Open Source and Critical Systems: Some Challenges and Lessons Learnt

Jean Arlat

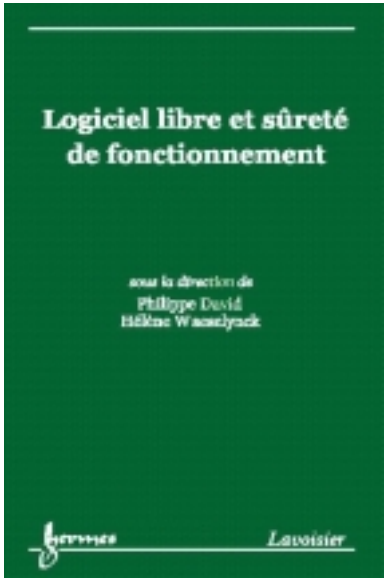
45th IFIP WG 10.4 Meeting — March 6-7, 2004 — Moorea, French Polynesia

RIS
Network on Dependability Engineering

<http://www.ris.prd.fr>

- **Working Group on OSS and Dependability**
  (RIS members + participants from Industry ESA, SNCF, and Academia LSV, INRETS, IRISA)

- **Publication of a book**
  Hermes Science Publications
  (in French) -->



Agenda

- Context
- Background
- Open Source Software and Dependability Issues
- Examples
  - ✓ RTEMS
  - ✓ Linux
- Concluding Remarks

Evolution of Critical Systems

- Ubiquity of Computerized Systems Including in Critical Applications
- Cost Issues Become More and More Stringent
  - ➔ Favor Reuse wrt Custom Solutions and Developments
- Systems are open entities and need to interact
  - ➔ Use of Standard Interfaces
  - ➔ Security Issues are Important
- Increase Use of Software
  - ✓ 48 kb on SPOT 1 in 1980 ➔ 1,2 Mo on Mars Express in 2003
  - ✓ 25 kb on A300B en 1974 ➔ 64 Mo on A380 en 2005
- Certification requirements extend to an increasingly larger set of industrial domains

How to Cope with Software?

- Insights gained by Industrial partners within RIS:
  COTS Components do not provide a fully satisfactory solution for critical systems
- Nevertheless, the principle of using pre-developed software components is an appealing approach

-> Open Source Software: Threat or Opportunity ?

Background

- **Protection of users wrt Software vendors**
  - ✓ 1984, **Richard Stallman's GNU manifeto**
  - ✓ 1985, *Free Software Foundation*
  - ✓ 1990, Maturité du projet GNU/Linux
  - ✓ 1990, "Open Source", terme déposé par l'OSI
  - ✓ 2000, Political and Economical Recognition
- **Principle: Force to freedom (with a license)**
  1. Freedom of use
  2. Freedom of change
  3. Freedom of (re)-diffusion
  4. Freedom of access to source code
- **Organization**
  - ✓ Structuring of OSS projects very similar to industrial projects
  - ✓ Financing of the projects achieved by foundations, grants, industrial partners
  - ✓ Recognition of developer's skill by some form of hierarchy

OSS and Critical Systems

Heresy or Tomorrow's Reality?

Insights Gained from Using COTS

- **Requirements for Critical systems**
  - ✓ Detailed knowledge
  - ✓ Adaptability of the COTS to the system
  - ✓ Availability of certification documents
  - ✓ Availability of the COTS for a 5-10 year period
  - ✓ Maintenance over a 10-20 year period
  - ✓ Compliance wrt standards
  - ✓ Cost and modalities of the license

- **Typical Concerns**
  - ✓ Cost for COTS support might not be interesting for the provider
    - ➔ cost of the support
  - ✓ Lack of Knowledge about the COTS
    - ➔ cost the certification documents
  - ✓ Diverging interests between the user and the provider
    - ➔ cost for freezing the COTS
  - ✓ Stability of the solution overtime
    - ➔ cost of maintenance
  - ✓ For small number of units
    - ➔ cost of licenses becomes significative
  - ✓ Proprietary clauses might constitute a blocking point
    - ➔ cost of negotiating licenses

