

Building Dependable COTS Microkernel-based Systems using MAFALDA

*Jean-Charles Fabre, Manuel Rodríguez, Jean Arlat, Frédéric Salles
and Jean-Michel Sizun*

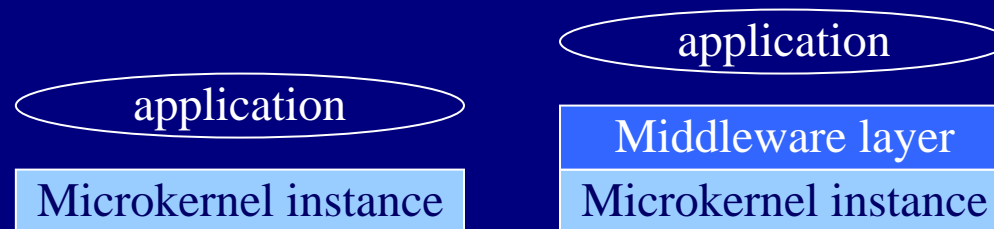


LAAS-CNRS
Toulouse, France

PRDC-2000, UCLA, Los Angeles, CA, USA — 18-20 December 2000

Problem statement

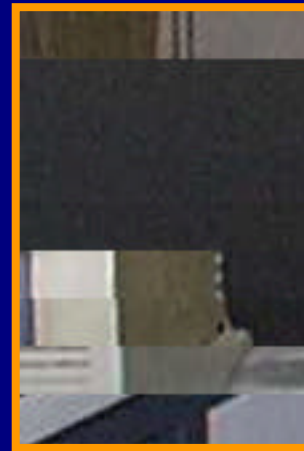
- **Building executive supports for dependable systems, two options:**
 - Development from scratch is complex & expensive
 - Use of commercial components is questionable
- **Main tendency for embedded systems**
 - Use of COTS componentized microkernels
 - Define a specific instance for the application
 - System development : two options



Outline

- **The objective of MAFALDA**
 - Failure mode analysis
 - Development of fault containment wrappers
- **Description of the tool**
 - Organization
 - Type of measurements
- **MAFALDA in action**
 - Campaign definition
 - Conducting experiments
- **Lessons learnt**

*MAFALDA
Microkernel
Assessment by
Fault injection
AnaLysis and
Design
Aid*



*Rack of target
machines*



*Host
Machine
controlling the
experiments*

Objectives of MAFALDA

*MAFALDA
Microkernel
Assessment by
Fault injection
AnaLysis and
Design
Aid*

- **Characterization by SWIFI**

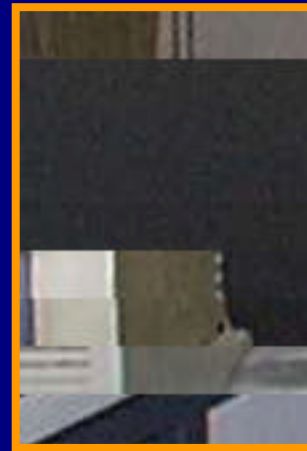
(S/W Implemented Fault Injection)

- Identification of failure modes
- Evaluation of error detection coverage
- Identification of propagation channels
- Assessment of interface robustness

- **Wrapping framework**

- Definition of formal wrappers
- Definition of a reflective implementation framework
- Application to both white-box & black-box candidates

