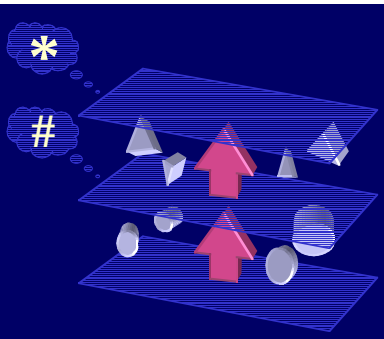


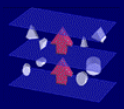
Open Middleware and Reflective Components for Adaptive Fault-Tolerant Systems

Marc-Olivier Killijian

Toulouse, France

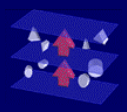
RIS - Groupe de Travail "Intergiciel et Sûreté de Fonctionnement"(Toulouse, 3 juin 2003)

Outline



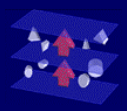
- **Context and status**
- **Reflective computing**
- **Components and reflection**
- **Current actions and Conclusion**

Context



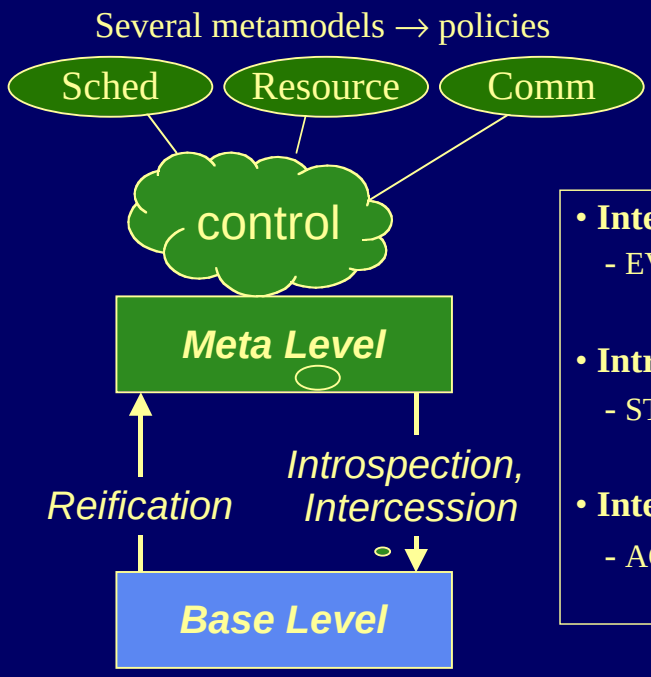
- **Middleware-based systems now used in systems with high dependability requirements (often generic, more and more specific, “customized”, also using COTS components)**
- **As middleware is used in large projects, long life cycles, both functional and non-functional evolution is required while meeting dependability requirements**
- **Several challenges…**
  - a technology that enables systems to evolve ⇒ **components**
  - techniques to make non-functional mechanisms adaptive ⇒ **reflection**
  - techniques to measure the effects of updates on dependability ⇒ **validation**

Context



- **Middleware-based systems now used in systems with high dependability requirements (often generic, more and more specific, “customized”, also using COTS components)**
- **As middleware is used in large projects, long life cycles, both functional and non-functional evolution is required while meeting dependability requirements**
- **Several challenges…**
  - a technology that enables systems to evolve ⇒ **components**
  - techniques to make non-functional mechanisms adaptive ⇒ **reflection**
  - techniques to measure the effects of updates on dependability ⇒ **validation**

Principles of Reflective Frameworks



Separation of concerns

- **Interception** ↑:
  - EVENTS
- **Introspection** ↕:
  - STATE
- **Intercession** ↓:
  - ACTION