Risk Management

COTS

- **License**
  - ✓ Strategic problem
  - ✓ Industry is used to cope with such an issue
- **Component Failure**
  - ✓ The liability of the provider is limited.
  - ✓ Risk of propagating errors of the component to the whole system is real and has to be handled by the Integrator
  - ✓ The integrator has no/little detailed knowledge about the component
  - ✓ The confidence relationship between the Provider and the Integrator is not sufficient
- **Discontinuity of Provider or component**

OSS

- **License**
  - ✓ Freedom of use
  - ✓ GPL introduces new risks (contamination)
- **Component Failure**
  - ✓ Integrator is only responsible
  - ✓ Risk of propagating errors of the component to the whole system has to be handled by the Integrator.
  - ✓ The Integrator has access to the source code
- **The component can be maintained by the Integrator**

Impact of Maintenance

- **Critical systems feature very long operational life**
  - ✓ Satellites: 15 years
  - ✓ Command/control for nuclear propulsion in submarines: 40 years
- **Maintenance policy is accounted for as part of design of the system**
  - ✓ Architectural principles that minimise the impact of changes between successive releases/versions
  - ➔ Favor interface standards (e.g., POSIX, …)
  - ➔ Encapsulation mechanisms to minimize the impact of evolutions
- **Long term maintenance has to account for the risk of loosing the Provider**
  - ➔ Source code availability is often necessary

Potential Benefits from Using OSS

- Is the effort necessary to comprehend the technology of an OSS component worth, so that it allows for a better management of the life-cycle of a critical system ?
- Several scenarios can be considered for the various phases:
  1. Acquisition of the technology of the OSS component
  2. Effort to adapt the OSSS to the system
  3. Elaboration of the documents for certification
  4. Maintenance in operational life
  5. Long term maintenance support for the OSS
  6. Management of the major evolutions of the system

Scenarios for Using an OSS Component

	Acquisition of technology	Adaptation to the system	Certification documents	Maintenance in operation	Long term maintenance	Major evolutions	Summary
Scenario 1 ✓No certification ✓No maintenance	Not necessary	Provider	Not necessary	Provider	Source code	Provider	No investment Risk is supported Case of space industry today
Scenario 2 ✓No certification ✓Maintenance	Provider	Provider	Not necessary	Integrator	Integrator	Integrator	Investment into a maitenance team for OSS In-house maintenance when the component is critical (e.g, OS) Trend in space
Scenario 3 ✓Certification ✓No maintenance	Provider	Integrator	Integrator	Provider	Provider	Provider	Acquisition of technology Certification by the Integrator No maintenance
Scenario 4 ✓Certification ✓Maintenance	Provider	Integrator	Integrator	Integrator	Integrator	Integrator	Acquisition of technology Certification by the Integrator No maintenance