- **Evaluation of the wrapped microkernel instance**



Rack of target machines



*Host Machine
controlling the
experiments*

Characterization of the failure modes

Evaluation

- interface robustness
- error detection mechanisms

Injection targets

- kernel-call parameters
- microkernel components

Fault model

- corruption of input data
- corruption of microkernel code & data segments

Fault types

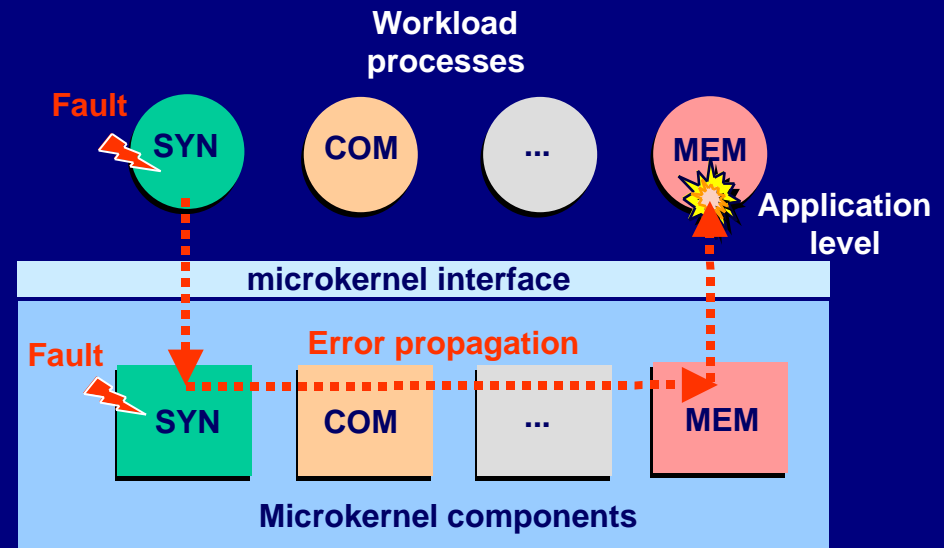
- bit-flip
- random selection

Observation

- behavior of a functional component
- error propagation between components

Results

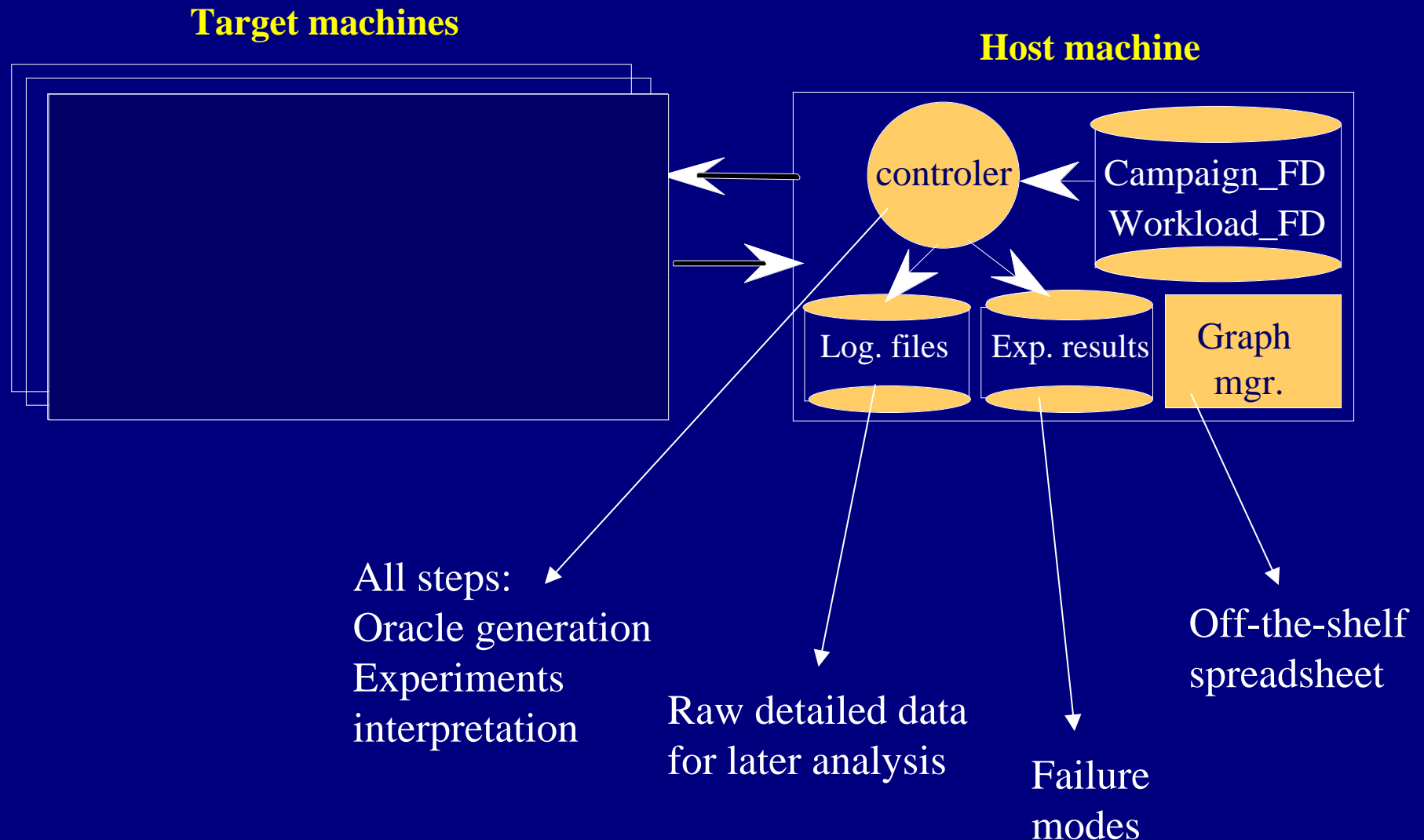
- statistics
- raw data (*a posteriori* in-deep analysis)