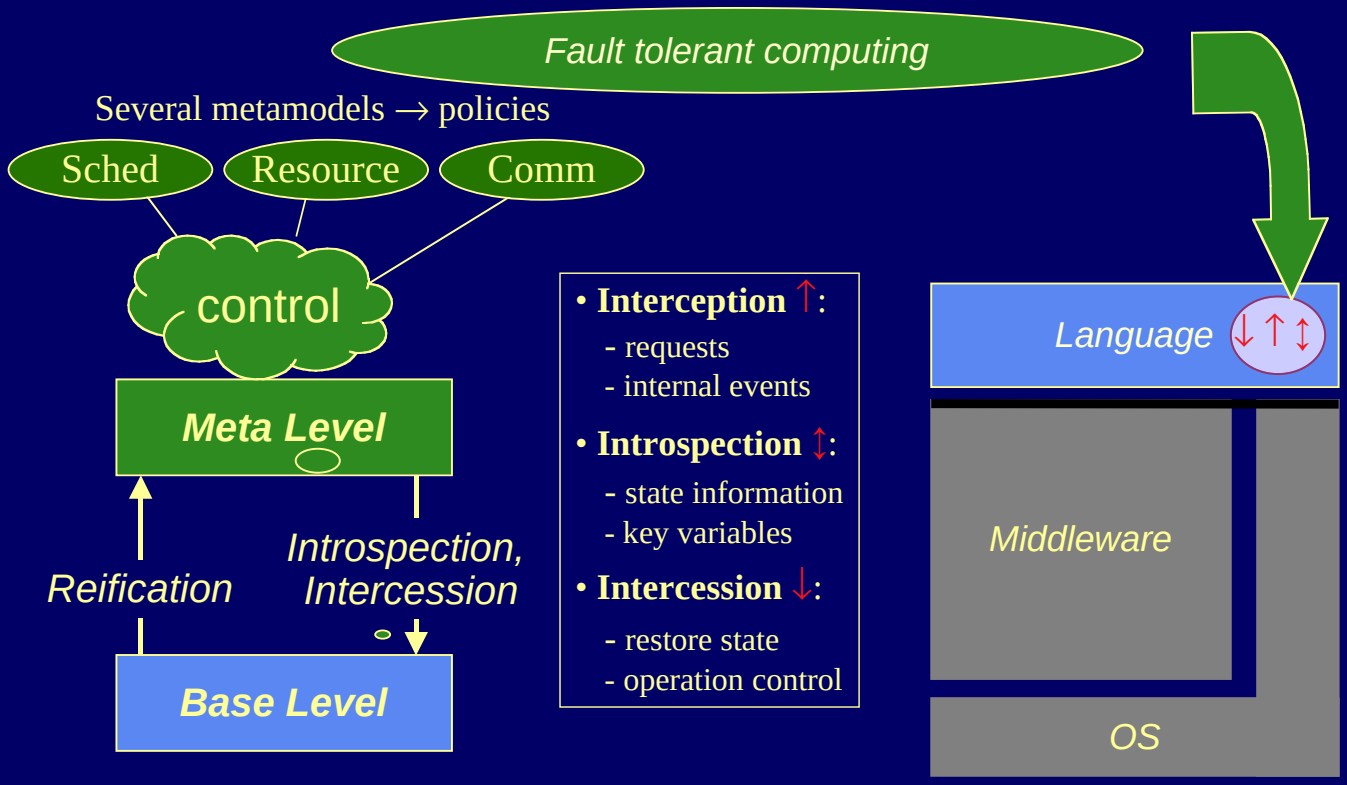
Principles of Reflective Frameworks

Fault tolerant computing

Separation of concerns

- **Interception** ↑:
  - requests
  - internal events
- **Introspection** ↕:
  - state information
  - key variables
- **Intercession** ↓:
  - restore state
  - operation control