Main Features

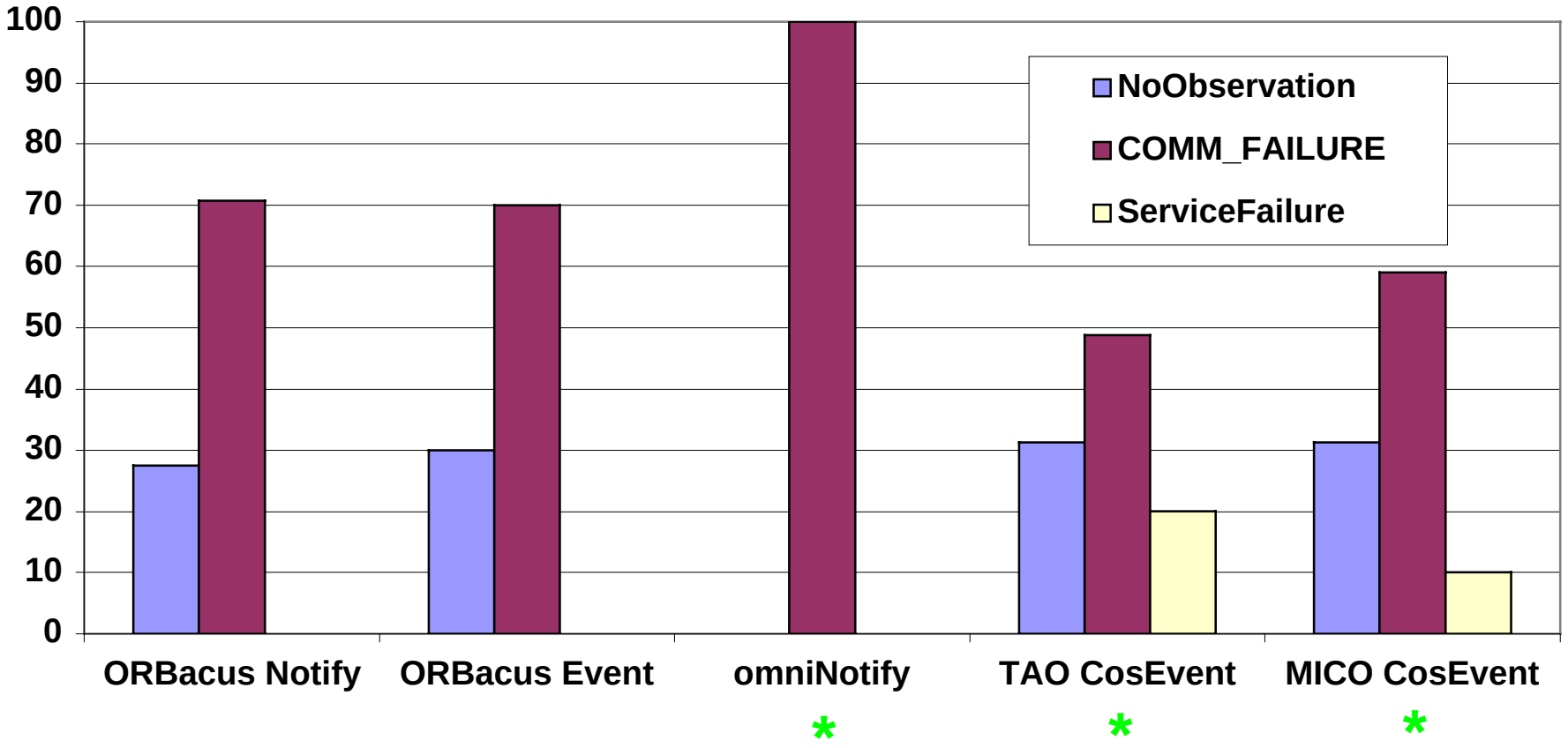
- **Access to the Source Code**
  - ✓ Cope with risks of evolution of the component
  - ✓ Disappearance of Provider
  - ✓ Better prepared to use the source code in case of a problem (wrt to case of COTS)
- **Support by an OSS Technology Provider**
  - ✓ Same service and relationship as in the case of COTS
  - ✓ The support provided is often of better quality, since this is this facet of expertise wrt the OSS that is the basic service provided
- **Acquisition of the Technology**
  - ✓ Acquisition of a detailed knowledge about the OSS can be long and costly (several p.y)
  - ✓ The investment is heavy (both short and long term) as it is necessary to maintain a competent team all along the system life-cycle

Dependability of OSS

- **An architectural platform to host the component is necessary:**
  - ✓ The OSS is suspected (in the sense of its failure modes)
  - ✓ Functionalities might be over-abundant or not fully suitable
- **Characterization of failure modes**
- **Use of wrappers**
- **Partitioning into zones with different levels of criticallity**
  - ✓ Support openness of critical systems to increased interactions (interoperability functions)
- **Security**
  - ✓ Transparency induced by access to the source code makes a significant difference?
  - ✓ Openness to various contributors might be a threat?

