Passive Approach for Robustness Testing of Communication Protocols

Eliane Martins, Anderson Morais
IC- Unicamp

Ana Cavalli
Institut Telecom & Management Sud-Paris

Topics

- Robustness Testing
  - Why
  - What
- Proposed Approaches
- An Hybrid Approach
- Results

Why robustness testing?

- Testing software to ensure that the functional requirements were met … → Conformance testing

  … is necessary but not enough

☞ How does the system behave in presence of
  - erroneous or unexpected user inputs?
  - internal or external failures?
  - stressful environmental conditions?

→ Robustness testing

Robustness

- Definition

  "*the degree to which a software system or component can function* ***correctly*** *in the presence of* ***invalid inputs*** *or stressful environmental conditions*."

IEEE Std 610.12-1990 - Glossary of Software Engineering Terminology

Robustness testing

- Definition [CW03] :

  "*aimed to determine whether a system or component can have an acceptable behavior in the presence of* ***faults*** *or stressful environmental conditions*"

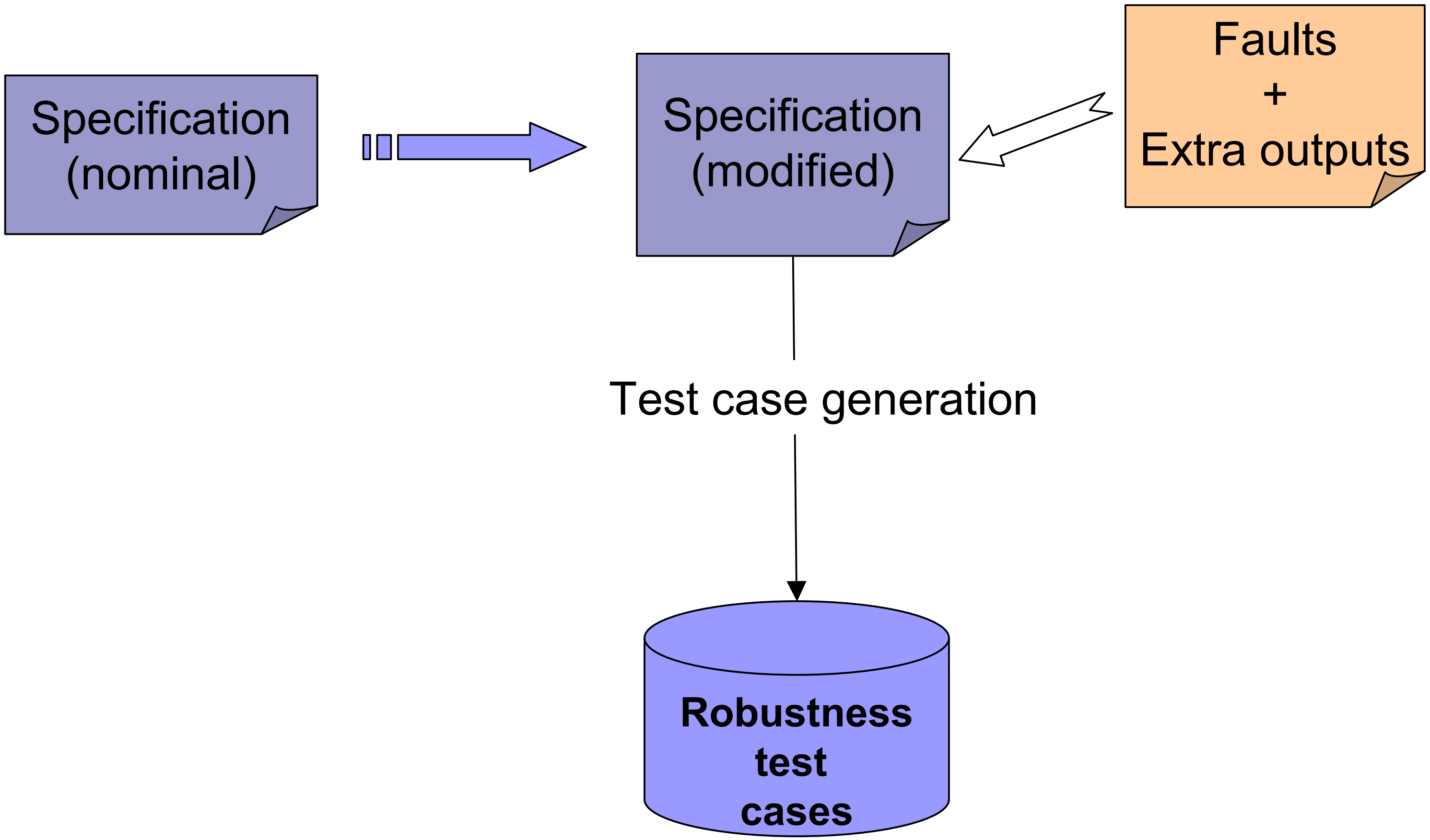
Robustness testing approaches

- Ad-hoc approaches
  - Hard to automate
- Based on models
- Based on fault injection

Model-based approaches

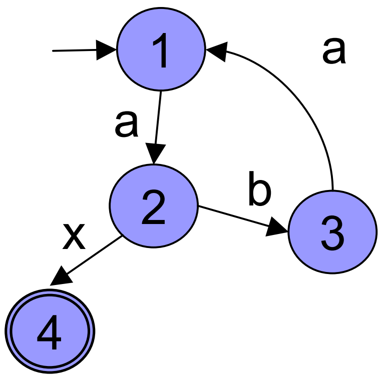
- Formalization of robustness testing is inspired on that of conformance testing
  - Conformance testing:
    - Goal: determining whether an implementation conforms to its specification
  - The specification is represented by a (behavior) model from which:
    - Test cases can be derived
    - Observed results can be analyzed