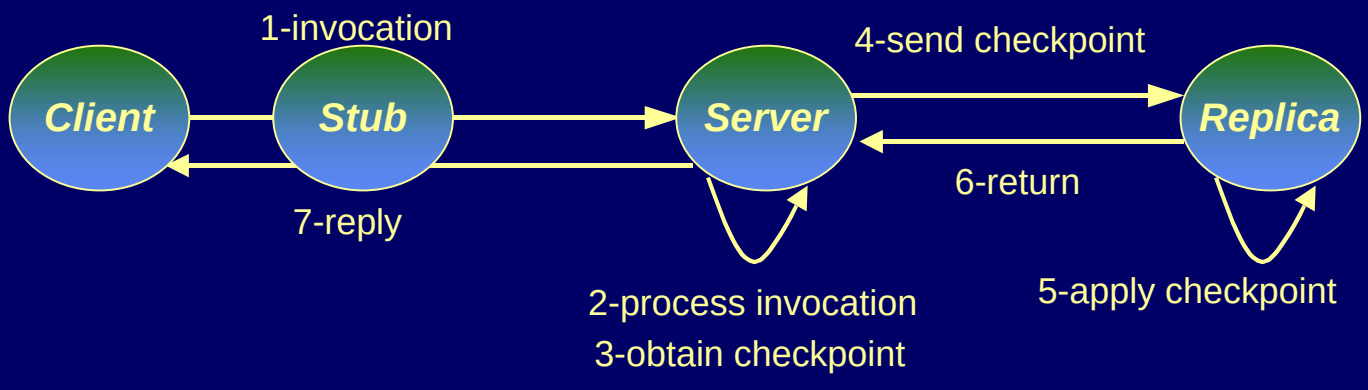
Principles of Reflective Frameworks



Example

Conventional Implementation

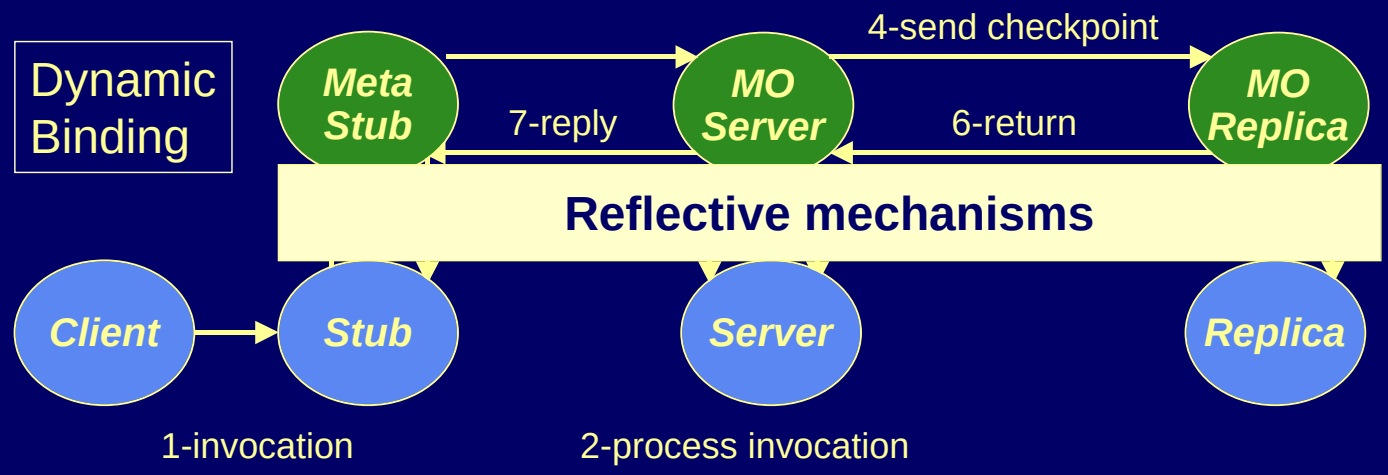
of fault tolerance strategies



Example

Reflective Implementation

of fault tolerance strategies



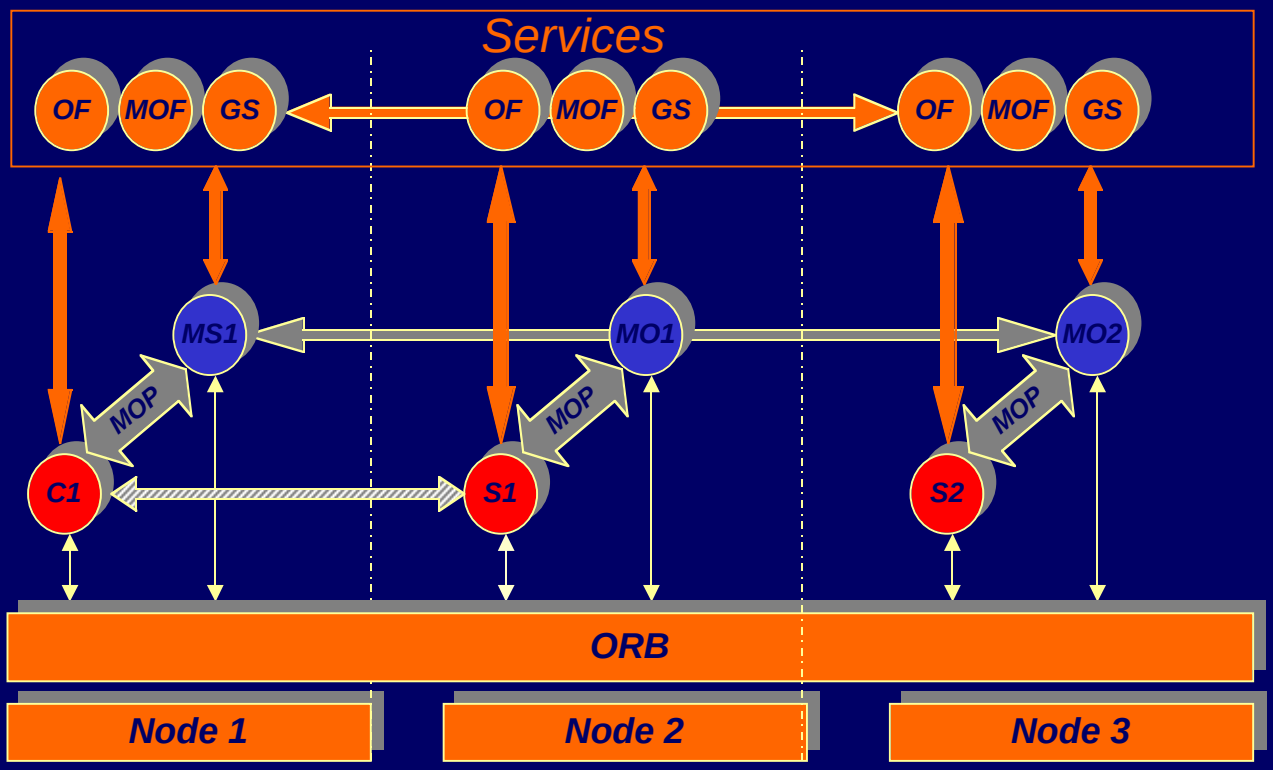
Observation
- invocations
- state capture