Characterization of Failure Modes

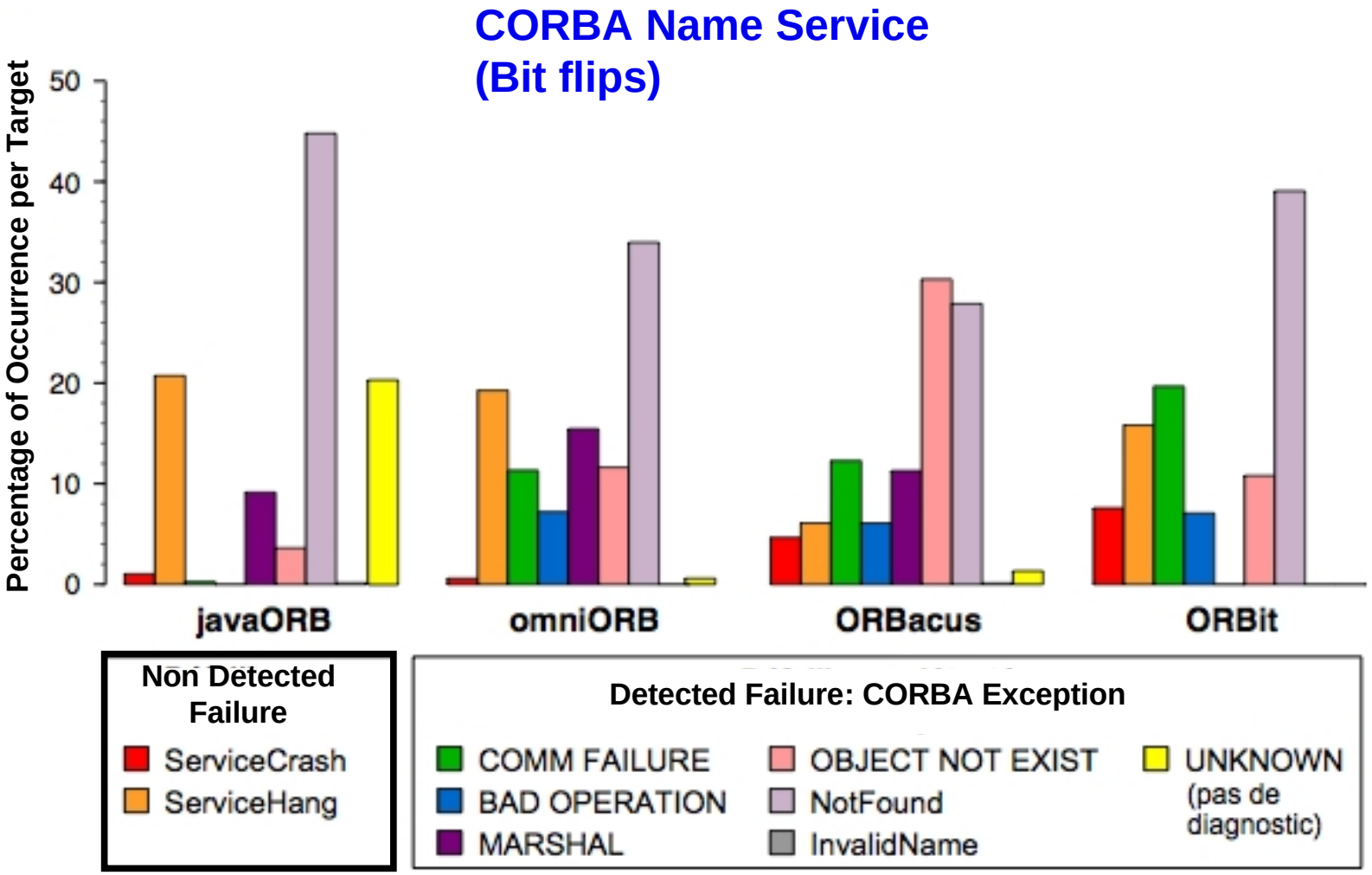
CORBA Event Service
Client-side exceptions (bit-flip experiments)