Robustness test cases generation

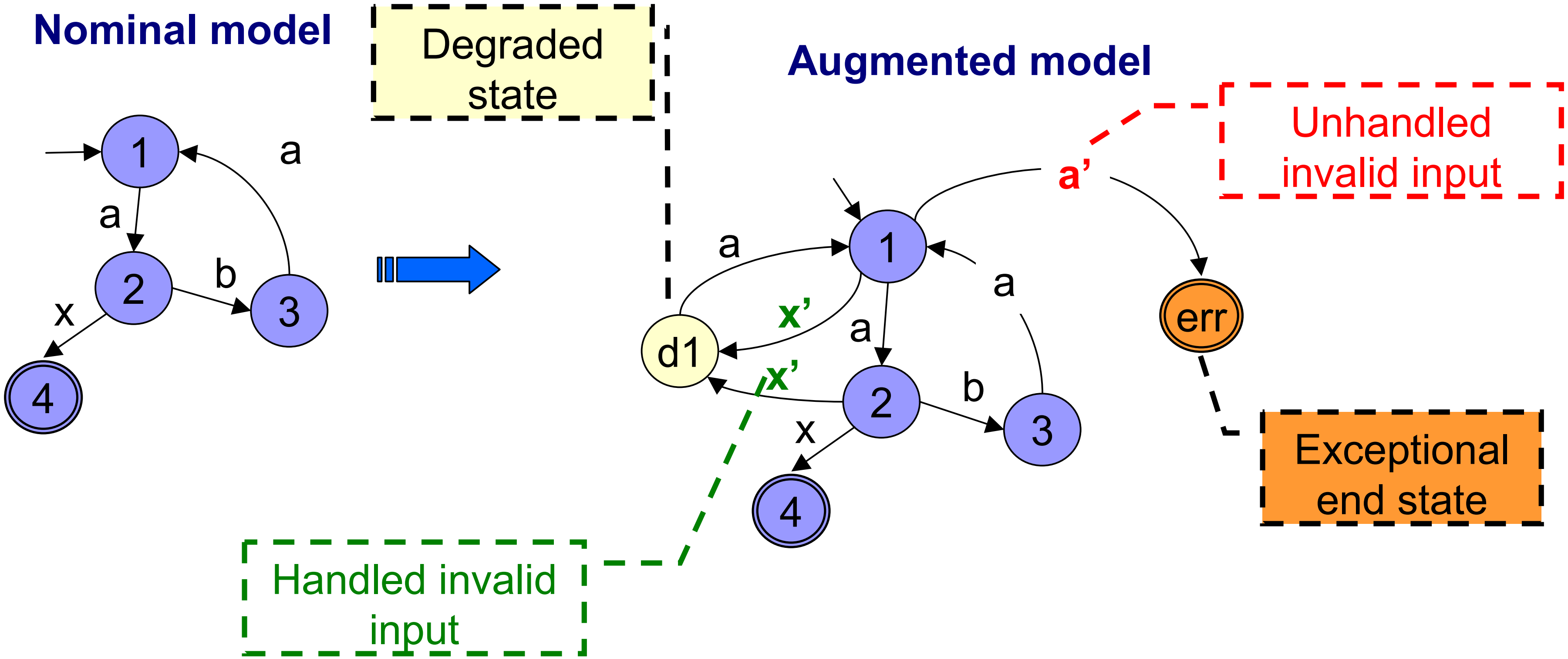


Illustrative example

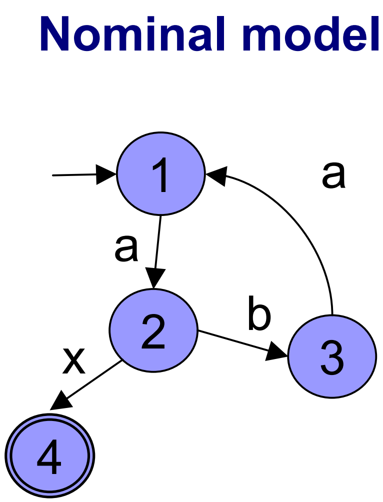
Nominal model

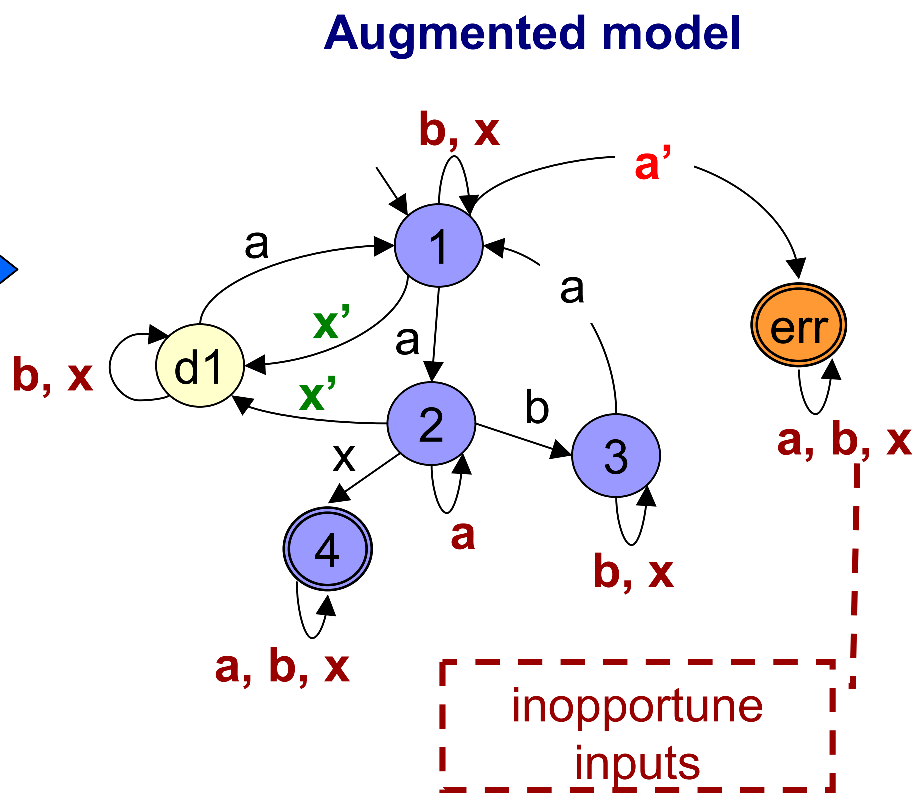


Illustrative example



Illustrative example





Difficulties with the model-based approaches

- Model size is too big for use
  - Need to carefully define test objectives
- Tester has limited control of faults
  - Faults to consider may depend on the application domain and on the system architecture
  - Environmental (context) faults (memory, processor, communication channel, device drivers) are not considered
- System behavior in the presence of faults cannot always be completely specified

