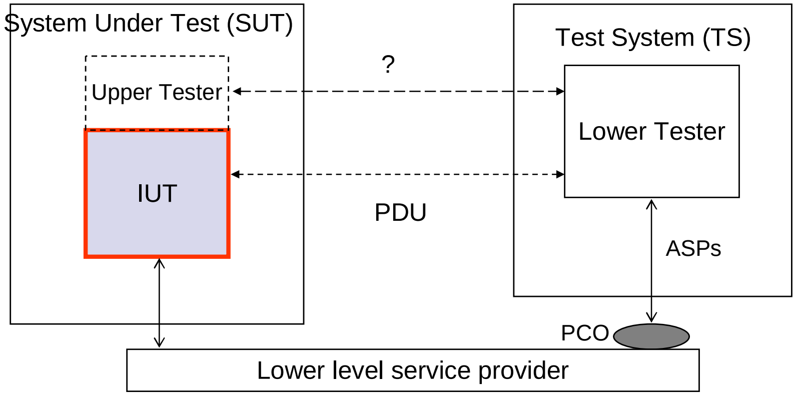


Coordinated

IUT: Implementation under Test
PCO: Point of Control and Observation
ASP: Abstract Service Primitive
PDU: Protocol Data Units



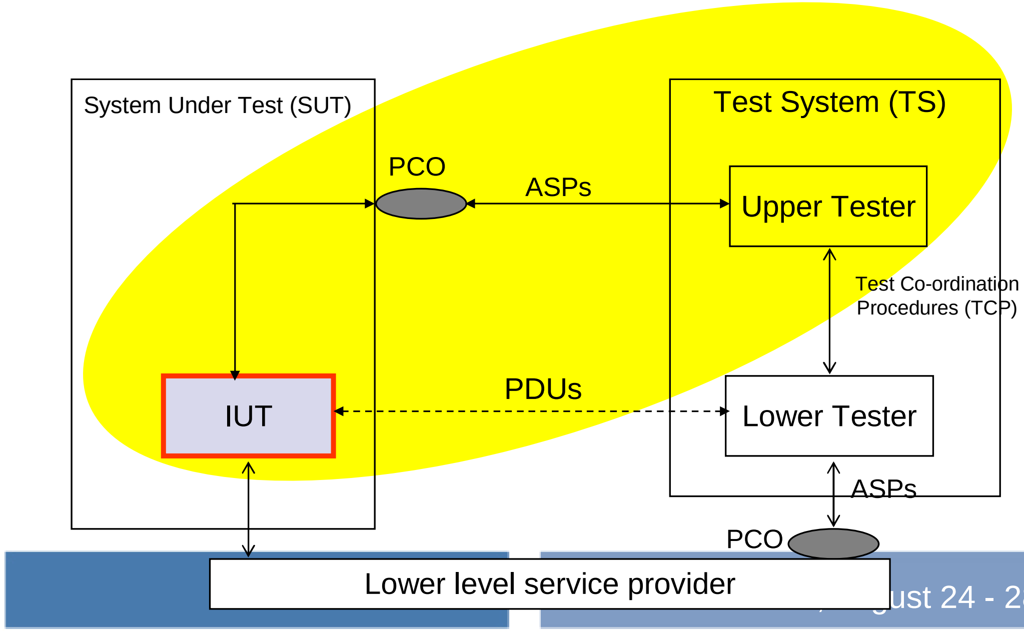
Distributed



Remote

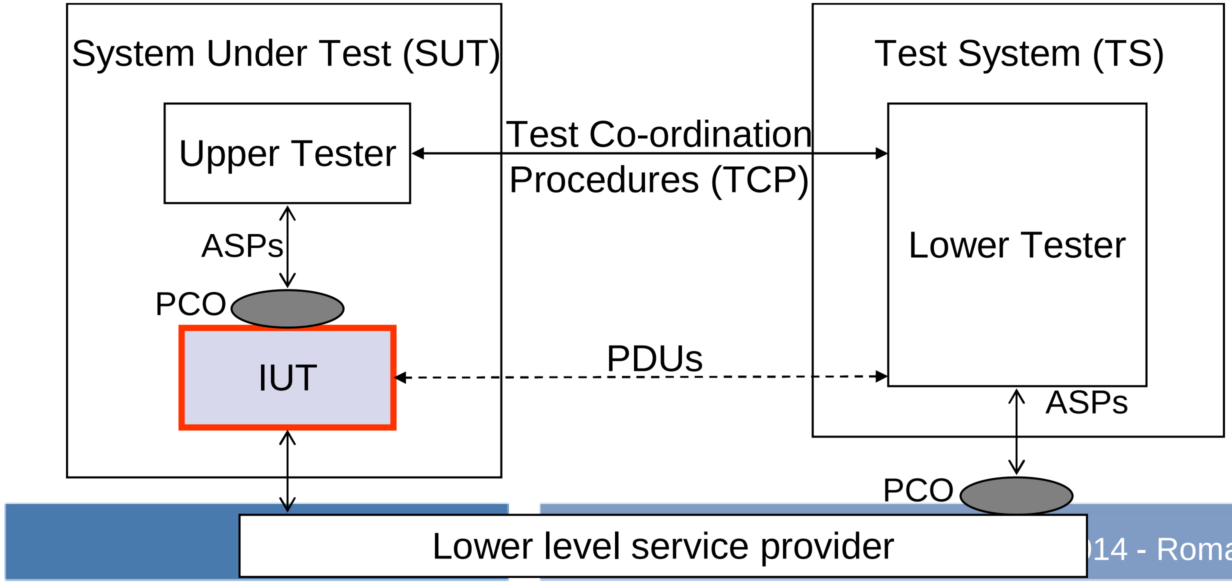
Controllability in the local architecture

- In a real system, the upper layer, here illustrated as the Upper Tester, communicates *directly* with the IUT.
- To be efficient, the communication between the IUT and the UT must be synchrone, both entities should work as they would be directly connected.
  - The yellow area in the figure represents this synchronization
- That’s why we commonly use this local architecture to test the devices.
  - Thus SUT, TS, PCO will be physical elements (devices)
- In order to test programs or software, it is then commonly used to apply *asynchronous* architectures, as it follows.



Distributed architecture

- The Upper Tester is implemented by the human testers,
- The TCP can be manual (by an operator) or automatized,
- The coordination between the UT and LT is a protocol developed by the human testers,
- The test suites are the same as in a local architecture
- Appropriated to test a complete protocol stack layer.



System Under Test (SUT)

Upper Tester

ASPs

PCO

IUT

Test Co-ordination Procedures (TCP)

PDUs

Test System (TS)

Lower Tester

ASPs

PCO

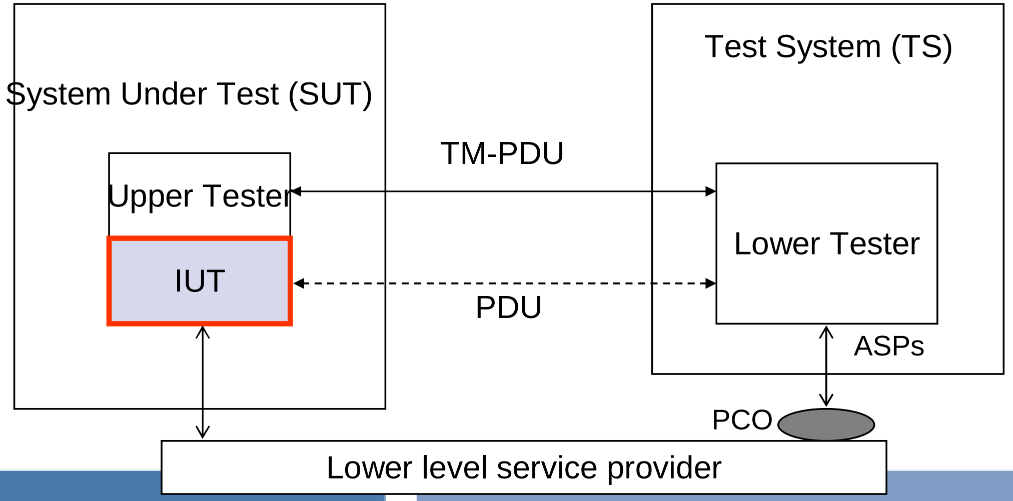
Lower level service provider

Exemple

- To test a phone switch:
  - The UT could simulate the user (directly connected)
  - The LT could
    - ❑ simulate the remote switch
    - ❑ could give instructions to the UT (e.g., pick up the phone)
    - ❑ and controls the answer on the PCO with which it is directly connected.