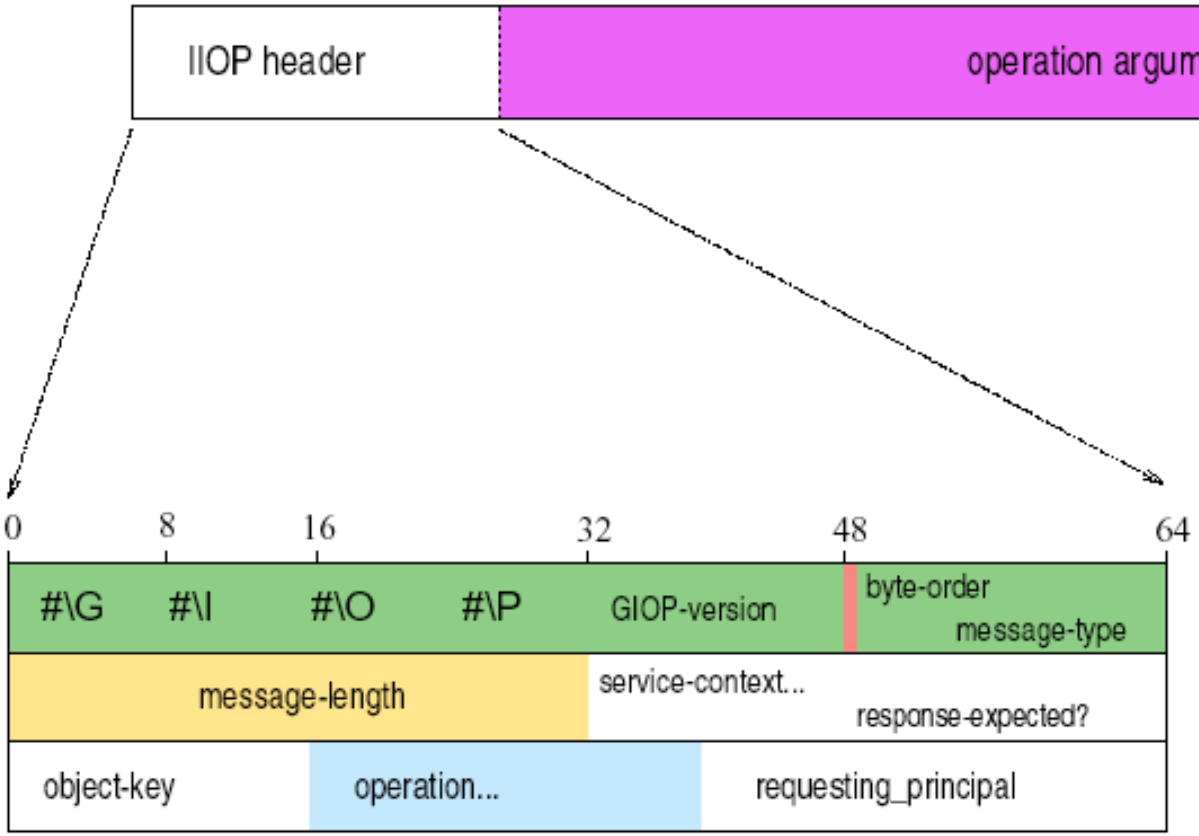
Rack of x86-based machines running **Linux kernel version 2.4.18**

* OSS Targets

Characterization of Failure Modes



Bit Flip - Detailed Analysis



OBJECT_NOT_EXIST
MARSHAL

BAD_OPERATION

COMM_FAILURE

ServiceHang

ServiceCrash

Certification

- Certification corresponds to the acceptance by a separate entity of the proof that a dependability level has been achieved
- Certification has a strong impact on system design
- Dependability and "certifiability" are not explicitly considered by OSS

  ➔ Reluctance of Industry to adopt OSS solutions

  - ✓ Responsibility of the Industrial Integrator
- Analyse of the certification processes of various domains in order to identify the methods and efforts necessary to allow the systems to be certifiable

  ➔ OSS has to demonstrate a significant competitive edge for the system, without compromising the dependability

Classification of Criticallity

- **Classification of criticallity is rather consistent among various industrial domains**
  - ✓ DAL (*Development Assurance Level*) for Avionics
  - ✓ SIL (*Safety Integrity Level*) fore Railway
  - ✓ …

Class	Railway	Avionics	Space	Nuclear
No impact	SIL 0	E	/	/
Impact on system	SIL 1-2	C-D	critical	B and C
Impact on human lifes	SIL 3-4	A-B	catastrophic	A

Impact of Assurance Levels for Civil Aviation

- **Top-Down analysis up to any equipment contributing to safety**
- **Regulation Authority has elaborated reference documents:**
  - ✓ Document ARP 4754 (system level)
  - ✓ Document DO-178B, critical software (e.g., flight control)
- **Various categories of software components (from A to E) are identified according to the harm that can be caused by the failure conditions at system level, in which the software can potentially contribute (ARP 4754)**