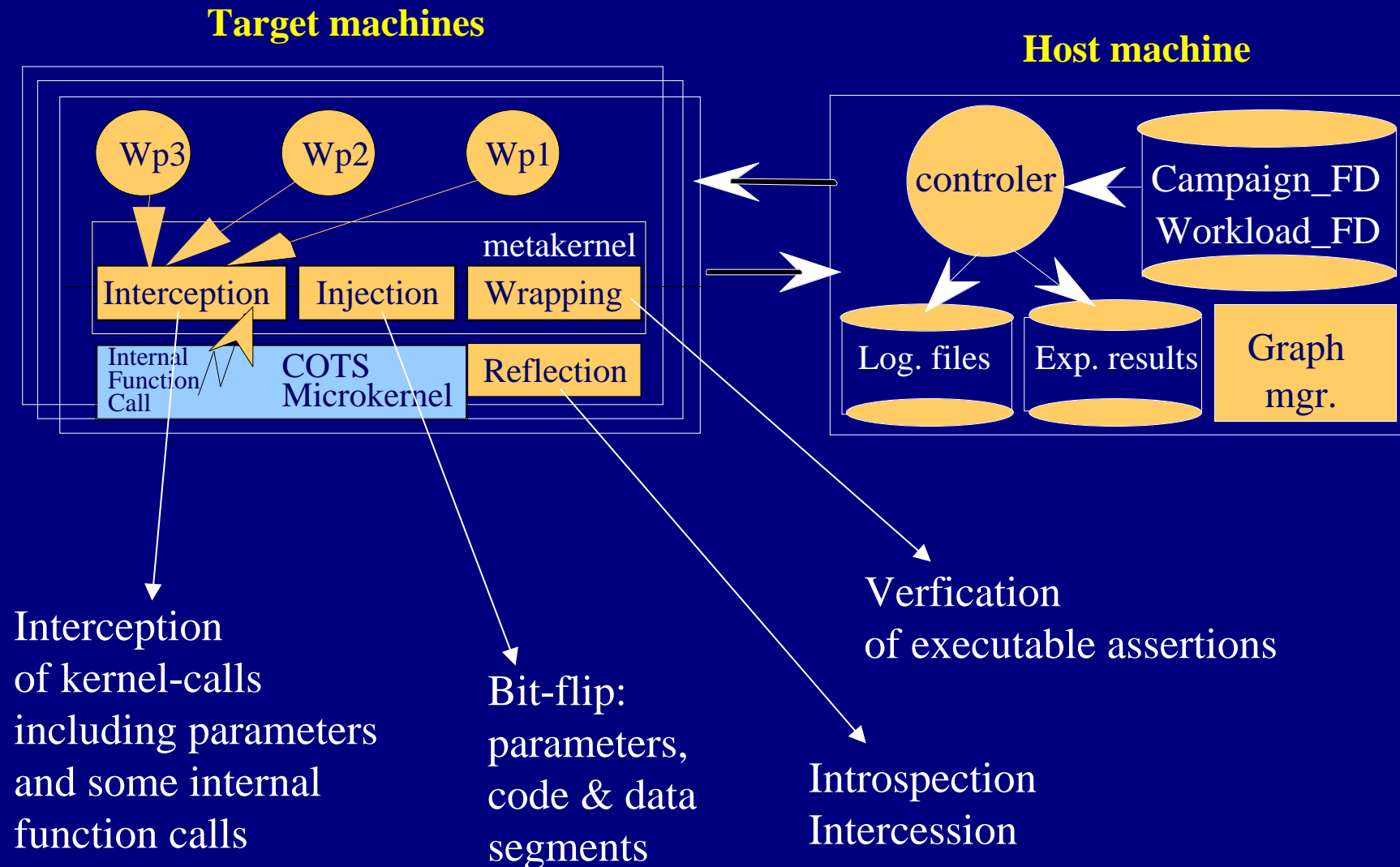
Fault containment wrappers

- **Principles and basic techniques**
 - Encapsulation of weak components
 - Modeling microkernel functional classes
 - On-line verification of expected properties
- **Implementation of generic wrappers**
 - Principle
 - Verification of executable assertions
 - Verification of formal expressions (model-checking)
 - Implementation based on the notion of reflective component
 - Interception of system calls and internal events
 - Some internal information is made observable from outside
 - Microkernel + observation/control = **reflective microkernel**