Coordinated architecture

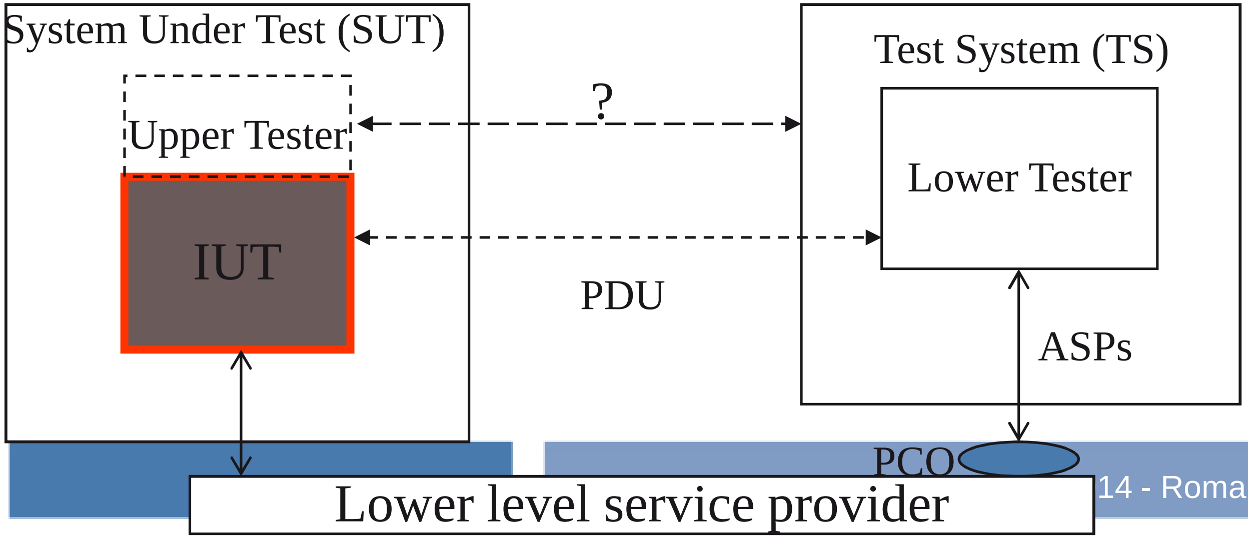
- This architecture has as a main drawback that the IUT has to integrate a UT directly controlled by the tester.
- The Upper Tester is directly and *normally* connected to the IUT, developed by the developer of the IUT.
- No PCO on the SUT side!
- It communicates with the tester by a Test Management Protocol that exchange some TM-PDUs
  - The Test Management Protocol must be normalized since the tester could be any kind of entity
- The coordination between LT and UT (TM-PDUs) has to make part of the test suites.
- The messages detailing this coordination could be:
  - either included in the data parts of the N-PDU (then pass through the LLSP)
  - or transmitted through a separated connexion.
- *Appropriated to test a intermediary layer.*



Remote architecture

- The UT is not necessary, this can be operated by following *informal instructions*.
- The LT can send PDUs that contain data that will be interpreted by the IUT as primitives to be applied to its upper interface (dotted line).

- The possibilities to detect failures are limited since it is not possible to control or observe directly the upper interface.
- However, this architecture is simple and easy developed.
  - Appropriated to test protocols whose the role of the upper interface of the SUT is limited (e.g., FTP)



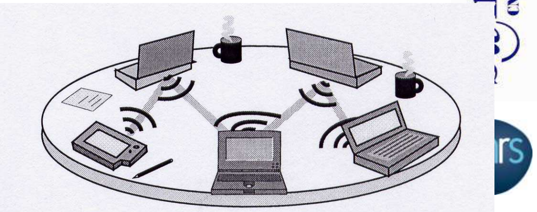
Link Upper Tester / Test System

- All architectures (except the Remote architecture) plan a link between the UT and TS.
- This link is real and must be implemented separately from the LLSP.
- Possibilities:
  - An independent and reliable implemented link?
  - Two persons communicating through another medium?

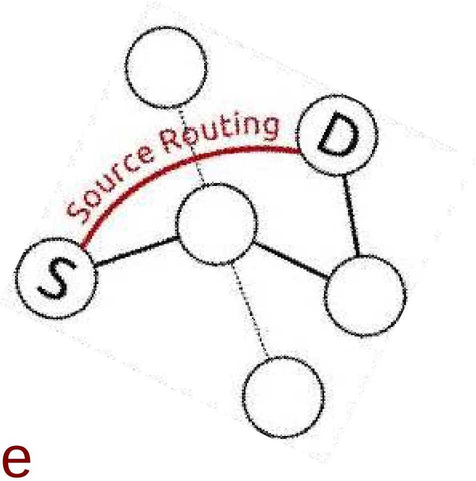
MANET - What's that ?

- "*An mobile ad hoc network is a collection of wireless mobile hosts forming a temporary network without the aid of any established infrastructure or centralized administration*", Johnson et al., 1994

- Infrastructure-less,
- Autonomous,
- The nodes behaves like routers,
- Multi-hops paths
- Dynamic topology (due to mobility or sleep mode),
- Energy constraints due to batteries,
- Heterogeneous radio communications (uni/bi-directional links, different interfaces),



DSR (Dynamic Source Routing) – RFC4728



- Reactive protocol
  - Unicast reactive routing protocol,
  - No routing table but Source Routing,
  - Two mechanisms: Route Discovery and Route Maintenance.

- **Our DSR implementation:**
  - DSR-UU-0.2 runs in the Linux kernel originally created at Uppsala University

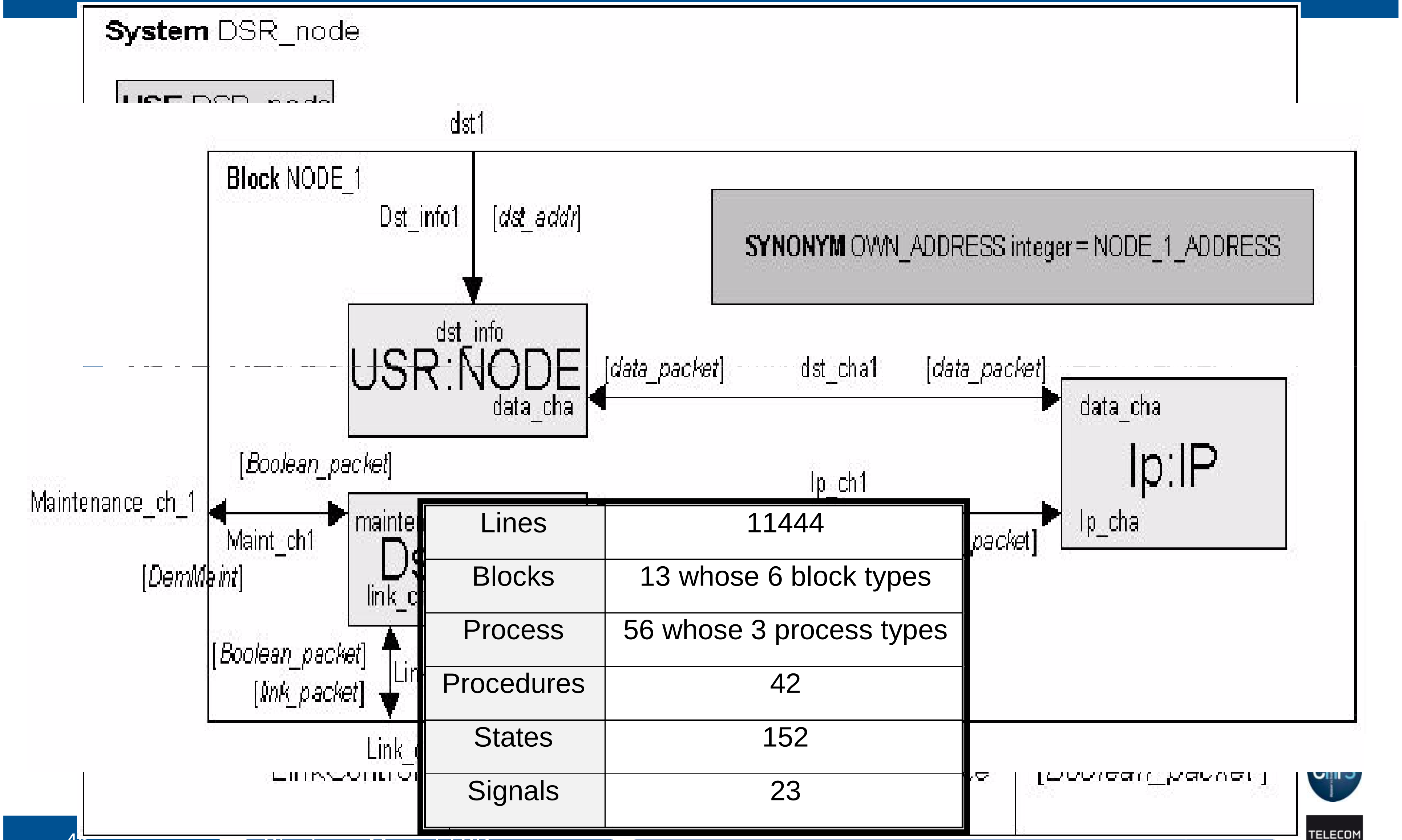
DSR formal specification

- DSR formal model designed in SDL (Specification and Description Language – ITU-T Z.100).
  - EFSM based, allows to specify the system architecture, the functional behaviors.

- Selection of the test purposes: from the DSR standard,

- Test sequences generation: from the specification and testing tools (TestGen-SDL).

DSR formal specification



Lines	11444
Blocks	13 whose 6 block types
Process	56 whose 3 process types
Procedures	42
States	152
Signals	23

DSR-UU Testing

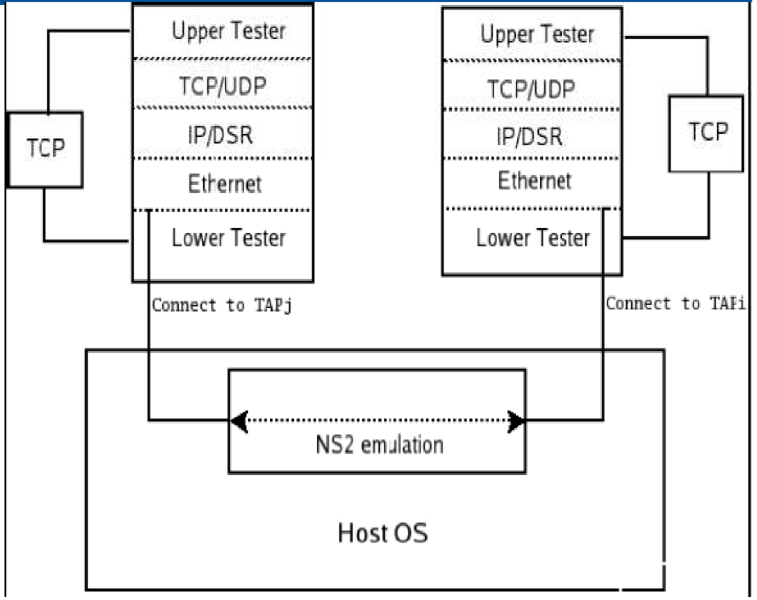
- Testing assumptions:
  - The system could be tested,
  - Destination node exists,
  - Source-destination path connected,
  - Stable routes (to execute the tests),
  - Test scenarios may be replayed.