Fault injection

- Definition

  Deliberate introduction of faults into a system to observe its behavior

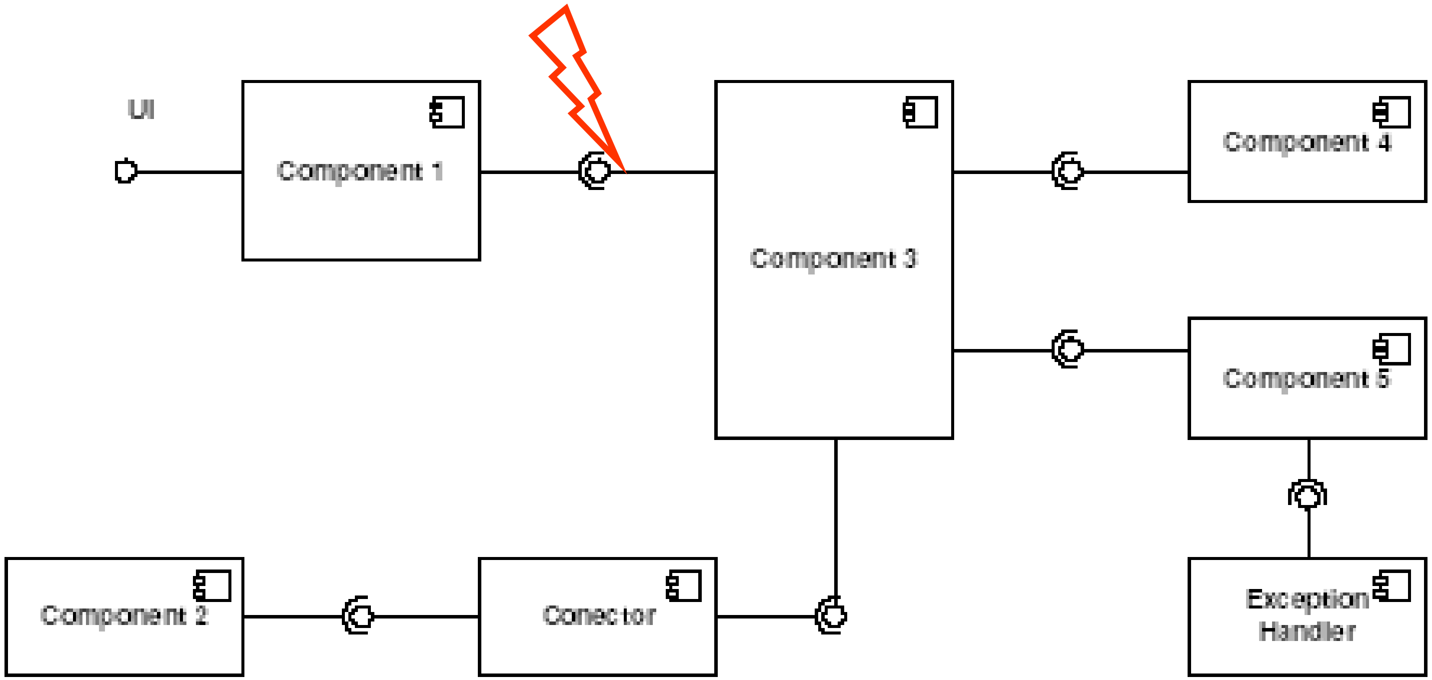
- Applicability
  - To verify whether the error detection and recovery mechanisms behave as expected.
  - To evaluate dependability measures such as reliability for a giving mission time, availability, performance degradation due to fault handling.
  - To understand the effects of real faults.

Fault injection approaches

- Faults can be injected:
  - Into a model ⇒ Simulation-based fault injection
  - Into a prototype or final system:
    - Hardware level ⇒ Hw-implemented fault injection (HWIFI)
    - Software level ⇒ Sw-implemented fault injection (SWIFI)

Robustness testing and fault injection

- Interface fault injection:
  - affects functions input/output parameters or protocol messages fields
  - Invalid values produced according to input/output domains or formats
- Some approaches and tools:
  - Ballista/Piranha, Mafalda, Fuzz, Riddle, PROTOS, Jaca


UI
Component 1
Component 3
Component 4
Component 5
Component 2
Conector
Exception Handler

Limitations of interface fault injection approaches

- Oracle is generally not based on the specification
  - "golden run" or reference implementation
  - Crash or not crash
- Knowledge about the system structure or behavior is not frequent
  - Exceptions: Avresky et al 1992; Echtle &Chen 1991; Sinha &Suri 1998; Loki 2000