Architectural Solutions According criticallity

Classification of failure conditions	Degree of redundancy (generic principle)		
	0	1	2
Catastrophic	A	B	C
Dangerous	B	C	D
Major	C	D	D
Minor	D	D	D
No impact on safety	E	E	E
	Software DAL (Development Assurance Level)		

If a critical function prone to lead to a catastrophic failure of the system is:

- not redunded —> it is considered as a category A software
- "duplicated" (d°1) —> Each replica/version can then be classified as category B
- "triplicated" (d°2) —> Each replica/version can then be classified as category C

Certification, Dependability and Open Source

- A Critical System can be designed using redunded elements of lower critality levels provided that the redundancies are managed according to the safety requirements at system level.
- Communication protocols or real time OSs are potential candidates for levels B or C.

  ➔ Provision of redundancy allows for intregrating OSS (e.g., levels C or D)
- The use of OSS for A (or B) level(s) corresponds to a specific process in which the software has been developed by the industrial and certified at such level(s).

  ➔ Specific development of OSS

Examples

- Space
- Avionics

Generic Layered Architectural Design (1)

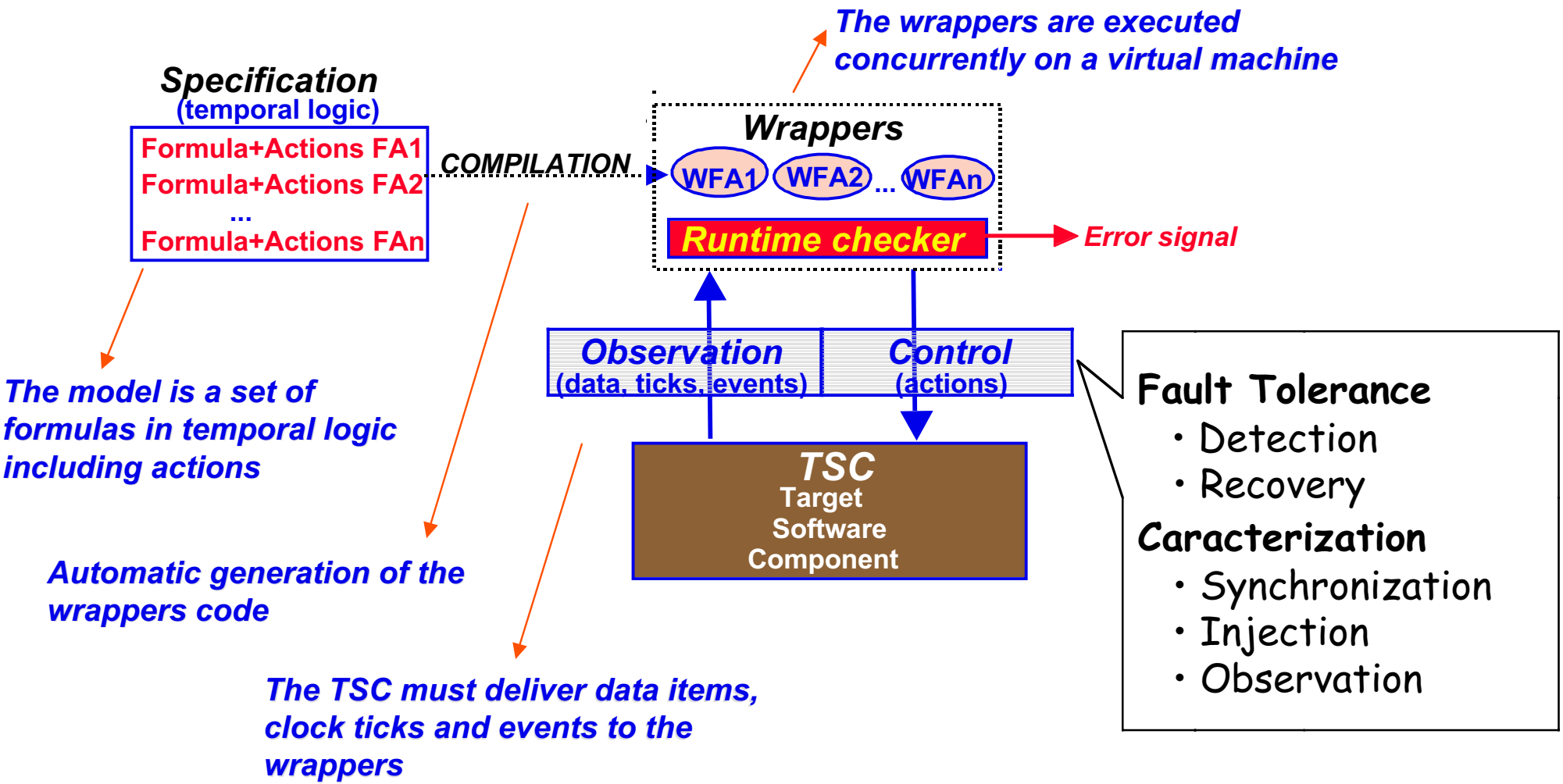
Integration of of-the-shelf components (COTS or OSS) relies usually on a three layer architecture (bottom-up):