- TCP = Tester Coordination Procedure
  - as an Oracle developed in C.

- UML = User Mode Linux for NS-2 Emulator with its own DSR implementation
- Fedora Core 4.2.6.16 with virtual wireless network interfaces
- DSR-UU = IUT (Implementation Under Test)

- 22 GOAL test purposes → test seq. generation TESTGEN-SDL

➢ RESULTS with ≅50 test objectives
- No FAIL verdicts – ≈ 5% of PASS verdicts
- ≈ **95% INCONCLUSIVE verdicts**
- Too many packets loss, interferences, uncontrollability of the emulator (≠ specification), so many topological changes from the emulator !



DSR-UU Testing

- **Main reason:** unpredictable topological changes.
  - The formal model did not plan such changes, then not expected in the test sequences.

- **Our solution : the Nodes' Self-Similarity approach.**

Nodes’ self-similarity

- Nodes Self-Similarity (NSS) definition
  - Upon the formal model, composition of nodes that are functionally similar from the IUT view point.

$N = \otimes_{i \in E} N_i \quad \{N_i\}_{i \in E}$ IOEFSM.

$O(N) = \cup_{i \in E} O(N_i)$

$I(N) = \cup_{i \in E} I(N_i) - \cup_{i \in E} O(N_i)$

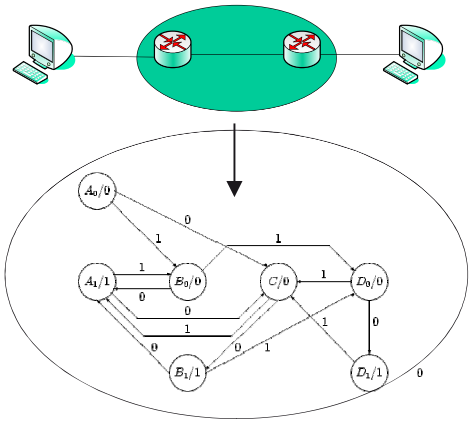
$S(N) = \prod_{i \in E} S(N_i)$

$x(N) = \prod_{i \in E} x(N_i)$

$Tr\,(\,ActHide\varphi\,(N_1 \otimes N_2)) \subseteq Tr(N_3)$ $\quad \varphi = \{$1-hop exchanged messages between the $N_i\}$

$\Leftrightarrow$

$N_1 \otimes N_2$ and N_3 are self-similar

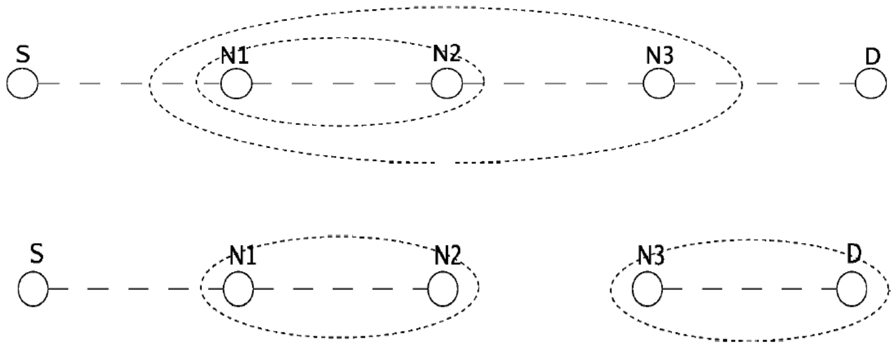


Objectives: To represent ***p*** real interconnected mobile nodes by ***q*** nodes formally modeled with ***q < p***.

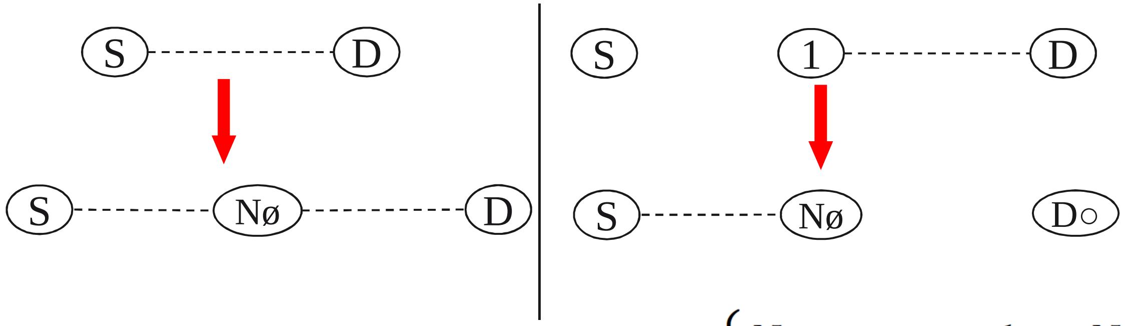
- Reduction of the combinatory explosion
- Reduction of inconclusive verdicts (minimal topology)

Nodes' self-similarity

- **Circumstantial NSS:**
  - Applied on the model
  - From the IUT view point
  - For a specific test objective



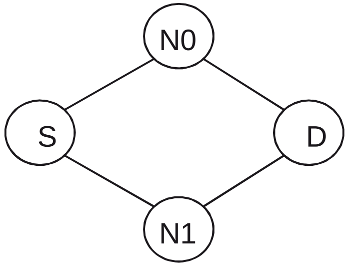
- **Exceptions**



$$N_\circ = \begin{cases} N_\varnothing & \text{where } N_\varnothing \text{ is the neutral element.} \\ N_x \circ N_{x+1} & \text{where } N_x \in \{N_\varnothing\} \cup \bigcup_{i=1}^{n} \{N_i\} \end{cases}$$

Nodes’ self-similarity

- NSS applied to our DSR specification.
  - Three distinguished DSR elements: Source, Destination, other nodes N (routers)
  - By self similarity and according to test objectives plus the RFC: reduction of the model.
    - ❑ Two hops paths needed S-N-D
    - ❑ Two routes needed



NSS for Conformance testing

- Test Execution
  - In *Spec* : 2 routes from *S* to *D* with *4* nodes
  - In *IUT*: *n* routes from *S* to *D* with *p* nodes

  ⇒ TCP manager algorithm to allow the TCP to match similar routes from *Spec* with *IUT* ($O(n^2)$)

- Experimental results on DSRUU and the same test suite
  - **No FAIL verdicts – ≈ 95% of PASS verdicts**
  - **≈ 5% INCONCLUSIVE verdicts**

Open issues in active testing?

- What to do when:
  - Access to the interface unavailable
  - Unreachable component – UT/LT not allowed to be integrated
  - SUT cannot be interrupted
  - SUT cannot be stimulated
- When stop testing?
- What test cases selected?

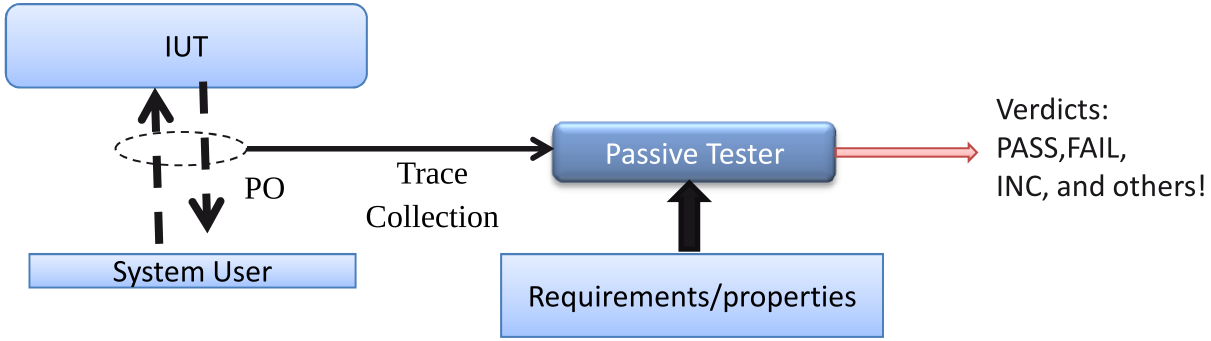
⇒ How to manage the incompleteness of the practice test?

1. To accept and find heuristics such as coverage criteria, time constraints, *randomness,* test objectives, etc.
2. Ask other assumptions leading to the completeness practice.