Control
- method calls
- state restoration

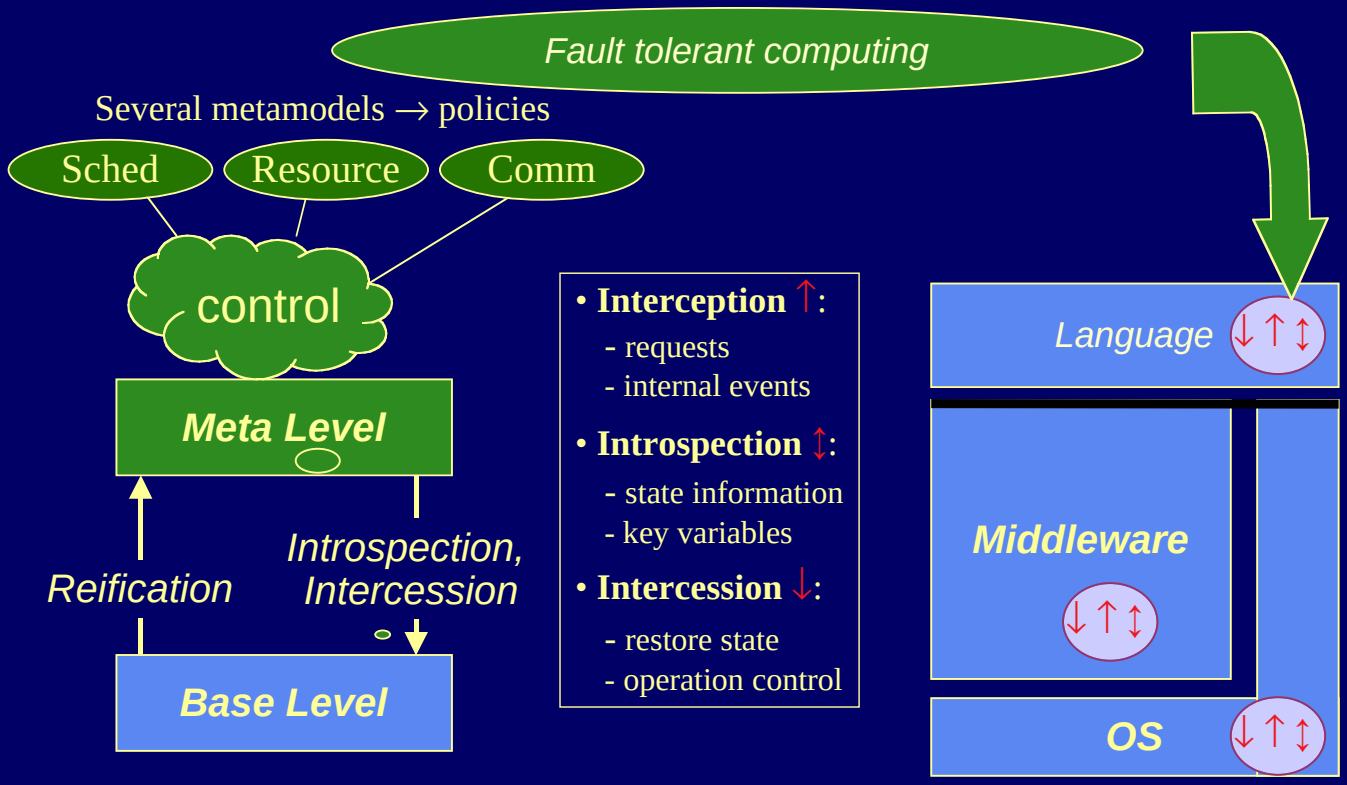
Example

Based on reflective languages and open compilers

FRIENDS V2

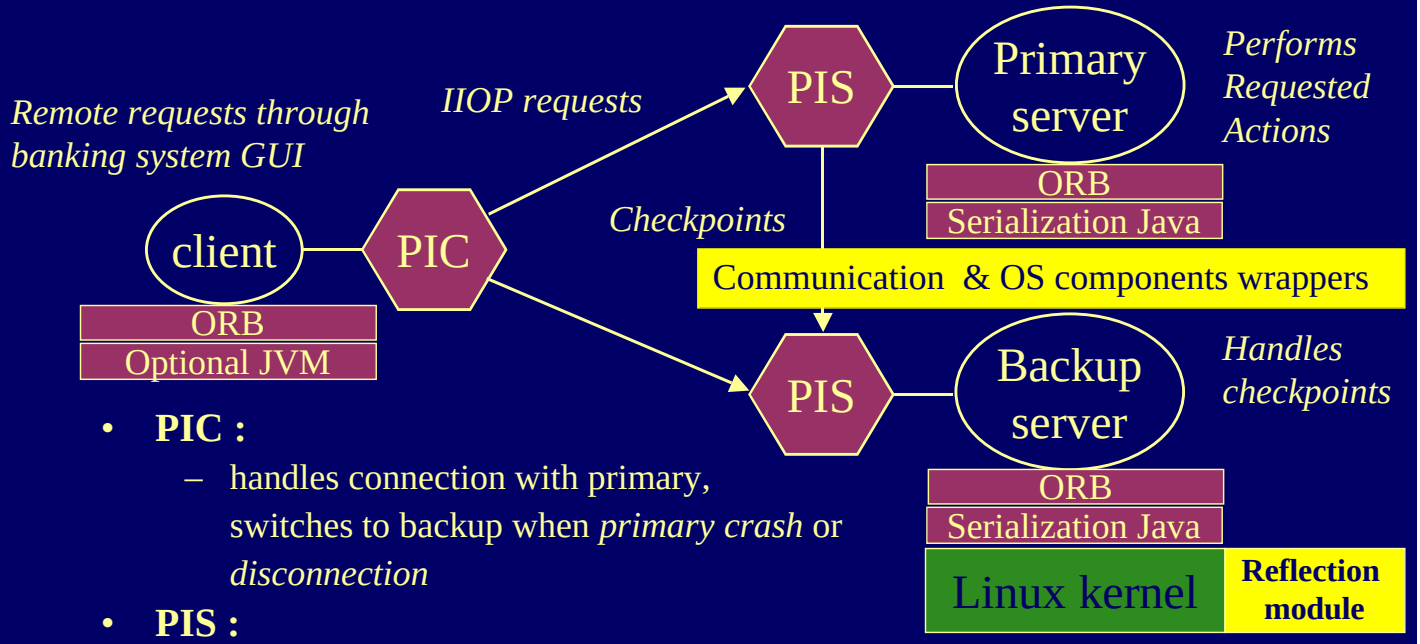


Principles of Reflective Frameworks



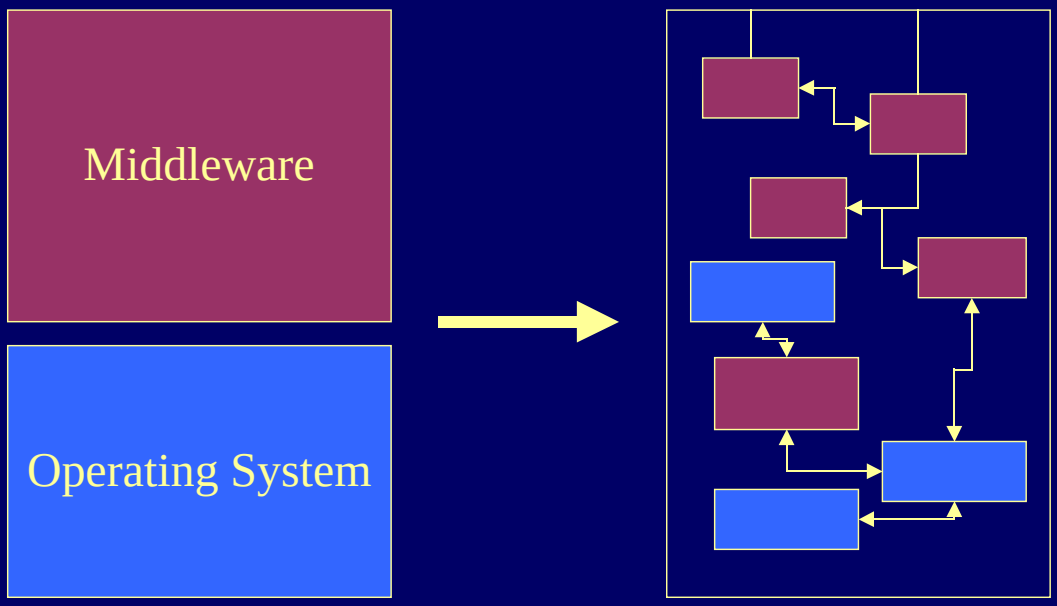
Example

DAISY: Dependable Adaptive Interceptors & Serialization-based sYstem



- **PIC :**
  - handles connection with primary, switches to backup when *primary crash* or *disconnection*
- **PIS :**
  - performs the primary-backup replication (PBR) protocol, including checkpointing and recovery when *primary crash* or *disconnection*