- Level 1 : Execution environment abstracting the hardware and basic software resources
- Level 2 : Standardized or proprietary services supporting the execution of applications
- Level 3 : High-level host platform for supporting applications

Integration of off-the-shelf solutions

- **Use of COTS or OSS mainly at level 1**
  - ✓ Real Time executives, communications stacks, graphical library, etc.
- **Level 2 usually (in embedded systems) features architectural solutions to the use of COTS or OSS in order to master their integration, assuming that their design cannot be sufficiently masterred**
  - ✓ Fault containment and fault tolerance mechanisms
    - ✦ Partitioning
    - ✦ Wrapping
  - ✓ Enforce independency between applications and restrict the dependency of applications wrt the lower layers (including hardware)
- **Level 3 is application domain dependent**
  - ✓ In practice, specific solutions exist for each domain
- **Environments for development and production constitute another emerging domain where OSS solutions are succesfully introduced (compilers, etc.)**

Conceptual Framework for Wrapping Microkernel-based Real Time Systems



Space Domain

Integration of RTEMS

Context

- **New generation of 32-bit processors**
  - ✓ Developments in C
  - ✓ Necessity to identify a suitable "real-time" software executive
- **On-board system**
  - ✓ Difficulty of maintenance -> autonomy to provide operational
  - ✓ Increasing economical and calendar constraints
  - ✓ Trends toward standardization (interfaces et services)
- **Three solutions (potentially)**
  - ✓ COTS
    - ✦ Already been conducted on VxWorks: Columbus,…
  - ✓ OSS
    - ✦ Encouraged by ESA
  - ✓ Development of a specific solution -> expertise required hardly cost-effective in space domain

Why RTEMS?

- **ESA Incentive**
- **Flexibility in the configuration** (componentized μkernel)
- **Flexibility in the elaboration and implementation of a comprehensive policy for maintenance, deployment and durability**
  - ✓ Source code is available and can be freely modified
- **Normalisation at European level to improve quality**

Support and Environment

- **Industry is willing to maintain the product**
- **Elaboration of a "RTEMS process"**
  - ✓ Execution platform (based upon ERC32SC processor)
  - ✓ Configuration management independently of user projects
  - ✓ Watch of the « public » version
  - ✓ Management of modification request from the users
- **Effort for documenting the product in conformance with standard rules of clients**
  - ✓ Creation of a validation plan
  - ✓ Implementation of traceability support
- **Environnement de développement**
  - ✓ Use of GNU tools available for RTEMS
  - ✓ Use of specific means for real time tests

Linux: A Candidate for Embeddeed Avionics?