➔ Let's try **Passive Testing** …

Passive Testing

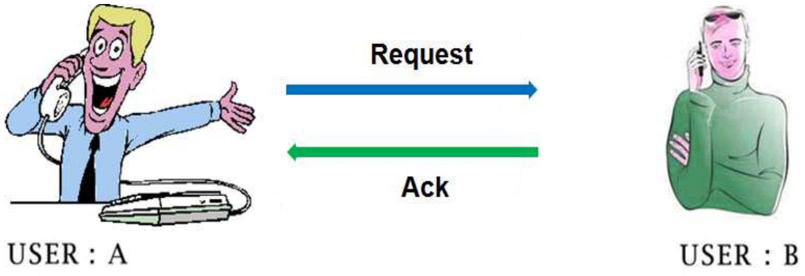
- Objectives: collecting some protocol execution traces in order to analyze if some expected standardized properties are observed (PASS) or not (FAIL, INC, …).



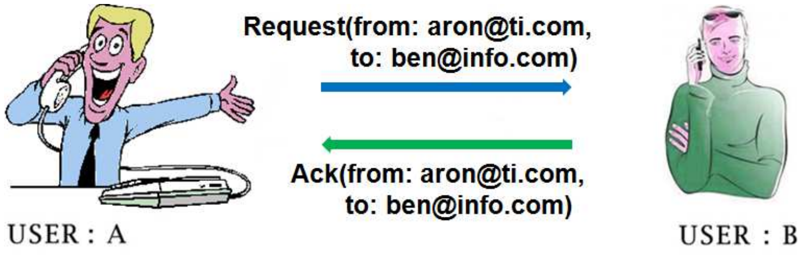
- As mentioned before, ∃ many drawbacks but also many advantages!
    - Complementary to active testing,
    - Very close to runtime monitoring and runtime verification.

Passive Testing

- During several years, passive testing was based on checking **only** the control parts of the protocol!
  - Of course this is no more possible!



System trace: *, Request, * , Request, Ack, * , …


Request(from: aron@ti.com, to: ben@info.com)
Ack(from: aron@ti.com, to: ben@info.com)
USER : A
USER : B

System trace : *, Request(from: aron@ti.com, to: ben@info.com), * , Request(from: aron@ti.com, to: carl@pouf.com), Ack(from: carl@pouf.com, to: ben@info.com), * , …

(i) **ONLY CONTROL PART**

Invariant : Req / Ack

Verdict = **True**

(ii) **CONTROL + DATA PART**

Invariant : Req(A) / Ack(B)

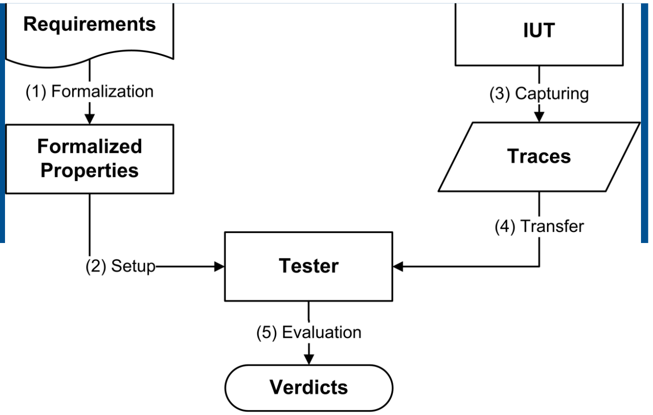
Verdict = **False or Inconclusive**

Passive Testing

- Main issues: the huge amount of data packets, important payload and runtime monitoring algorithms (complexity!) to match **observed traces + expected properties** and to provide test verdicts!

- Challenge: While the control part still plays an important role, data is essential for the execution flow: how to formalize the data relations between multiple packets (data causality)?

Related works



- Runtime verification for LTL and TLTL [Bauer2011]
- A Formal Data-Centric Approach for Passive Testing of Communication Protocols. [Lalanne&Maag2013]
- Formal passive testing of timed systems: theory and tools [Andres2012]
- Model-based performance testing [Barna2011]
- The design and implementation of a domain-specific language for network performance testing [Pakin2007]
- Diperf: An automated distributed performance testing framework [Dumitrescu2004]
- A passive testing approach based on invariants: application to the WAP [Bayse2005]

The DataMon approach [Lalanne&Maag2013]

Basics:

Data

- **Atomic**: A set of numeric or string values
- **Compound**: The set of pairs $\{(l_i, v_i) \mid l_i \in L \wedge v_i \in D_i \cup \{\epsilon\}, i = 1...n\}$ where $L = \{l_1, ..., l_n\}$ is a predefined set of labels and *Di* are data domains.

Example:

$$m = \{(method, \textbf{`INVITE'}), (time, \text{`644.294133000'}), (status, \epsilon), (from, \text{`alice@a.org'}), (to, \text{`bob@b.org'}), (cseq, \{(num, 7), (method, \textbf{`INVITE'})\})\}$$

The DataMon approach

Horn clause: Horn clause is a clause with at most one positive.

Logician **Alfred Horn**, who first pointed out their significance in 1951,
"On sentences which are true of direct unions of algebras"
Journal of Symbolic Logic, 16, 14–21

Disjunction form: $\neg p \vee \neg q \vee \ldots \vee \neg t \vee u$

Implication form: $u \leftarrow p \wedge q \wedge \ldots \wedge t$

assume that u holds if p and q and ... and t all hold

The DataMon approach

- **Term** $term ::= c \mid x \mid x.l.l...l$ Where ***c*** is a constant, ***x*** is a variable, ***l*** represents a label

- **Atom**

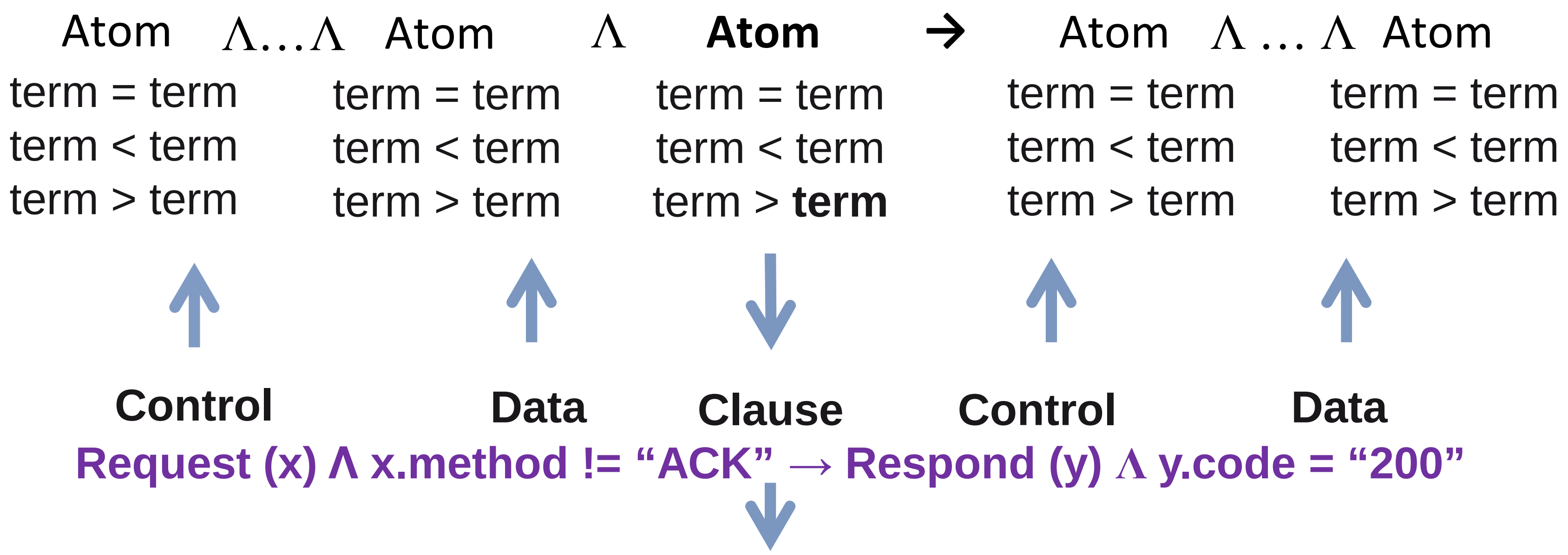
$$A ::= p(\overbrace{term, ..., term}^{k}) \\ \mid term = term \\ \mid term \neq term \\ \mid term < term \\ \mid term + term = term$$

Where $p(term, ..., term)$ is a predicate of label ***p*** and arity ***k***.

- **Formula** $\phi ::= A_1 \wedge ... \wedge A_n \mid \phi \rightarrow \phi \mid \forall_x \phi \mid \forall_{y>x} \phi \mid \forall_{y<x} \phi \mid \exists_x \phi \mid \exists_{y>x} \phi \mid \exists_{y<x} \phi$ Where $A_1, ..., A_n$ are atoms

The DataMon approach

Protocol properties are defined as Horne based formulas.



