Integrating COTS Software into Dependable Systems: Support to the Selection Process

Jean Arlat
(delivered by Jean-Charles Fabre)
Components and Systems Concerned

Components of a computer system

- **Application**: Oracle, Flight Control, ...
- **Middleware**: CORBA, DCOM, OLE, ...
- **Operating System**: Unix, Windows, Linux, ...
- **Microkernel**: Chorus, LinxOS, PalmOS, ...
- **Processor**: Pentium, PowerPC, ...

- Embedded control systems... RT microkernel-based
- Large-scale distributed systems... middleware-based
How to Build Dependable Systems from (Undependable) COTS Components?

- Assess the behavior in presence of faults -> Selection

- Level of Confidence sufficient -> Integrate

- Level of Confidence not sufficient:
  - Discard!
  - Fault containment mechanisms & service degradation
  - Error recovery mechanisms & service continuity
Target Systems: Software Executives

Motivation

- Complex software components whose development requires a great deal of expertise
- Basic services (management of memory, communication, synchronization, tasks, I/O, files, etc.) supporting application requirements
- Applications rely heavily on their behavior, including in the presence of faults

What executives?

- Real-time microkernels: Chorus, LynxOS, VxWorks, ...
- Generic OSs: Linux, Windows, ...
- Middleware: Corba, Dcom, ...
**Targeting COTS microkernels**

Parameter fault injection

Synchronisation by mutex

Fault injection

**Chorus vs. LynxOS**

Memory management

![Graph showing comparison between Chorus Classix r3.1 and LynxOS r 3.0.1](image)

- **Chorus Classix r3.1**
- **LynxOS r 3.0.1**

- **Applic. Fail**
- **Applic. Hang**
- **Syst. Hang**
- **Exception**
- **Error Status**
- **No Obs**
Example of 3 fault injection campaigns
- SCH: corruption of the running task
- TIM: corruption of timers
- SYN: corruption of synchronization system calls

Application failures in SCH and TIM and efficient error detection mechanisms in PCP
Targeting CORBA implementations

bitflips in IIOP requests
Situation can change according to the evolution of the COTS implementation through successive versions!

Additional ED and FT mechanisms must evolve accordingly!
Conclusion, Ongoing Work & Challenges

- Objective Insights to support Developer’s Design Choices and selection of the most robust COTS candidate

- Development of protection mechanisms by means of complementary wrapping techniques

- Solutions to adapt architectural choices, error detection and recovery mechanisms to the evolution of systems in operation

- Crucial need for dependability benchmarking
  - Comprehensive set of Benchmark Prototypes: Transaction Systems, OSs, Embedded Control Applications, ...
  - The DBench project had this aim of disseminating Benchmark Prototypes