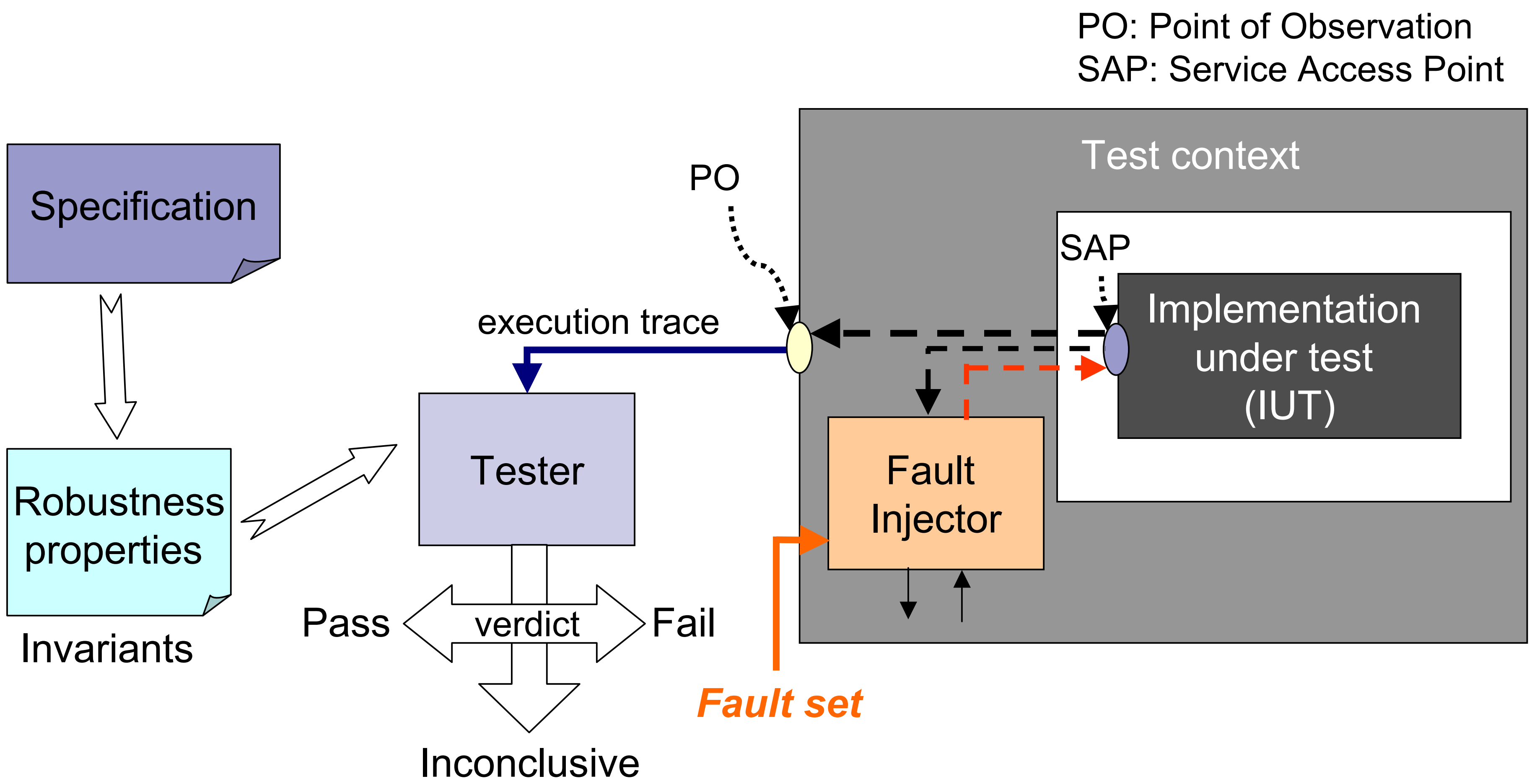
Proposed approach

- Hybrid approach combining
  - Fault injection
  - Passive testing

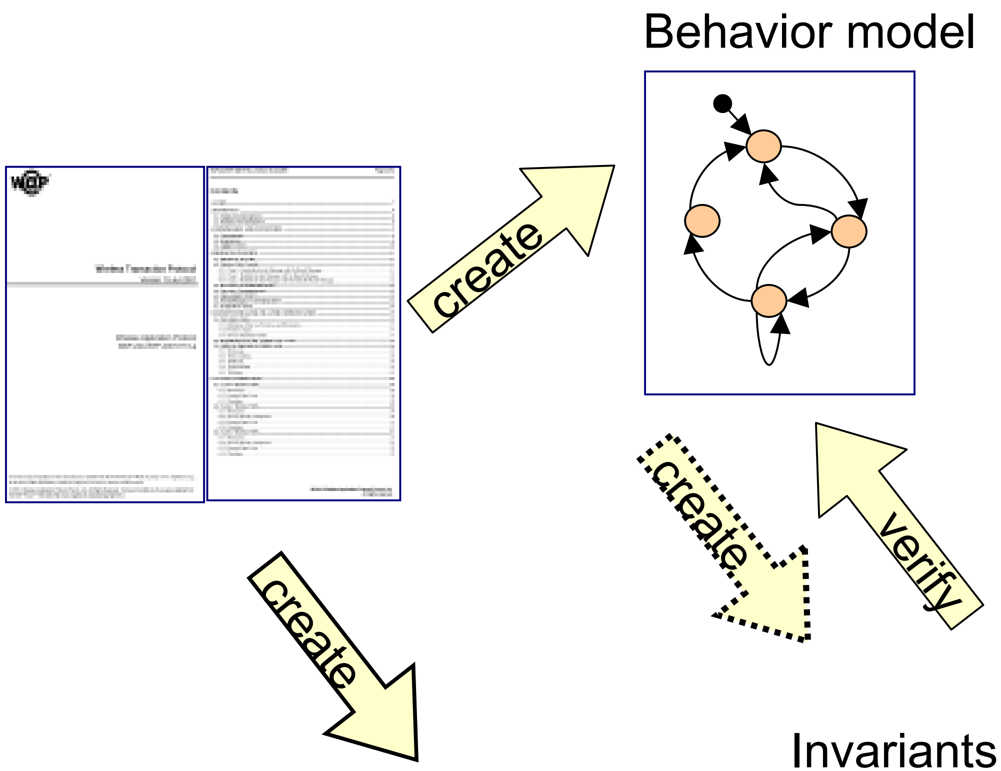
Passive testing approaches

- Based on trace acceptation
  - determines whether the observed trace satisfies the specification model
- Based on invariants