$\text{Atom} \wedge \ldots \wedge \text{Atom} \quad \wedge \quad \textbf{Atom} \quad \rightarrow \quad \text{Atom} \wedge \ldots \wedge \text{Atom}$

term = term	term = term	term = term	term = term	term = term
term < term	term < term	term < term	term < term	term < term
term > term	term > term	term > **term**	term > term	term > term

Control **Data** **Clause** **Control** **Data**

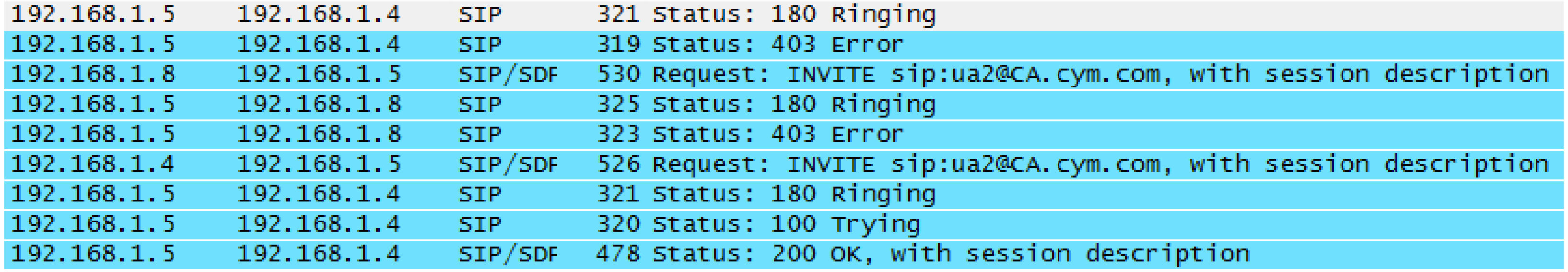
Request (x) ∧ x.method != "ACK" → Respond (y) ∧ y.code = "200"

Formula:

$$\forall_x(\text{request }(x) \wedge x.\text{method} \mathrel{!=} \text{“ACK”} \rightarrow \exists_{y>x} (\text{nonprovisional }(y) \wedge \text{Respond }(y,x)))$$

The DataMon approach

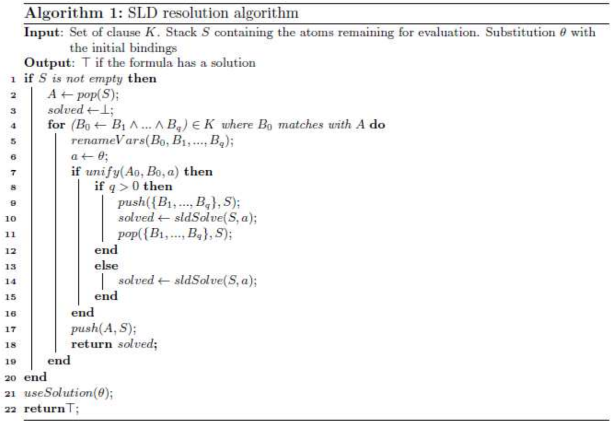
Trace:



```
192.168.1.5     192.168.1.4     SIP         321 Status: 180 Ringing
192.168.1.5     192.168.1.4     SIP         319 Status: 403 Error
192.168.1.8     192.168.1.5     SIP/SDF     530 Request: INVITE sip:ua2@CA.cym.com, with session description
192.168.1.5     192.168.1.8     SIP         325 Status: 180 Ringing
192.168.1.5     192.168.1.8     SIP         323 Status: 403 Error
192.168.1.4     192.168.1.5     SIP/SDF     526 Request: INVITE sip:ua2@CA.cym.com, with session description
192.168.1.5     192.168.1.4     SIP         321 Status: 180 Ringing
192.168.1.5     192.168.1.4     SIP         320 Status: 100 Trying
192.168.1.5     192.168.1.4     SIP/SDF     478 Status: 200 OK, with session description
```

- **Pass: The requirement is satisfied**
- **Fail: The requirement is not satisfied**
- **Inconclusive: Uncertain verdict**

Algorithm:



Algorithm 1: SLD resolution algorithm

Input: Set of clause K. Stack S containing the atoms remaining for evaluation. Substitution θ with the initial bindings

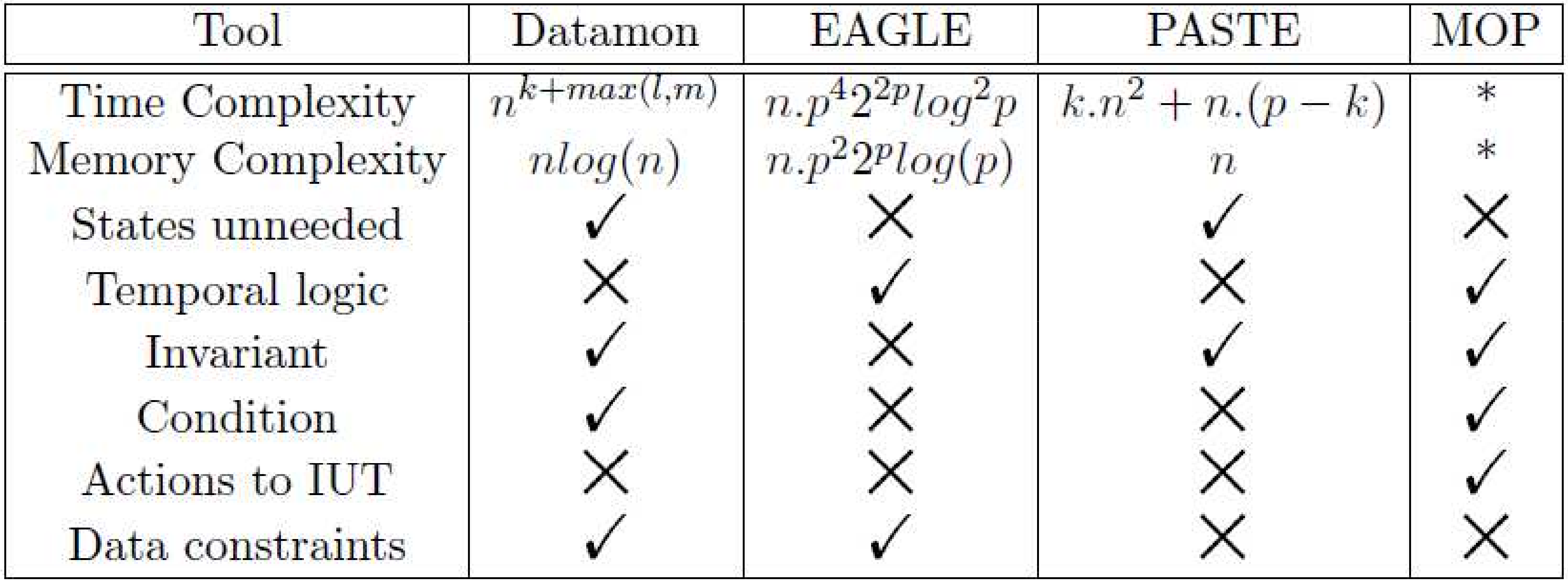
Output: $\top$ if the formula has a solution

```
if $S$ is not empty then
    $A \leftarrow pop(S)$;
    $solved \leftarrow \bot$;
    for $(B_0 \leftarrow B_1 \wedge ... \wedge B_q) \in K$ where $B_0$ matches with $A$ do
        $renameVars(B_0, B_1, ..., B_q)$;
        $a \leftarrow \theta$;
        if $unify(A_0, B_0, a)$ then
            if $q > 0$ then
                $push(\{B_1, ..., B_q\}, S)$;
                $solved \leftarrow sldSolve(S, a)$;
                $pop(\{B_1, ..., B_q\}, S)$;
            end
            else
                $solved \leftarrow sldSolve(S, a)$;
            end
        end
        $push(A, S)$;
        return $solved$;
    end
end
$useSolution(\theta)$;
return$\top$;
```

Formalized requirement:

$$\forall_x(\text{request }(x) \wedge x.\text{method} \mathrel{!=} \text{“ACK”} \rightarrow \exists_{y>x}(\text{nonprovisional }(y) \wedge \text{Respond }(y,x)))$$

The DataMon approach



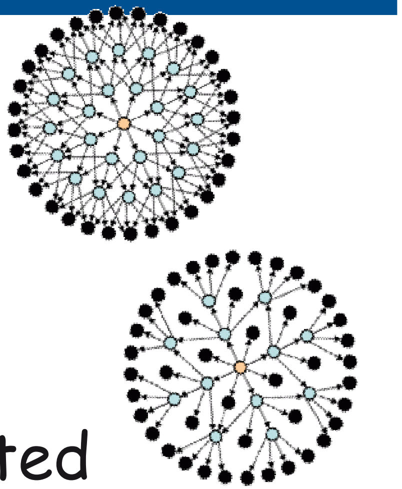
Tool	Datamon	EAGLE	PASTE	MOP
Time Complexity	$n^{k+max(l,m)}$	$n.p^4 2^{2p} log^2 p$	$k.n^2 + n.(p-k)$	*
Memory Complexity	$nlog(n)$	$n.p^2 2^p log(p)$	n	*
States unneeded	✓	✗	✓	✗
Temporal logic	✗	✓	✗	✓
Invariant	✓	✗	✓	✓
Condition	✓	✗	✗	✓
Actions to IUT	✗	✗	✗	✓
Data constraints	✓	✓	✗	✗

m: # right formula quantifiers
k, l,: # left formula quantifiers
n: trace length
p: # clauses

Real Experiments with Passive Testing

- **2 industrial case studies:**
  - Into a real MANET through a Asian ICT project (MAMI): to test the **routing protocol OLSR** [IETF RFC 3626].
  - On a real IMS platform (hosted by Alcatel-Lucent) to test the **Session Initiation Protocol SIP** [IETF RFC 3261 (+RFC 3265)].

OLSR (Optimized Link State Routing) – RFC3626



- Proactive protocol
  - Control packets are periodically broadcasted through the network,
    => The routing tables are continuously updated

- MPR - Multi Point Relays
  - Limit the flooding into the network
  - Routes are optimal
  - Routes are always available

Why Passive Testing in this OLSR case?