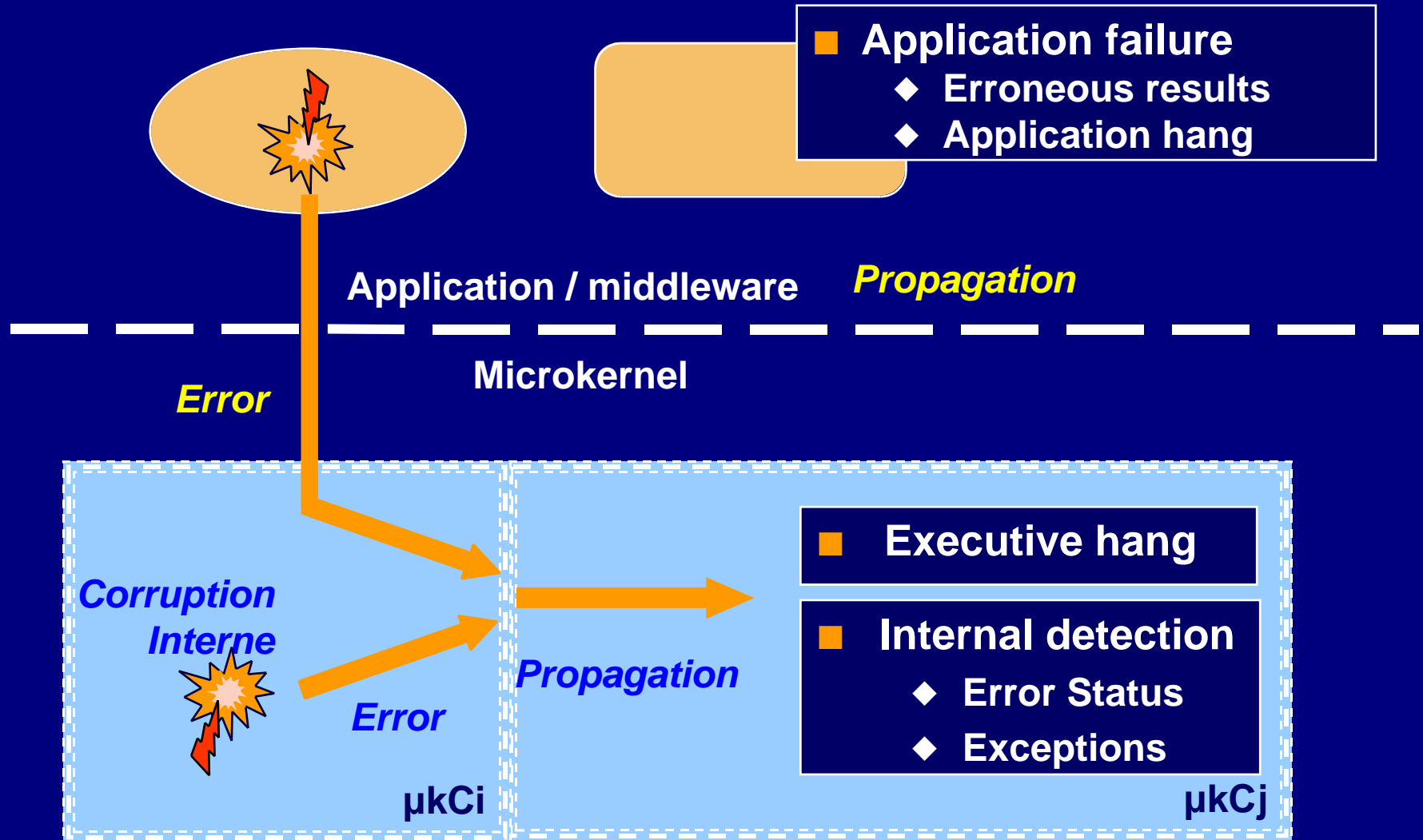
Description of the tool



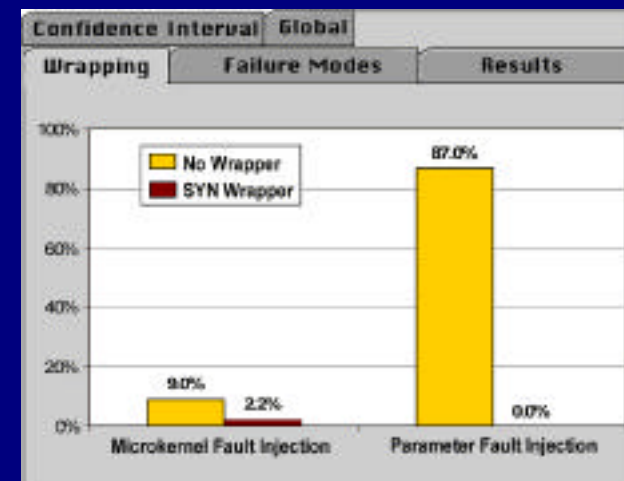
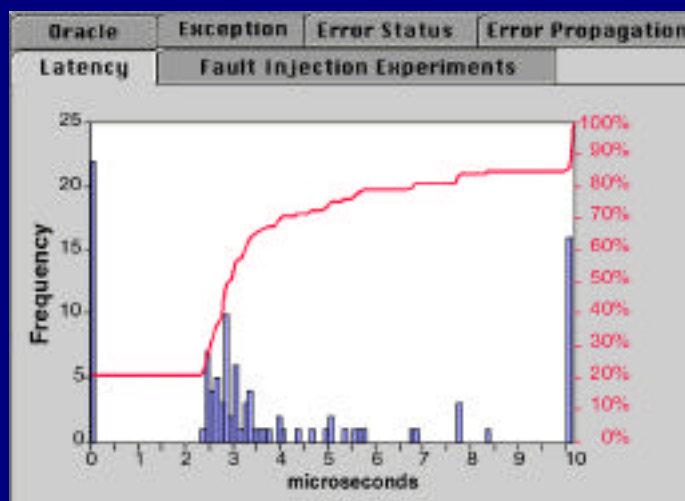
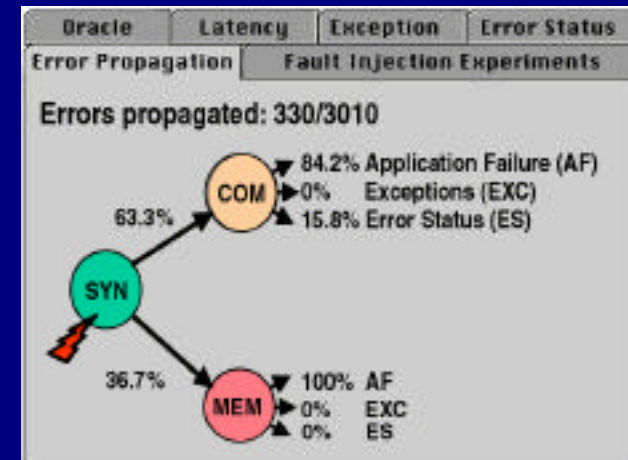
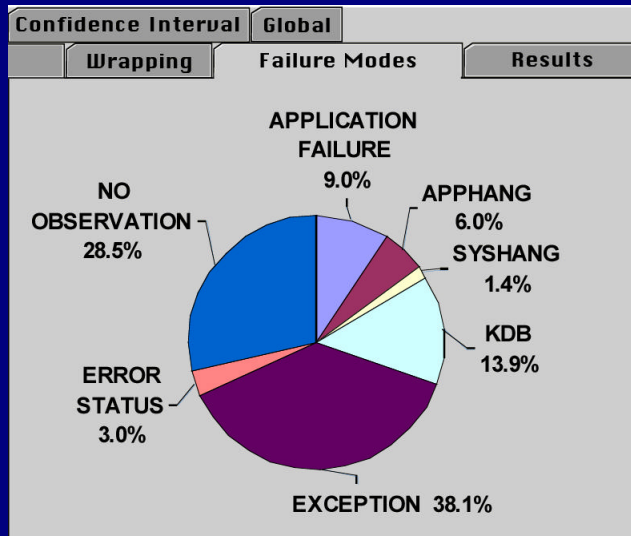
Description of the tool



Campaign outputs



Sample of measures



MAFALDA in action

The screenshot displays the MAFALDA configuration interface with the 'Fault Injection' tab selected. The interface is organized into five columns, each with a title and a list of radio button options:

- General** | **Wrapper** | **Fault Injection** | **Workload Processes** | **Auxiliary Processes** | **Destination Directories**
- Target Component**
 - Scheduling
 - Synchronization
 - Memory
 - Communication
- Model**
 - Microkernel**
 - Code
 - Data
 - Parameter
- Duration**
 - Transient
 - Permanent
- Trigger**
 - Spatial
 - Temporal
 - Both
- Type**
 - Stuck at
 - Bit Flip

Running experiments: the oracle

The Oracle interface is divided into several sections:

- Progress Bar:** Shows the duration of different phases: Rebooting (60s), Launching Processes (60s), and Monitoring (40s). A blue bar below indicates the current state.
- Console:** Lists the actors and their configurations:
 - th2(21), actor(SCH_scheduling), times (2)
 - th3(22), actor(SYN_semaphore), times (2)
 - th4(23), actor(SYN_semaphore), times (2)
 - memTester(26), actor(MEM_memory01), times (2)
 - memTransposer(27), actor(MEM_memory02), times (2)
 - ipcSender(24), actor(COM_comm01), times (2)
 - ipcReceiver(25), actor(COM_comm02), times (2)
- Outputs:** A log of system events and actions:
 - th1_create(th2) th2_threadScheduler(prio-->) MEM_BLOCK_1_OK SENDER_STEP1_OK th3_semP(s1)
 - th3_threadDelay(0) MEM_BLOCK_2_OK th1_doWork(A) th1_delay(20) th4_doWork(A) th4_semP(s1)
 - RECEIVER_STEP1_OK SENDER_STEP2_OK RECEIVER_STEP2_OK MEM_BLOCK_3_OK th2_doWork(B) th1_threadSchedu
 - th3_doWork(B) th3_semV(s1) th3_semP(s2) th2_delay(30) th4_doWork(C) th4_semV(s2)
 - th4_semV(s1) th4_threadDelay(0) th1_doWork(C) th2_doWork(D) th2_exit(0) SENDER_STEP3_OK
 - RECEIVER_STEP3_OK th1_create(th2) th2_threadScheduler(prio-->) th3_semP(s1) th3_threadDelay(0) MEM_BLOCK_1_
 - SENDER_STEP1_OK MEM_BLOCK_2_OK th1_doWork(A) th1_delay(20) th4_doWork(A) th4_semP(s1)
- State Monitors:** Four panels showing the state of different components:
 - SCH:** th1_create(th2), th2_threadScheduler(prio-->), th1_doWork(A), th1_delay(20), th2_doWork(B), th1_threadScheduler(prio-->), th2_delay(30).
 - SYN:** th3_semP(s1), th3_threadDelay(0), th4_doWork(A), th4_semP(s1), th3_doWork(B), th3_semV(s1), th3_semP(s2), th4_doWork(C).
 - MEM:** MEM_BLOCK_1_OK, MEM_BLOCK_2_OK, MEM_BLOCK_3_OK, MEM_BLOCK_1_OK, MEM_BLOCK_2_OK.
 - COM:** SENDER_STEP1_OK, RECEIVER_STEP1_OK, SENDER_STEP2_OK, RECEIVER_STEP2_OK, SENDER_STEP3_OK, RECEIVER_STEP3_OK, SENDER_STEP1_OK, RECEIVER_STEP1_OK.

Running experiments: fault injection

The screenshot displays a software interface for running fault injection experiments. At the top, there are several tabs: Error Propagation, Latency, Exception, Error Status, Confidence Interval, Global, Oracle, Fault Injection Experiments, Results, Failure Modes, and Wrapping. The 'Fault Injection Experiments' tab is active.

Experiment # 750 is shown in a text box. To the right, there are progress bars for Rebooting (60s), Launching Processes, and Monitoring (40s). A red bar indicates the Rebooting progress.

A yellow information icon is followed by a blue bar containing the text: **Fault impact: APPLICATION FAILURE**.

Console

```
[c042d32e:c042d32e:506:1978:_uSemUnblock:TEXT]: 3e XOR 1 -> 3f
```

Time trigger has occurred

The fault has been activated (breakpoint0)

Fault removed (Single Step Called)

Outputs

```
th2_exit(0) SENDER_STEP3_OK RECEIVER_STEP3_OK th1_create(th2) th2_threadScheduler(prio-->) MEM_BLOCK_1_
SENDER_STEP1_OK MEM_BLOCK_2_OK th1_doWork(A) th1_delay(20) th3_doWork(B) th3_semV(s1)
th3_semP(s2) th4_doWork(A) th4_semP(s1) th2_doWork(B) RECEIVER_STEP1_OK SENDER_STEP2_OK
RECEIVER_STEP2_OK th1_threadScheduler(prio-->) th2_delay(30) SENDER_STEP3_OK RECEIVER_STEP3_OK th4_dow
th4_semV(s2) th4_semV(s1) th4_threadDelay(0) th1_doWork(C) th2_doWork(D) th2_exit(0)
MEM_BLOCK_3_OK
```

SCH

```
th2_doWork(B)
th1_threadScheduler(prio
th2_delay(30)
th1_doWork(C)
th2_doWork(D)
th2_exit(0)
```