Context

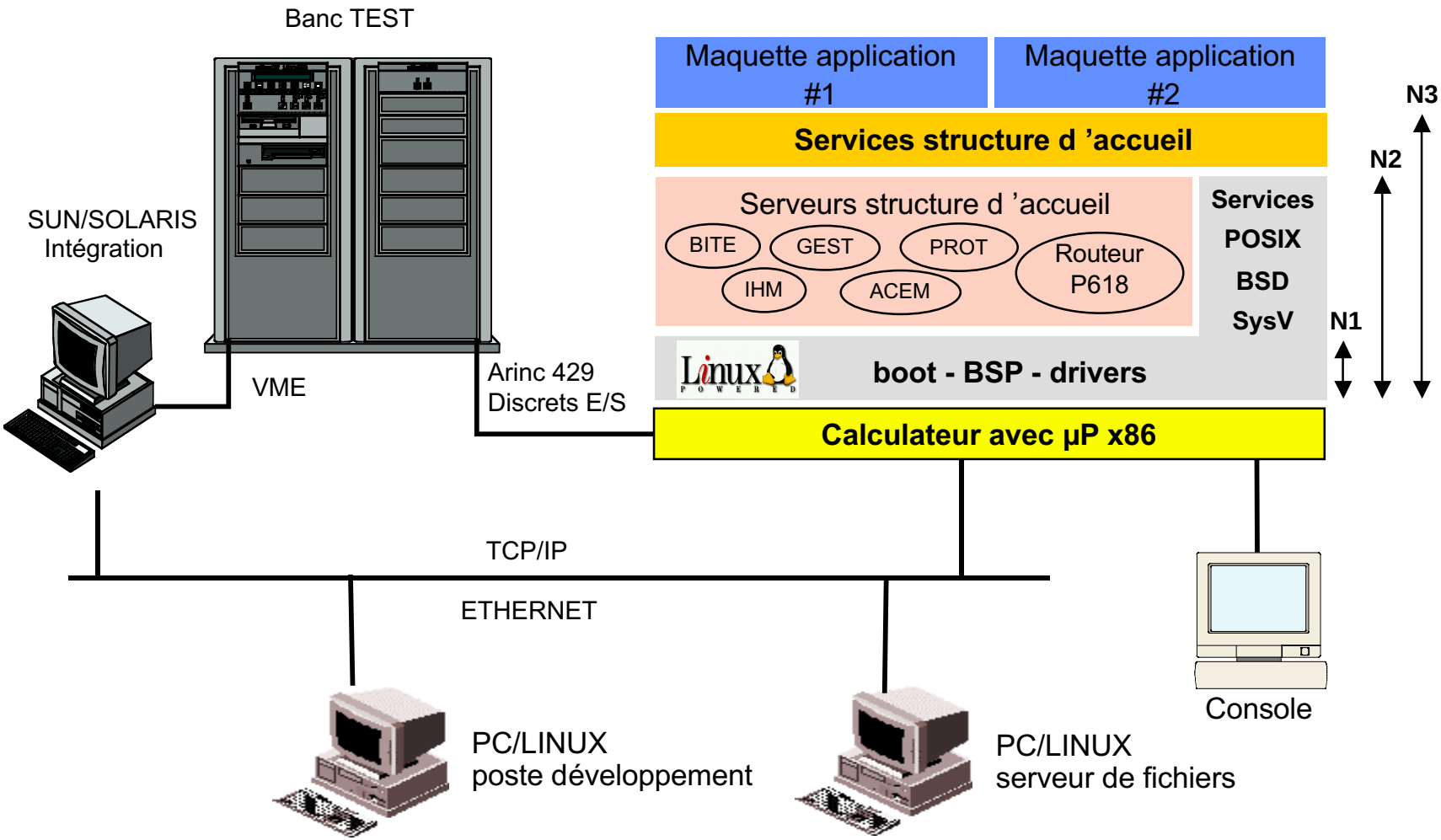
- **Questions about Linux… Is it possible that Linux be:**
  - ✓ Integrated into non critical avionics equipments?
  - ✓ Suitable for avionics requirements
  - ✓ Included into a DO178B-based certification process

- **Properties required**
  - ✓ Partitioning: Memory, Timing, I/O

- **ARINC A653 Interface Standard -> Loss of interoperability -> Need for developing specific tools (reuse is difficult)**

  => Use Linux as a platform for hosting avionics applications ensuring interoperability and partitioning

Prototype Platform for "Embedded Linux"



Concluding Remarks

- As is already the case for many computer sectors, industry concerned with critical systems is positively considering using OSS solutions (several experiments are underway)

- The main advantage is accessibility to the source code:
  - ✓ Mandatory for Certification
  - ✓ Less dependency with respect to provider

- Some "open" issues:
  - ✓ Security?
  - ✓ Licensing?
  - ✓ Overall cost?
  - ✓ How to capitalize insights among various industrial sectors?
  - ✓ How to contribute to the "Open Development Movement"?
  - ✓ …