Abstract test architecture



Invariants analysis approach



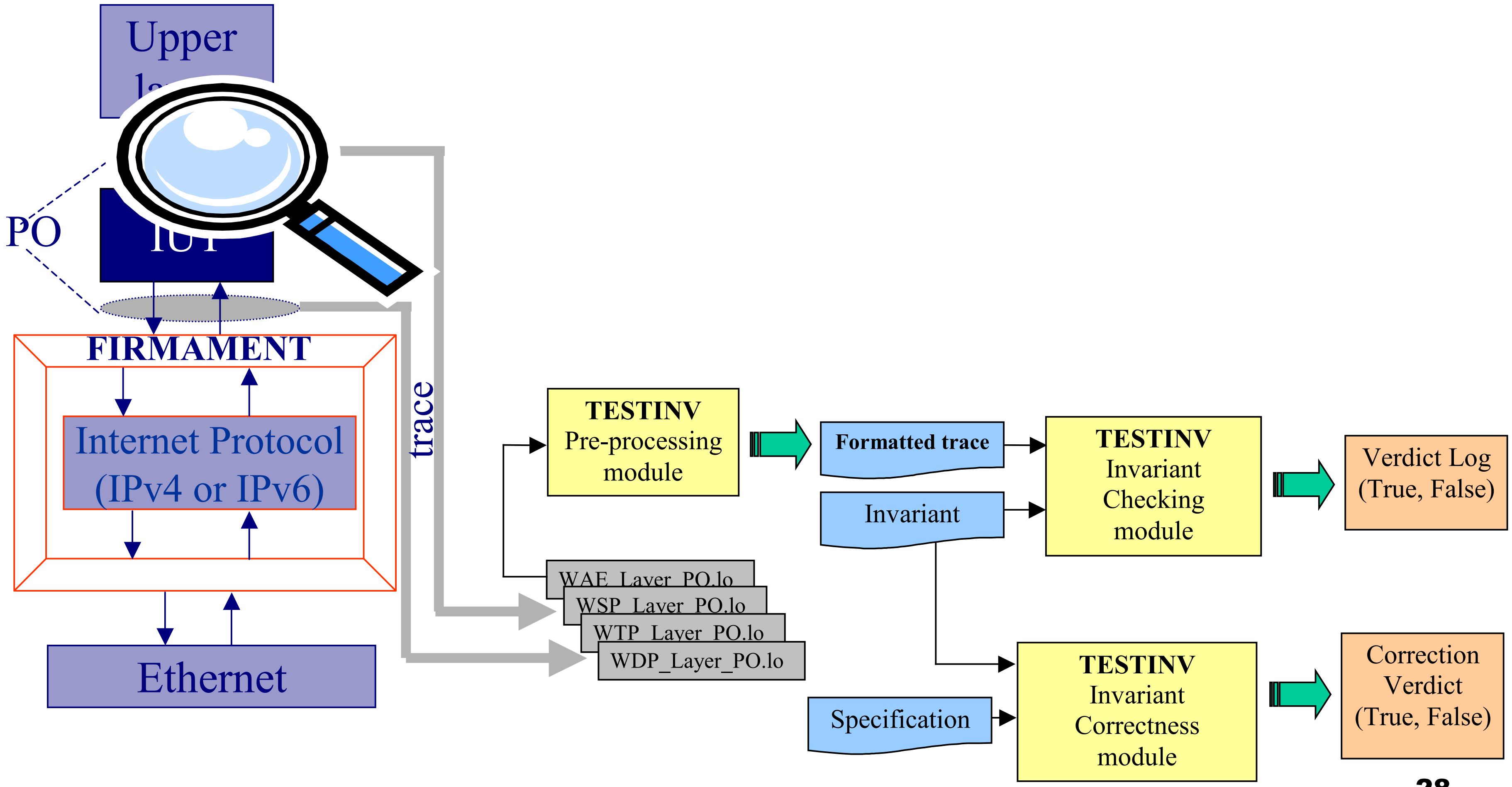
Invariants = properties of interest

Invariants in the form of regular expressions

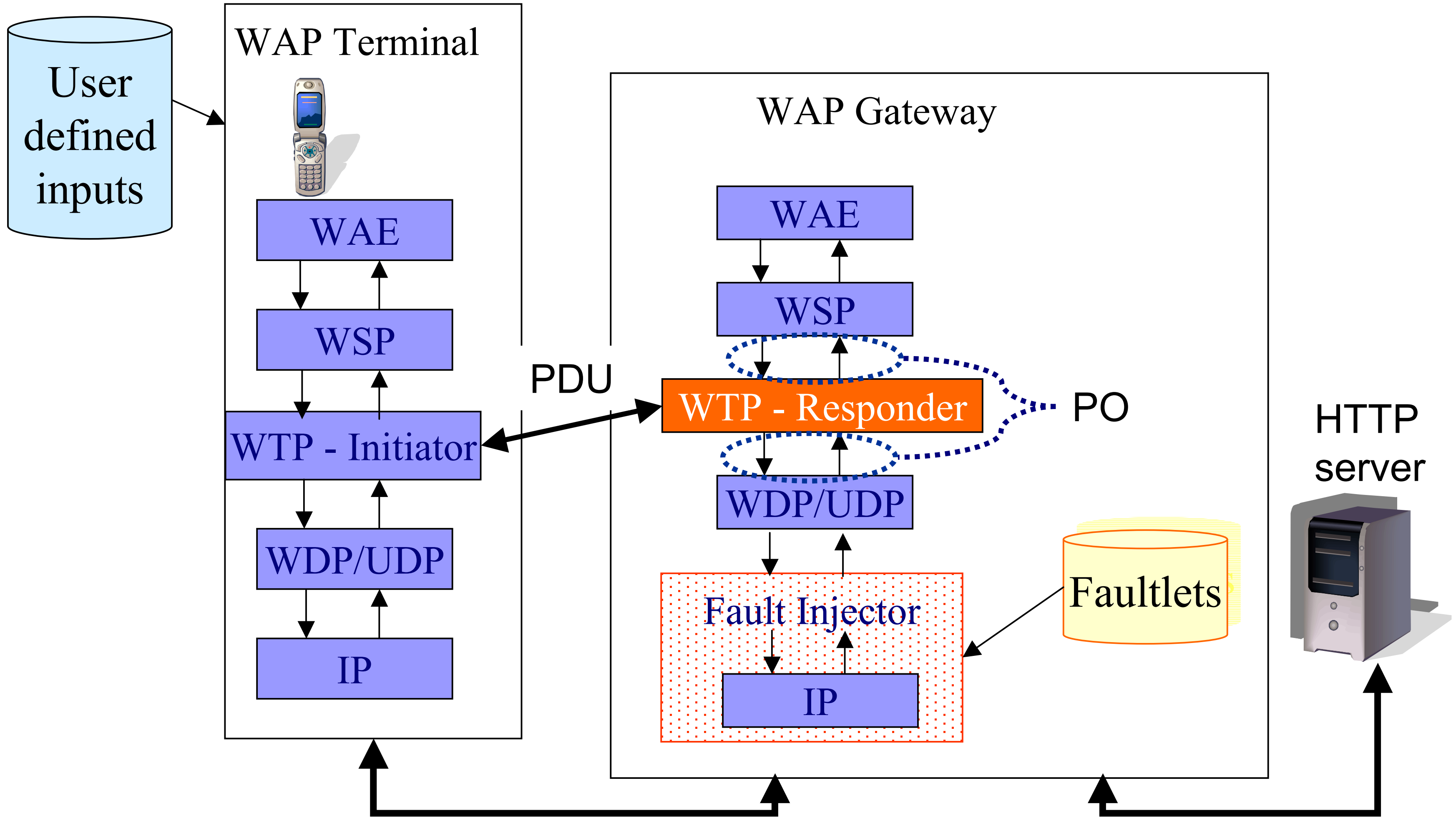
*I1 = RcvInvoke(TID = N)/?, *, TR-Invoke.res/{Ack (TID = N)}*