SYN

```
th4_doWork(A)
th4_semP(s1)
th3_doWork(B)
th3_semV(s1)
th3_semP(s2)
th3_threadDelay(0)
```

MEM

```
MEM_BLOCK_2_OK
MEM_BLOCK_3_OK
MEM_BLOCK_1_OK
MEM_BLOCK_2_OK
MEM_BLOCK_3_OK
```

COM

```
SENDER_STEP2_OK
RECEIVER_STEP2_OK
SENDER_STEP3_OK
RECEIVER_STEP3_OK
SENDER_STEP1_OK
RECEIVER_STEP1_OK
SENDER_STEP2_OK
```

Running experiments: results

The screenshot shows a software interface with a tabbed menu at the top. The tabs are: Error Propagation, Latency, Exception, Error Status, Confidence Interval, Global, Oracle, Fault Injection Experiments, Results, Failure Modes, and Wrapping. The 'Results' tab is selected, and the content area displays the following text:

```
Results File
Experiment number 1
Fault Activated! (recovered)
Fault :
[c042b32e:c042b32d:203:1978:_uSemBlock:TEXT]: 3e XOR 1 -> 3f
Failure :
Undetected Failure in SYN

Experiment number 2
Fault Activated! (recovered)
Latency : 2431.428571
Fault :
[c042a54b:c042a54b:921:2077:_uSemUnblock:TEXT]: 89 XOR 1 -> 88
Failure :
exception: Segmentation fault thread 23 PC c042a584 faultAddr 1 Actor 19 (SYN_semaphore) killed on exception
```

Lessons learnt (1-3)

- **Workload definition and oracle**
 - Generic workload / component \mathcal{A} design, programming flaws
 - Specific workload / application \mathcal{A} failure modes (oracle)
- **Fault injection**
 - Selection: random vs. predefined location of the bit-flip
 - Kernel injector: debugging features of modern microprocessors
 - Parameter injector: interception of kernel-calls (library-based vs. trap-based μ kernel)
- **Assertions and wrappers**
 - Formalize the expected behavior from the integrator viewpoint
 - Performance: tradeoff between modeling and runtime overhead
 - Temporal logic expressions interpreted on-line by a model checker

Lessons learnt (4-6)

- **Raw data analysis**
 - Analysis of logged data \Rightarrow identification of program flaw
 - User-defined semantics of the failure modes
- **Interpretation of results**
 - One campaign targets a microkernel instance & an activation profile
 - Variability of the results:
 - Stand-alone version vs. Posix-based version
 - Reactive application vs. static application
- **Target system evolution**
 - A slightly new instance \Rightarrow new campaign needed
 - Is the new release/version acceptable?
 - Is the new instance compatible with architectural solution?

Lessons learnt (7)

- **Integrator's vs. supplier's viewpoint**
 - Integrator
 - Weaknesses revealed
 - Decision: reject or encapsulate
 - » Appropriate wrappers
 - » Tradeoffs measurements
 - » Implementation: reflective framework
 - Supplier
 - identification of bugs not revealed by standard benchmarking activities \Rightarrow product improvement
 - Implementation of external error detection mechanisms:
 - » Development of the reflection module
 - » Mechanisms left open to the integrator

Conclusion

- **Experiments**

- Chorus ClassiX

- Failure mode analysis and wrapping (SYN & SCHED)
 - Source code \mathcal{AE} implementation of the reflective framework

- LynxOS

- Only failure mode analysis on a black-box instance
 - Metainterface delivered to the supplier

- **Current work and perspective**

- Characterization: extension of MAFALDA to real-time issues
 - Wrapping: formal description in temporal logic + on-line model checking
 - Implementation: reflective framework

Thank you!

*Jean-Charles Fabre, Manuel Rodríguez, Jean Arlat, Frédéric Salles
and Jean-Michel Sizun*



LAAS-CNRS
Toulouse, France

PRDC-2000, UCLA, Los Angeles, CA, USA — 18-20 December 2000