From « black box » to Components

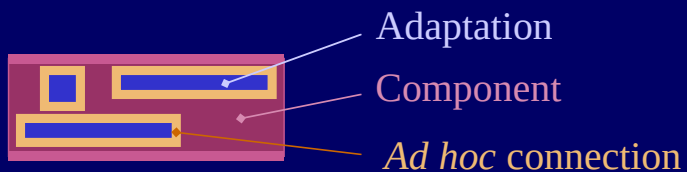


Black box layering

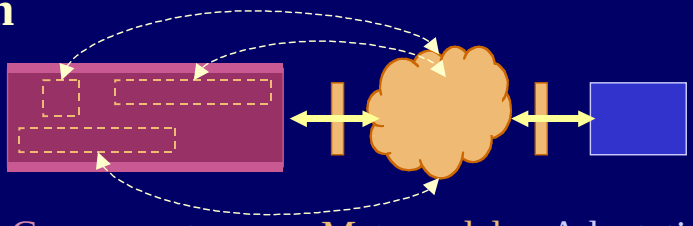
Open components

Mastering Adaptation with Reflection

- **Ad hoc adaptation on a case-by-case basis: too rigid**

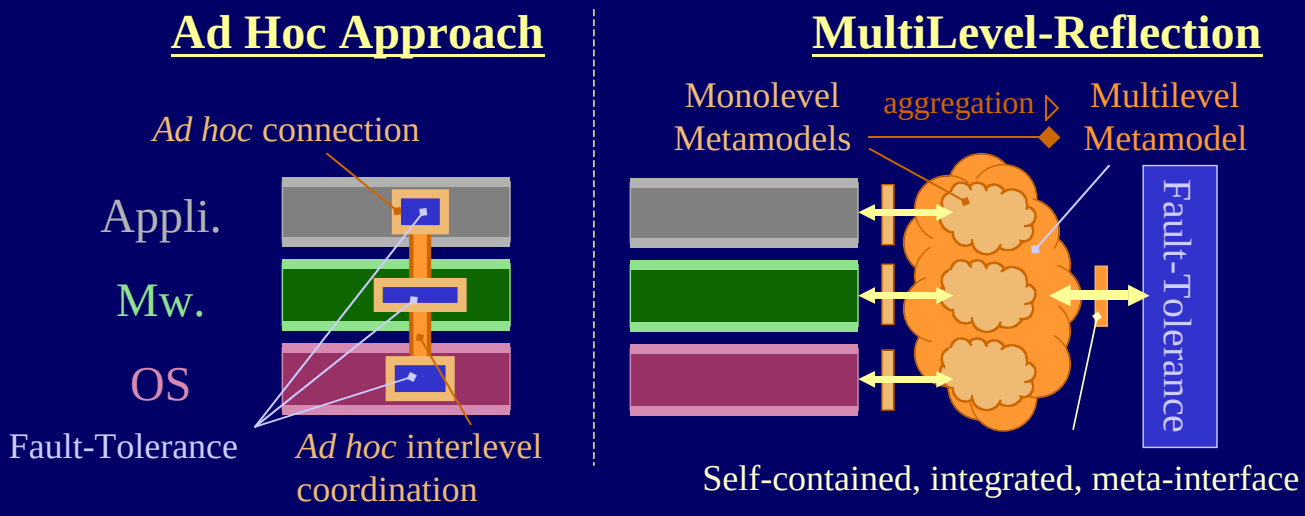


- **Reflective architecture: principled, disciplined adaptation**