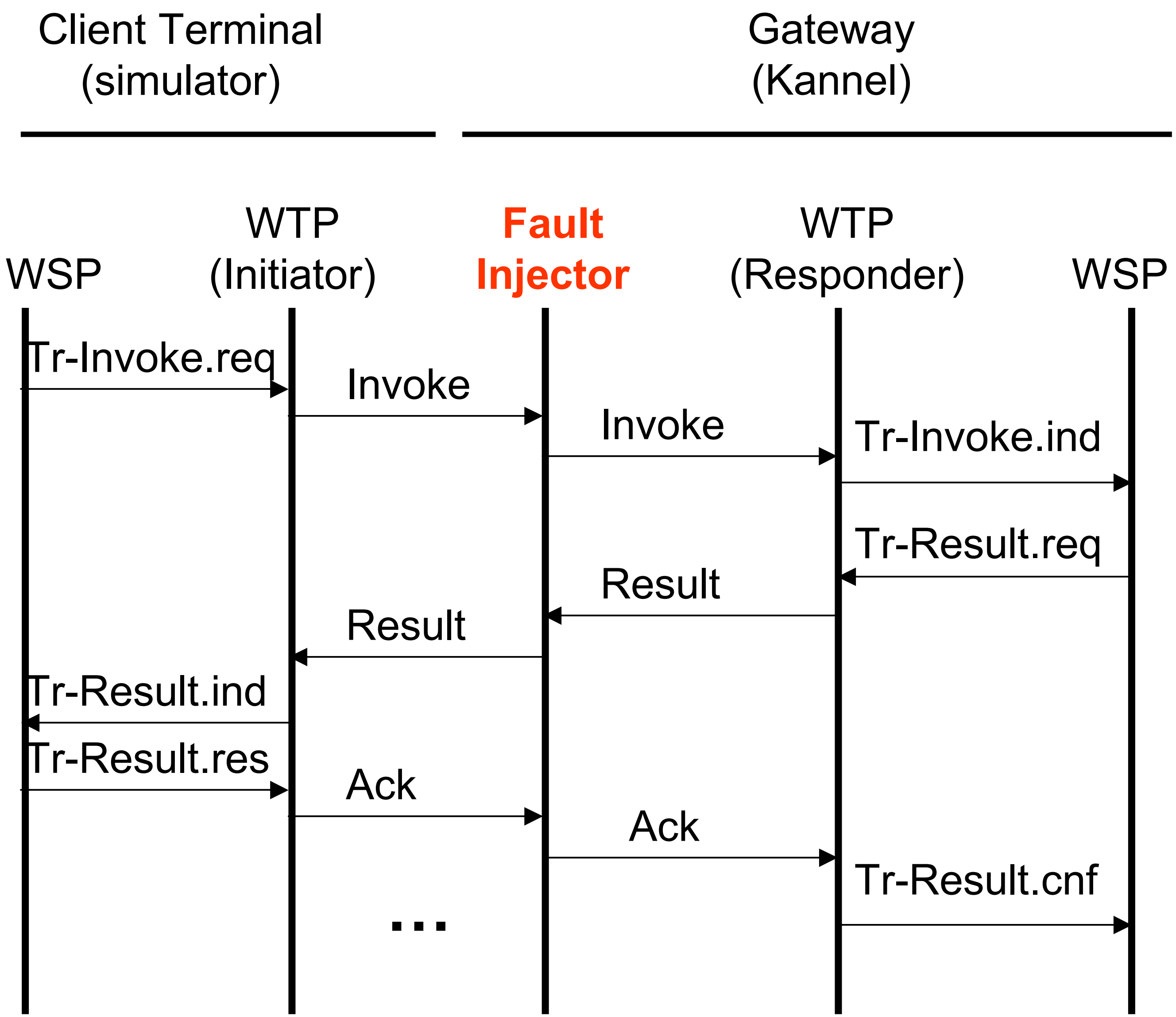
*I2= RcvInvoke(TID = N) / TR.Invoke.ind, *, TR-Invoke.res /* {Ack (TID = N) }