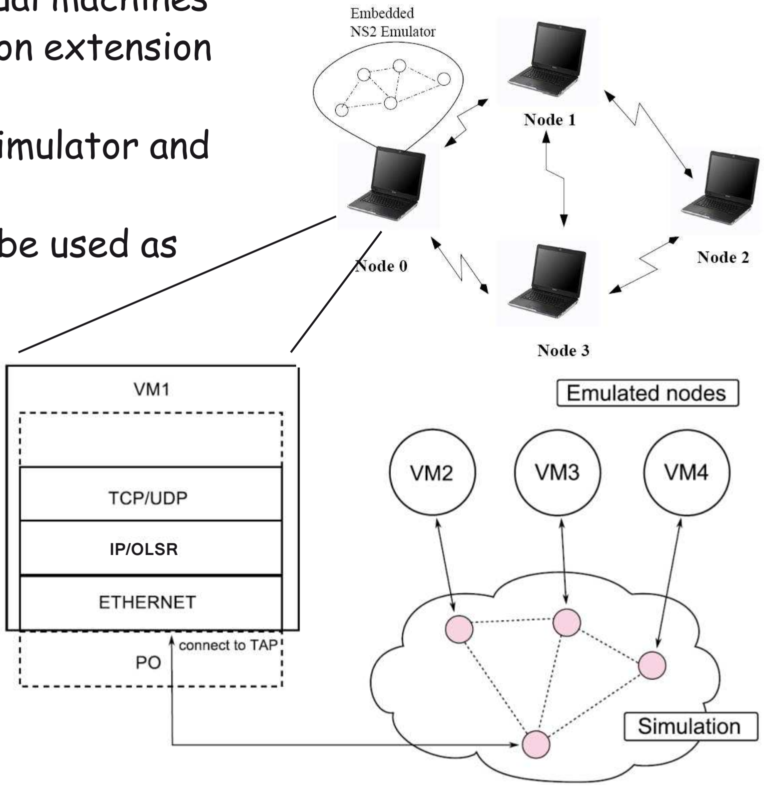
- Active testing approaches were applied,
- Interesting verdict results on functional properties have been provided.
- But due to dynamicity, mobility and topological changes → many inconclusive verdicts obtained (>60% of all obtained verdicts)!

➔ Passive testing approach to bypass that main issue

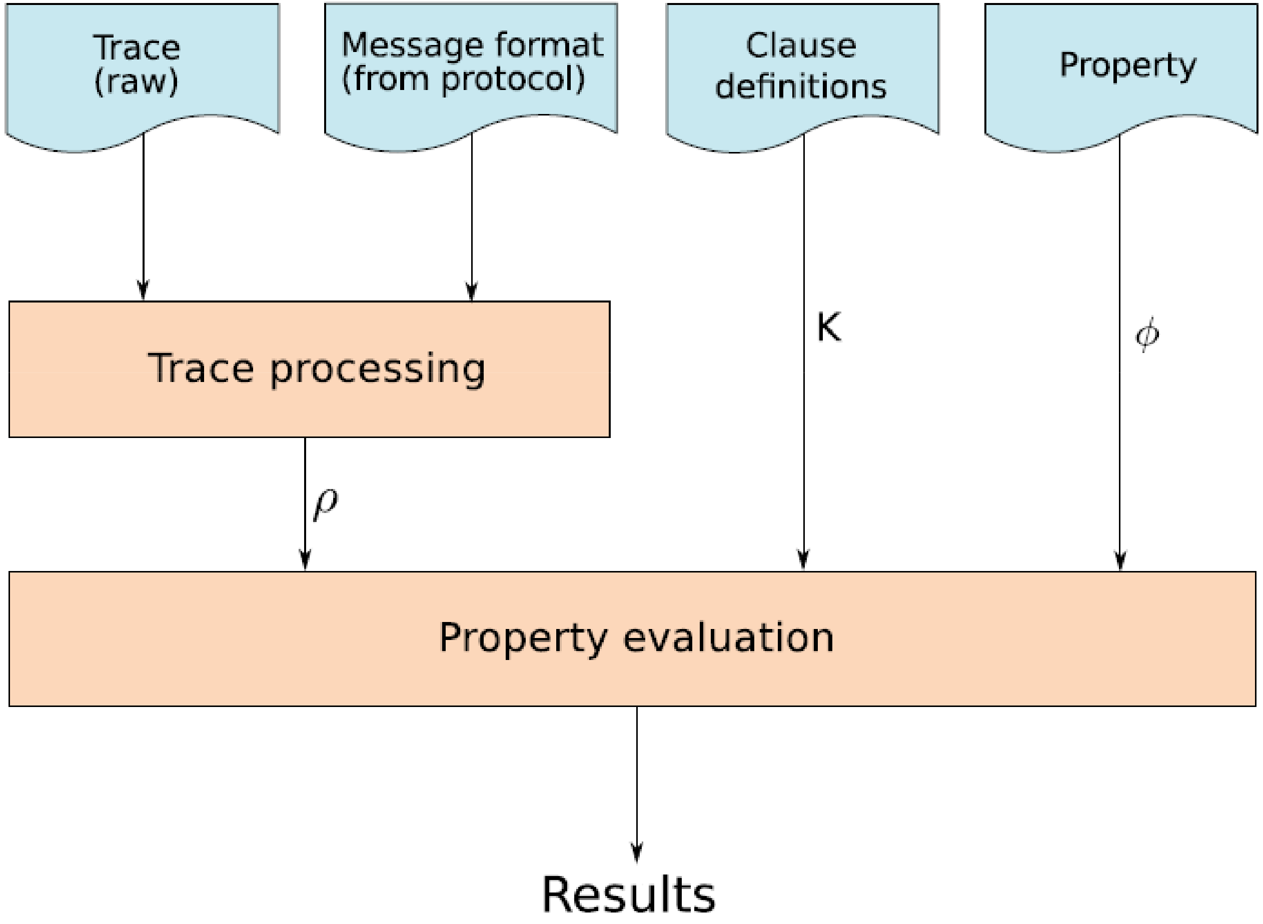
The OLSR testbed

- **4 real OLSR nodes + an NS2 emulated wireless testbed:**
  - Real nodes:
    - ❑ 3 laptops with 802.11 a/b/g
    - ❑ 1 laptop with a wireless adapter WPN111
    - ❑ OLSR implementation: olsrd-0.6.3
  - NS2e:
    - ❑ A simulator: it manages the nodes' mobility and wireless communication in their simulated environment. Virtual machines are connected to the simulation through an emulation extension (UML),
    - ❑ A host (or focal) machine: this machine hosts the simulator and the nodes emulated through virtual machines,
    - ❑ Virtual machines (VMs). But real machines can also be used as additional nodes.

- **Traces captured by wireshark in *eth0***
  - ❑ XML format + XSL style sheets to filter and format the information



Datamon


Trace (raw)
Message format (from protocol)
Clause definitions
Property
Trace processing
K
ϕ
ρ
Property evaluation
Results

Four functional OLSR properties

- **Property 1:** This property expresses that if a message (*msg*) is received announcing an asymmetrical link with the node of address '10.0.0.2', then at a previous point in the trace (*p_msg*), a message must have been sent by the PO broadcasting its own address.

$$\begin{aligned}&\forall_{msg}(asym\_bcast\_recv(\text{`10.0.0.2'}, msg, addr) \rightarrow\\ &\qquad \exists_{p\_msg<msg} initial\_bcast(\text{`10.0.0.2'}, p\_msg))\end{aligned}$$

The initial broadcast is defined by the clause:

$$\begin{array}{ll} initial\_bcast(local\_addr, message) & \leftarrow \\ \quad message.message\_type = 201 & \wedge \\ \quad message.origin\_addr = local\_addr & \wedge \\ \quad message.link\_type = \epsilon & \end{array}$$

- **Property 2:** if a message is received announcing the establishment of a symmetrical link with a given node (in this case with address '10.0.0.2') then a previous message must have been received from the same node broadcasting the creation of an asymmetrical link.

$$\begin{aligned}&\forall_{msg}(sym\_bcast\_recv(\text{`10.0.0.2'}, msg, addr) \rightarrow\\ &\qquad \exists_{p\_msg<msg} asym\_bcast\_recv(\text{`10.0.0.2'}, p\_msg, addr))\end{aligned}$$

- **Property 3:** The formula will return true only if when an MPR broadcast is observed by a node, a broadcast establishing a symmetrical link must have been observed before in the trace.

$$\begin{aligned}&\forall_{msg}(mpr\_bcast\_recv(\text{`10.0.0.2'}, msg, addr) \rightarrow\\ &\qquad \exists_{p\_msg<msg} sym\_bcast\_recv(\text{`10.0.0.2'}, p\_msg, addr))\end{aligned}$$

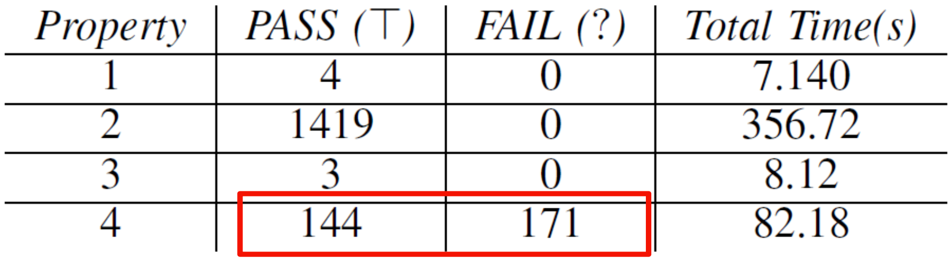
- **Property 4:** In order to update their routing tables, nodes must be kept regularly informed of changes in the topology. TC messages are emitted periodically by each MPR to all nodes in the network to declare its MPR selector set. From this information, it is possible to compute optimal routes from a node to any destination. Property 4 expresses that if the local node sends a TC message broadcasting a list of neighbors, then at least one of those neighbors must have declared the local node its MPR.

$$\begin{aligned}&\forall_{msg}(tc\_bcast(\text{`10.0.0.2'}, msg, neigh\_list) \rightarrow\\ &\qquad \exists_{p\_msg<msg}(mpr\_bcast\_recv(\text{`10.0.0.2'}, p\_msg, addr)\\ &\qquad\quad \wedge\, in\_neigh\_list(addr, neigh\_list)))\end{aligned}$$

OLSRd testing – Experimental results

- Inputs:
  - *properties* file - Java
  - PDML XML trace (Wireshark)
- Outputs:
  - PASS, FAIL or INC
- Tool performance:
  - Results for an ~100Mo trace / 100 000 packets.