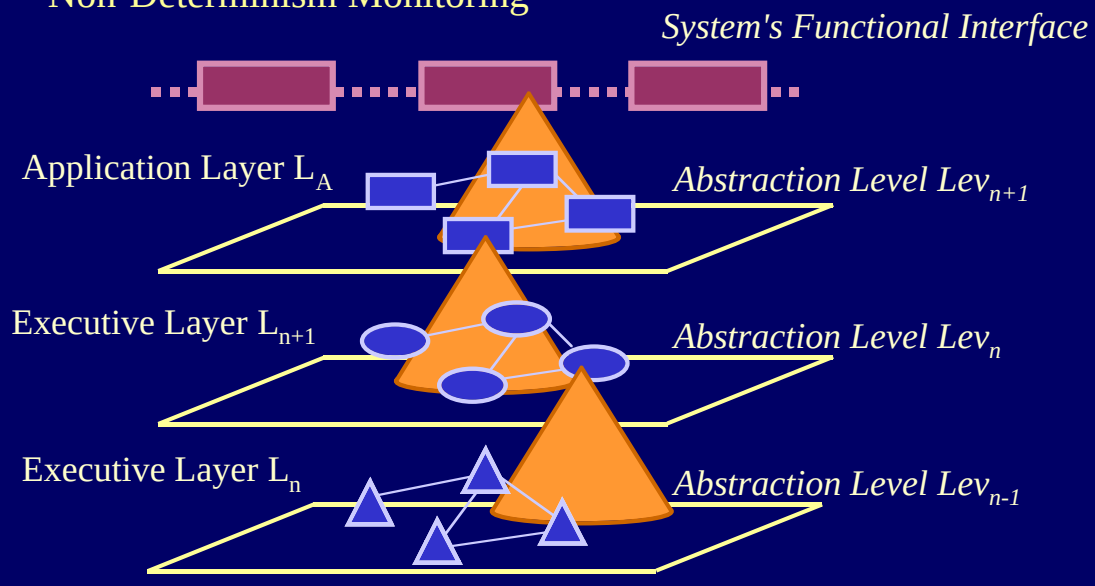
Mastering Inter-Component Coupling

- **Abstraction levels.**
  - Higher levels: partial info / rich semantics
  - Lower levels: complete info / poor semantics



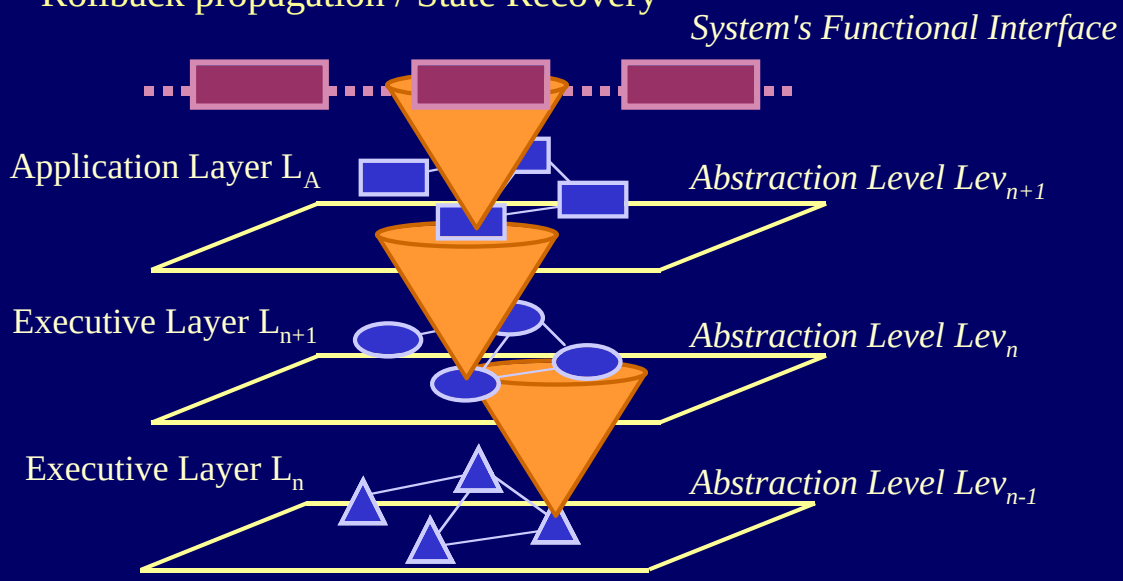
Coupling Metamodels (I)

- **Top Down Observation & Control**
  - State Capture in several components
  - Non-Determinism Monitoring