Test configuration

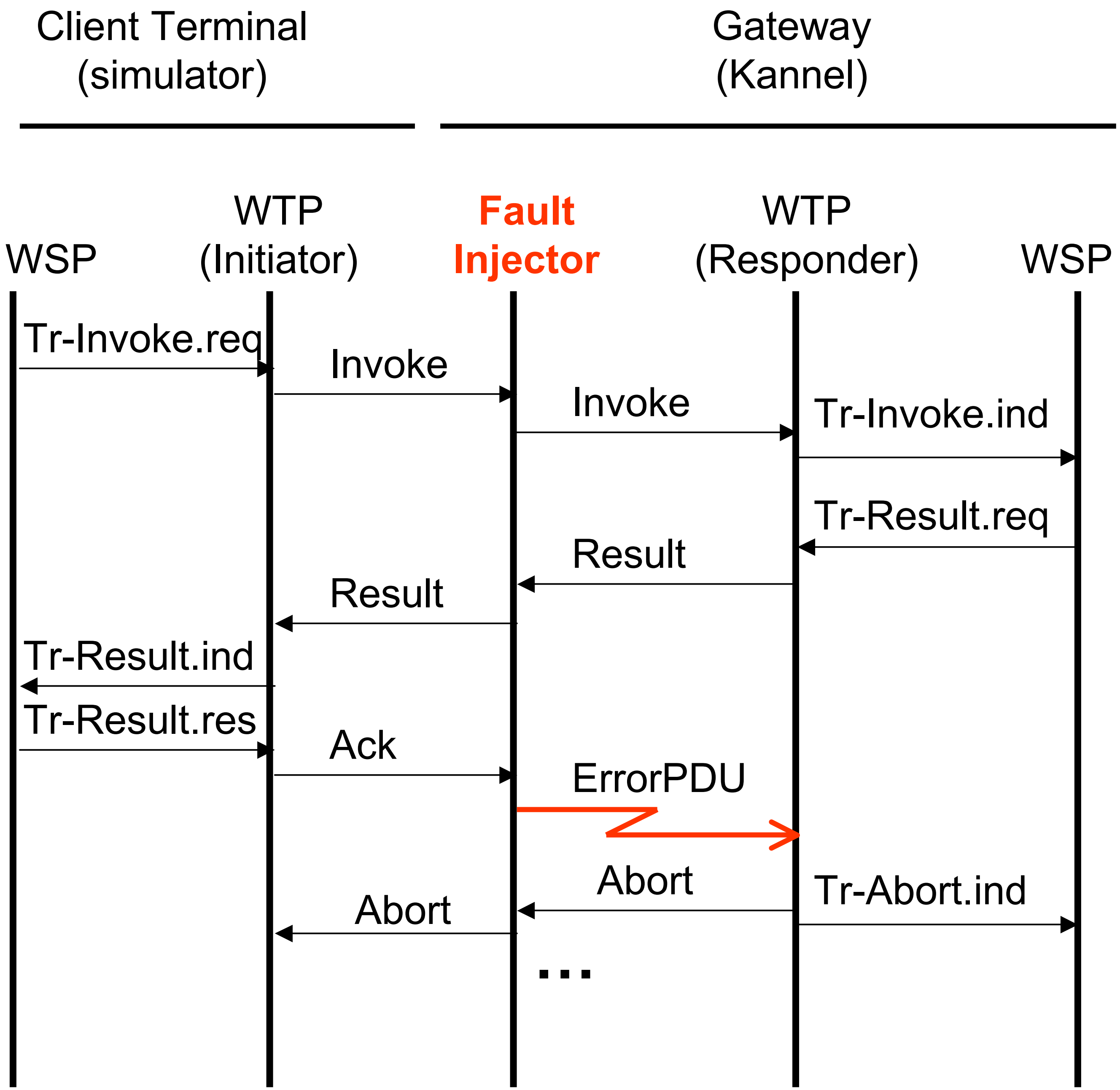


The WAP stack





No faults injected



Fault injected

An experiment that failed

Experiments	Runs	Result seen by Nokia browser	Observ.
E1- Test packet corruption. Change PDU Type	R1- Ack (0x3) →Invoke (0x1)	Requested page	
	R2- Ack (0x3) →Invalid (0x00)	Requested page	
	R3- Ack (0x3) →Result (0x2)	Error message: “Server aborted connection”	
	R4- Ack (0x3) →Invalid (0xff)	—	Browser blocked

Example of observed trace with failure (1)

```
2007-10-11 01:21:50 [6] INFO: (ORIGINATE STATE: LISTEN ; NEXT STATE:
INVOKE_RESP_WAIT)
2007-10-11 01:21:50 [6] INFO: FROM WDP: Event Name: RcvInvoke(TID=78, class=2,
Uack=1, TIDNew=0, RID=0)
2007-10-11 01:21:50 [6] INFO: TO WSP: Primitive Name: TR-Invoke.ind(class=2)
2007-10-11 01:21:50 [6] INFO:

2007-10-11 01:21:50 [6] INFO: (ORIGINATE STATE: INVOKE_RESP_WAIT ; NEXT STATE:
RESULT_WAIT)
2007-10-11 01:21:50 [6] INFO: FROM WSP: Primitive Name: TR-Invoke.res
2007-10-11 01:21:50 [6] INFO:

2007-10-11 01:21:50 [6] INFO: (ORIGINATE STATE: RESULT_WAIT ; NEXT STATE:
RESULT_RESP_WAIT)
2007-10-11 01:21:50 [6] INFO: FROM WSP: Primitive Name: TR-Result.req
2007-10-11 01:21:50 [6] INFO: TO WDP: PDU Name: Result(TID=78, RID=0)
2007-10-11 01:21:50 [6] INFO:

2007-10-11 01:21:50 [0] ERROR: pdu unpacking returned NULL
2007-10-11 01:21:50 [6] INFO: TO WDP: PDU Name: Abort(TID=78, abort-type=0,
abort-reason=1)
2007-10-11 01:21:50 [6] INFO:

2007-10-11 01:21:50 [6] INFO: TO WSP: Primitive Name: TR-Abort.ind(abort-
reason=1)
2007-10-11 01:21:50 [6] INFO:
2007-10-11 01:24:32 [0] ERROR: SIGINT received, let's die.
```

Abort PDU

TR-Abort

Wapbox hangs

Run aborted by the user

Wapbox log:

2007-10-11 01:21:50 [6] DEBUG: WTP: **Destroying WTPRespMachine 0x820def0 (23)**

2007-10-11 01:21:50 [6] DEBUG: WTP: **Created WTPRespMachine 0x8209c90 (24)**

2007-10-11 01:21:50 [6] DEBUG: WTP: resp_machine 24, state LISTEN, event RcvInvoke.

…

2007-10-11 01:21:50 [6] DEBUG: WTP: **Destroying WTPRespMachine 0x8209c90 (24)**

. . .

2007-10-11 01:21:50 [1] DEBUG: WSP: machine 0x81e90e8, state CONNECTING_2, event TR-Abort.ind

2007-10-11 01:21:50 [1] DEBUG: ----------1)handle_session_event

2007-10-11 01:21:50 [1] DEBUG: WSP 2: New state NULL_SESSION

2007-10-11 01:21:50 [1] DEBUG: **Destroying WSPMachine 0x81e90e8**

2007-10-11 01:24:32 [0] ERROR: SIGINT received, let's die.

Another experiment that failed

- Experiment 5: wrong packet size.
  - Run 2: change PDU size to small value (=2)
    - Failure: no Abort message generated as was expected!

```
2007-10-11 03:53:21 [6] INFO: (ORIGINATE STATE: LISTEN ; NEXT STATE:
INVOKE_RESP_WAIT)
2007-10-11 03:53:21 [6] INFO: FROM WDP: Event Name: RcvInvoke(TID=306,
class=2, Uack=1, TIDNew=0, RID=0)
2007-10-11 03:53:21 [6] INFO: TO WSP: Primitive Name: TR-Invoke.ind(class=2)
2007-10-11 03:53:21 [6] INFO:

2007-10-11 03:53:21 [6] INFO: (ORIGINATE STATE: INVOKE_RESP_WAIT ; NEXT STATE:
RESULT_WAIT)
2007-10-11 03:53:21 [6] INFO: FROM WSP: Primitive Name: TR-Invoke.res
2007-10-11 03:53:21 [6] INFO:

2007-10-11 03:53:21 [6] INFO: (ORIGINATE STATE: RESULT_WAIT ; NEXT STATE:
RESULT_RESP_WAIT)
2007-10-11 03:53:21 [6] INFO: FROM WSP: Primitive Name: TR-Result.req
2007-10-11 03:53:21 [6] INFO: TO WDP: PDU Name: Result(TID=306, RID=0)
2007-10-11 03:53:21 [6] INFO:

2007-10-11 03:53:21 [0 PANIC wap/wap_events.c:161: wap_event_assert:
Assertion `event != NULL' failed.
```