- Results
  - Several Pass as expected!
  - Several Fail + Inc !!
    - ❑ The Inc were expected (due to the mobility, topology, dynamicity and wireless connections)
    - ❑ The Fail were not !!
      ➔ Still n analysis.

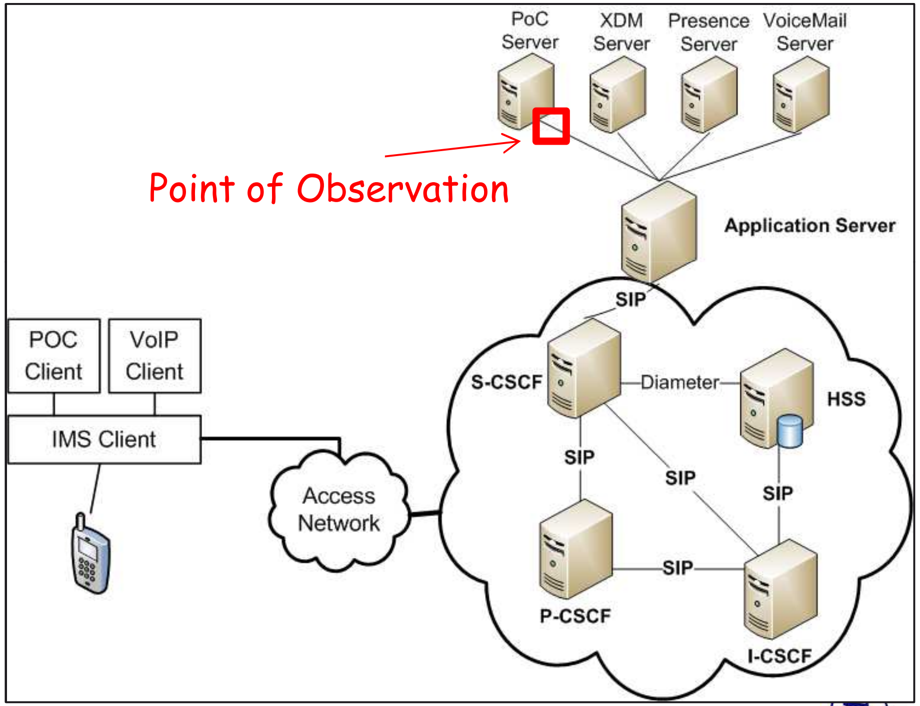


Property	*PASS* ($\top$)	*FAIL* (?)	*Total Time(s)*
1	4	0	7.140
2	1419	0	356.72
3	3	0	8.12
4	144	171	82.18

IMS / SIP Testbed – Alcatel-Lucent

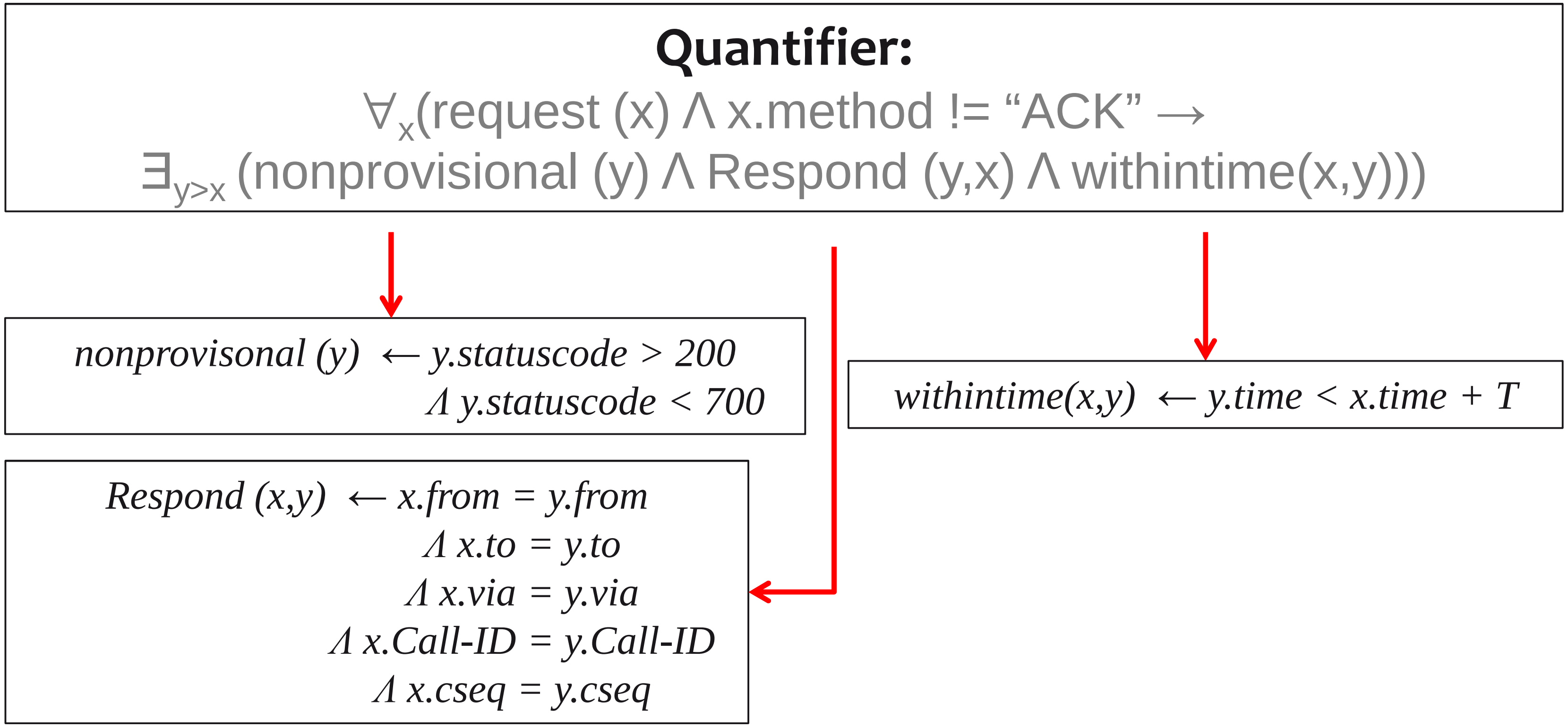
- Main goal:
  - To collect SIP traces on the PoC Server,
  - To define functional PoC properties to be tested,
  - To formalize them and to provide both PDML XML traces + Formulas in the tool

➔ To obtain **test verdicts**!



SIP testing

Property: For every request there must be a response, within T=0.5s



Quantifier:

$$\forall_{x}(\text{request }(x) \land x.\text{method} \mathrel{!=} \text{“ACK”} \rightarrow$$
$$\exists_{y>x}\,(\text{nonprovisional }(y) \land \text{Respond }(y,x) \land \text{withintime}(x,y)))$$

$$\textit{nonprovisonal }(y) \leftarrow y.statuscode > 200 \land y.statuscode < 700$$

$$\textit{withintime}(x,y) \leftarrow y.time < x.time + T$$

$$\textit{Respond }(x,y) \leftarrow x.from = y.from \land x.to = y.to \land x.via = y.via \land x.Call\text{-}ID = y.Call\text{-}ID \land x.cseq = y.cseq$$

SIP testing

Property: For every request there must be a response, within T=0.5s

Quantifier:

$$\forall_{x} (\text{request } (x) \land x.\text{method} \mathrel{!=} \text{“ACK”} \rightarrow$$
$$\exists_{y>x} (\text{nonprovisional } (y) \land \text{Respond } (y,x) \land \text{withintime}(x,y)))$$

Based on

Property: For every request there must be a response

Quantifier:

$$\forall_{x} (\text{request } (x) \land x.\text{method} \mathrel{!=} \text{“ACK”} \rightarrow$$
$$\exists_{y>x} (\text{nonprovisional } (y) \land \text{Respond } (y,x)))$$

SIP testing

Property: For every request there must be a response, within T=0.5s

Quantifier:

$$\forall_x(\text{request }(x) \wedge x.\text{method} \mathrel{!=} \text{“ACK”} \rightarrow \exists_{y>x} (\text{nonprovisional }(y) \wedge \text{Respond }(y,x) \wedge \text{withintime}(x,y)))$$

Trace	No. of packets	Pass	Fail	Inc	Time (ms)
1	500	33	0	322	9.21
2	1000	85	0	636	26.26
3	1500	187	0	872	58.89
4	2000	427	0	1014	95.21
5	2500	535	0	1308	179.42

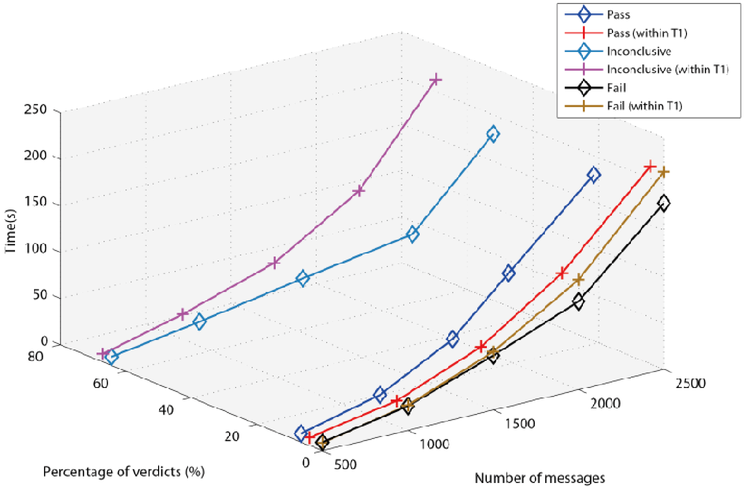
Trace	No. of packets	Pass	Fail	Inc	Time (ms)
1	500	150	335	0	8.67
2	1000	318	687	0	27.11
3	1500	504	1003	0	62.92
4	2000	674	1340	0	118.69
5	2500	798	1740	0	213.17

SIP testing

Trace	No. of packets	Pass	Fail	Inconclusive	Time (ms)
1	500	33	0	322	9.21
2	1000	85	0	636	26.26
3	1500	187	0	872	58.89
4	2000	427	0	1014	95.21
5	2500	535	0	1308	179.42

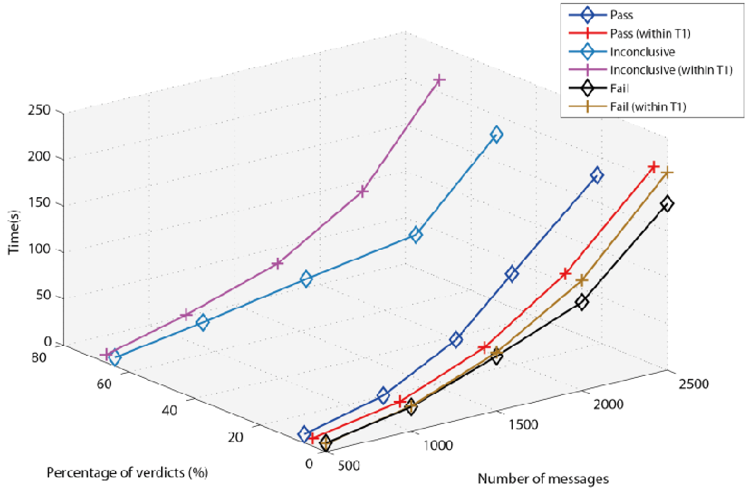
Trace	No. of packets	Pass	Fail	Inc	Time (ms)
1	500	20	335	1	8.67
2	1000	34	687	0	27.11
3	1500	56	1003	2	62.92
4	2000	101	1340	0	118.69
5	2500	103	1740	1	213.17

Trace	No. of packets	Pass	Con-Fail	Per-Fail	Inconclusive
1	500	20	0	334	1
2	1000	34	0	687	0
3	1500	56	0	1001	2
4	2000	101	0	1340	0
5	2500	103	0	1739	1



SIP testing

Trace	No. of packets	Pass	Con-Fail	Per-Fail	Inconclusive
1	500	20	0	334	1
2	1000	34	0	687	0
3	1500	56	0	1001	2
4	2000	101	0	1340	0
5	2500	103	0	1739	1



Pass:
Indicates the messages satisfy all the requirements.

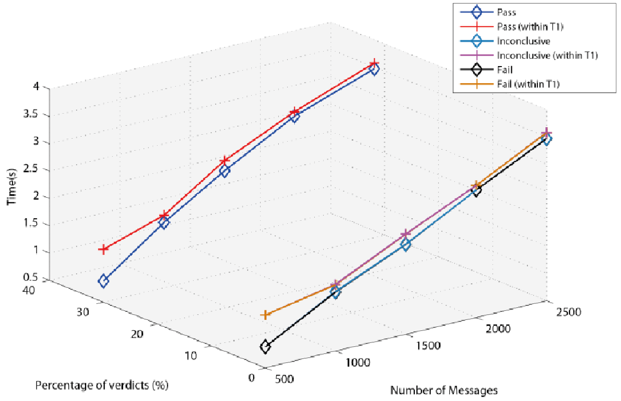
Per-Fail:
Indicates the messages received by the SUT exceeded the expected time T.

Con-Fail:
Indicates the messages violate the data portion requirements.

Inconclusive:
No definite verdict can be returned.

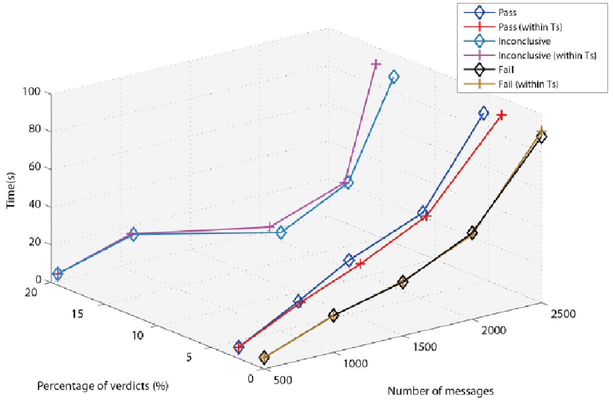
SIP testing

For each INVITE request there should be a 2xx response, received within 16s



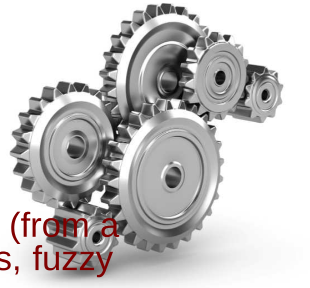
Tra ce	No. of messages	Pass	Con -Fail	Per-Fail	Inconclusive
1	500	150	0	0	1
2	1000	318	0	0	0
3	1500	504	0	0	1
4	2000	674	0	0	0
5	2500	798	0	0	1

For each REGISTER request there should be a 2xx response, received within 16s



Tra ce	No. of messages	Pass	Con -Fail	Per-Fail	Inconclusive
1	500	12	0	97	0
2	1000	33	0	188	1
3	1500	76	1	172	2
4	2000	93	0	233	2
5	2500	136	0	347	1

To conclude (if we can…)



Passive testing – in short

- Relevant when:
  - Access to the interface unavailable,
  - Unreachable component,
  - SUT cannot be interrupted,
  - SUT cannot be stimulated, …
- Several syntax/semantics and algorithms to check protocol requirements on traces
  - High complexity ➔ results for very complex formulas may be provided after *days*!
  - Still offline ➔ tradeoff between offline, storage and trace pruning
  - Many INC verdicts ➔ analysis/controllability of the environment needed.
  - Study of eventual false positives/negatives!
  - …

Active testing – in short

- Efficient when:
  - test suites generated automatically (from a formal model, learning mechanisms, fuzzy testing techniques,…),
  - Testing architecture may easily evolve,
  - Disturbing the SUT is not an issue, …
- Several issues while applying the test suites and analyzing the obtained verdicts
  - When to stop when the TCP is awaiting?
  - What verdict when no answers?
  - What about the distributed systems with multiple PCOs ?! ➔ clock synchronism, verdicts correlation, controllability, …)
  - Component testing ➔ some components are not directly stimulated ☹
  - What about the formal models in the industry?!
  - …

REFERENCES

- **Materials from L.Logrippo & J.Tretmans. Thanks a lot!**

- [Hierons2009] R.Hierons et al., Using formal specications to support testing. ACM Computing Surveys, page 41(2):176, 2009.
- [ISO 9126] ISO/IEC 9126 Software engineering — Product quality
- [Bauer2011] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verication for LTL and TLTL. ACM Transactions on Software Engineering and Methodology, 20(4):14, 2011.
- [Lalanne2013] Felipe Lalanne and Stephane Maag. A formal data-centric approach for passive testing of communication protocols. IEEE / ACM Transactions on Networking, 21(3):788{801, 2013
- [Andres2012] Cesar Andres, Mercedes G Merayo, and Manuel Nu~nez. Formal passive testing of timed systems : theory and tools. Software Testing, Verication and Reliability, 22(6):365{405, 2012.
- [Barna2011] C. Barna, M. Litoiu, and H. Ghanbari. Model-based performance testing: NIER track. In Proceedings of 33rd International Conference on Software Engineering (ICSE), pages 872 {875, May 2011
- [Pakin2007] Scott Pakin. The design and implementation of a domain-specic language for network performance testing. IEEE Transactions on Parallel Distributed System, 18(10):1436{1449, 2007.
- [Dumitrescu2004] Catalin Dumitrescu, Ioan Raicu, Matei Ripeanu, and Ian Foster. Diperf: An automated distributed performance testing framework. In Proceedings of 5th International Workshop in Grid Computing, pages 289{296. IEEE Computer Society, 2004.
- [Bayse2005] E. Bayse, A. Cavalli, M. Nunez, and F. Zaidi. A passive testing approach based on invariants: application to the wap. Computer Networks, pages 48(2):247{266, 2005.
- [Dorofeeva2010] Rita Dorofeeva, Khaled El-Fakih, Stephane Maag, Ana R. Cavalli, Nina Yevtushenko, FSM-based Conformance Testing Methods: a Survey annotated with Experimental Evaluation, in Elsevier Information and Software Technology, Vol. 52, p.1286-1297, 2010.

thank you