Crash of the wapbox

Wapbox log:

```
2007-10-11 03:53:21 [6] DEBUG: WTP 1: New state RESULT_RESP_WAIT
...
2007-10-11 03:53:21 [0] DEBUG: A too short PDU received
2007-10-11 03:53:21 [0] DEBUG: Dumping WAPEvent 0x820bad0
2007-10-11 03:53:21 [0] DEBUG:   type = T-DUnitdata.ind
2007-10-11 03:53:21 [0] DEBUG: WAPAddrTuple 0x820bb40 =
   <127.0.1.1:32787> - <0.0.0.0:9201>
2007-10-11 03:53:21 [0] DEBUG: user_data =
2007-10-11 03:53:21 [0] DEBUG:  Octet string at 0x820bd38:
2007-10-11 03:53:21 [0] DEBUG:    len:  1
2007-10-11 03:53:21 [0] DEBUG:    size: 2
2007-10-11 03:53:21 [0] DEBUG:    immutable: 0
2007-10-11 03:53:21 [0] DEBUG:    data: 18                          .
2007-10-11 03:53:21 [0] DEBUG:  Octet string dump ends.
2007-10-11 03:53:21 [0] DEBUG: WAPEvent dump ends.
2007-10-11 03:53:21 [0] PANIC: wap/wap_events.c:161: wap_event_assert:
   Assertion `event != NULL' failed.
```

Simple invariants used

S1. RcvInvoke/TR-Invoke.ind,*,TR-Result.req/{Result}

S2. RcvInvoke/TR-Invoke.ind,*,RcvAck/{TR-Result.cnf, NULL}

S3. RcvErrorPDU/{Abort, TR-Abort.ind}

S4. ?/?, *, RcvAbort/{TR-Abort.ind}

S5. ?/?, *, TimerTO_R/{Result,TR-Abort.ind}

S6. ?/?, *, TimerTO_A/{Ack,TR-Abort.ind, NULL}

S7. ?/?, *, TR-Abort.req/{Abort}

S8. RcvInvoke/Ack, *, RcvAck/{TR.Invoke.ind}

S9. RcvInvoke/Ack, *, RcvInvoke/{Ack, NULL}

S10. ?/?, *, NULL/{CRASH, HANG}

Alphabet of the machine: ≈ 20 WTP events + Hang + Crash + NULL

Discussion about observed results

- Only control flow was considered in the invariant analysis
- Observed anomalous behavior:
  - Lack of resources created new sources of failures:
    - IUT did not tolerate some OS exceptions
  - Lack of information in the specification
    - Ex.: Initiator continues to send requests for new transactions even when the Responder keeps retransmitting the same results

Conclusions

- Hybrid approach for robustness testing, combining formal and fault injection techniques:
  - Fault injection:
    - Allows better coverage of environment faults than in traditional testing
  - Passive testing:
    - Allows more precise result analysis than simply observing crash or hangs, as is usual in FI
  - Possibility to test an IUT in its context → useful in later stages of system testing or even in the field

Current work

- Approach is in use for testing robustness against attacks:
  - Cryptographic protocol testing
  - Instead of communication faults, attacks are injected
  - Attack scenarios derived from real successful attacks reported in the literature
  - Attacker is implemented by a fault injector
  - Goal: reveal vulnerabilities in the protocol implementation
  - Invariants used to represent security properties

Future works

- Algorithm for the transformation of attack scenarios into executable scenarios for the fault injector (Attacker)
- Application of the approach to a case study
- Use of sequence alignment algorithms for results analysis

Thanks!

Email: eliane@ic.unicamp.br
anderson.morais@ic.unicamp.br

References (1)

Report about model-based robustness testing:

Castanet R., Waeselynk H., “Techniques avancées de test de systèmes complexes: test de robustesse“, report CNRS-AS23, 2003.

Ballista

Koopman, P., Siewiorek, D, DeVale, K., DeVale, J., Fernsler, K., Guttendorf, D., Kropp, N., Pan, J., Shelton, C., Shi, Y.: “Ballista Project : COTS Software Robustness Testing.” Carnegie Mellon University.

Piranha

J.L.Griffin. “Testing protocol implementation robustness”, Proc. 29th. Annual International Symposium on Fault-Tolerant Computing (FTCS), 15-18 June 1999, Madison, Wisconsin

Mafalda

J. Arlat, J.-C. Fabre, M. Rodríguez and F. Salles, “Dependability of COTS Microkernel-Based Systems”, IEEE Trans on Computers, 51 (2), pp.138-163, February 2002.

Fuzz

Miller B. P., et. al. (1995) Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services. Networked Computer Science Technical Reports Library CS-TR-95-1268, UW Madison Computer Sciences Department, April 1995

PROTOS

R. Kaksonen, M.Laakso, A.Takanen. **Vulnerability Analysis of Software through Syntax Testing. http://www.ee.oulu.fi/research/ouspg/protos/analysis/WP2000-robustness/**

References (2)

RIDDLE

A. Ghosh, V. Shah, and M. Schmid. An approach for analyzing the Robustness of Windows NT Software. In Proceedings of the 21st National Information Systems Security Conference, pages 383–391, Crystal City, VA, 1998.

Jaca

Naaliel Mendes, Regina Moraes, Eliane Martins, Henrique Madeira. “Jaca Tool Improvements for Speeding Up Fault Injection Campaigns”. 13ª. Tools Session. 20º. Brazilian Symposium on Software Engineering (SBES), Florianópolis, Oct./2006

About invariant testing:

E. Bayse, A. Cavalli, M. Nunez and F. Zaidi, “A Passive Testing Approach based on Invariants: Application to the WAP”, Computer Networks, 48, pp247-266, 2005

Introduction to fault injection:

Hsueh, Mei-Chen; Tsai, Timothy; Iyer, Ravishankar. “Fault Injection Techniques and Tools”. IEEE Computer, Abril/1997

Hybrid approaches for active testing:

K. Echtle and Y. Chen, “Evaluation of deterministic fault injection for fault-tolerant protocol testing,” in Proc. 21st Int. Symp. Fault-Tolerant Computing (FTCS-21), IEEE, Montréal, Québec, Canada, June 1991, pp. 418-425.

D. Avresky, J. Arlat, J.-C. Laprie, and Y. Crouzet, “Fault injection for the formal testing of fault tolerance,” in Proc. 22nd Int. Symp. Fault-Tolerant Computing (FTCS-22), IEEE, Boston, MA, July 1992, pp. 345-354.

N.Suri; P.Sinha. “On the Use of Formal Techniques for Validation”. Proc. of FTCS-28, pp. 390--399, 1998.