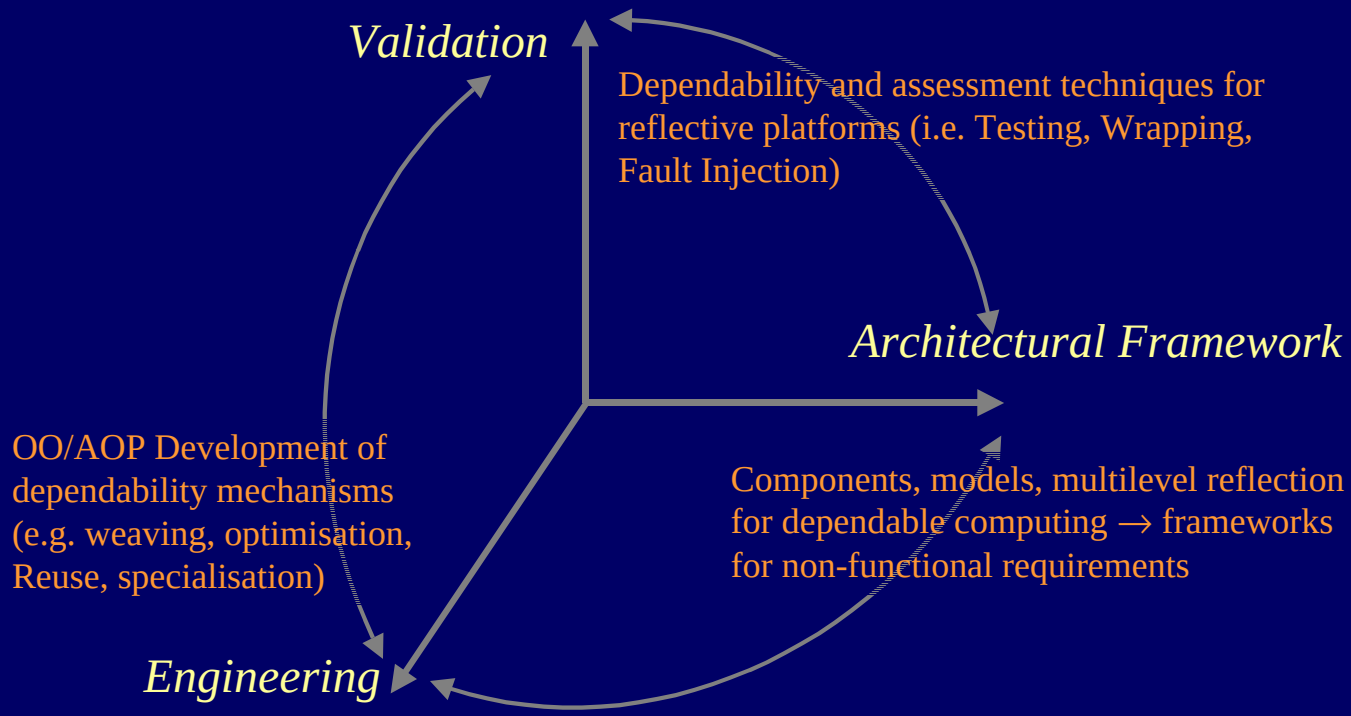


Coupling Metamodels (II)

- **Bottom Up Observation & Control**
  - Fault-Propagation Analysis / Confinement
  - Rollback propagation / State Recovery



Research Tracks & Challenges



Defence Space Energy Transport Telecom Automotive

Validation

Dependability and assessment techniques for reflective platforms (i.e. Testing, Wrapping, Fault Injection)

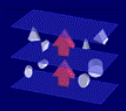
Application on Real Case Studies

Architectural Framework

OO/AOP Development of dependability mechanisms (e.g. weaving, optimisation, Reuse, specialisation)

Components, models, multilevel reflection for dependable computing → frameworks for non-functional requirements

Engineering



Key Research Actions

- **Components, frameworks, reflection for building Dependable Systems**
  - Middleware and OS components
  - Bindings and frameworks
  - Metamodels and meta protocols
- **Dependability of component-based and reflective architectures**
  - Testing, Fault injection and other validation techniques…
  - Evaluation w.r.t a non-functional property
  - Runtime assertions and wrapping

Example

Reflective Wrapping Framework

Kernel services specification

Properties:
temporal logic formulas

TL2C Compiler

W1 Wi Wn

Runtime checker

Synchronous calls

Asynchronous evts

Specification of reflective features:
- introspection
- intercession
- reification

MetaAPI

- external interceptors (EI)
- internal interceptors (II)

MetaAPI | Standard API

Introspection
Intercession

External Interceptors (EI)

II RTK II

Example

Reflective Wrapping Framework

always

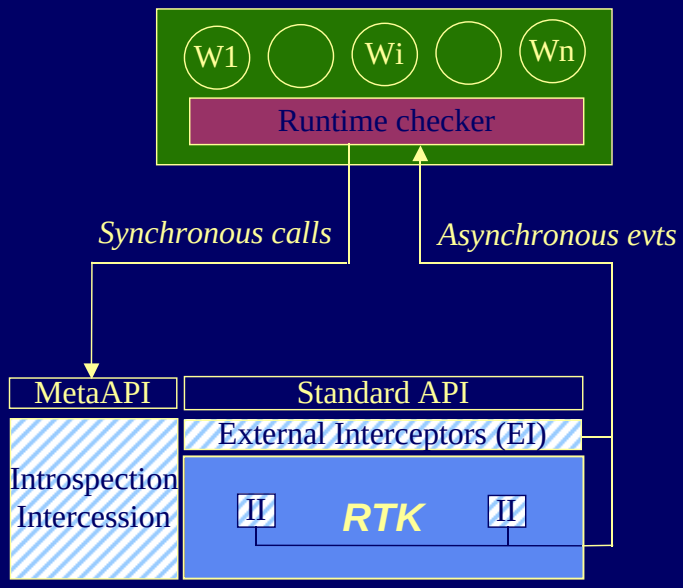
([Flow] = ↑Create (th_b) ∧ th_a = [Running]

∧ sometime([Flow] = ↑signal (th_b)

∧ [Running] = th_a ∧ prio (th_b) ≤ prio (th_a))

⇒ $Next_{event}$

([Running] = th_a ∧ th_b ∈ [$ReadyQueue_{prio(thb)}$]))



Example

Reflective Wrapping Framework

always

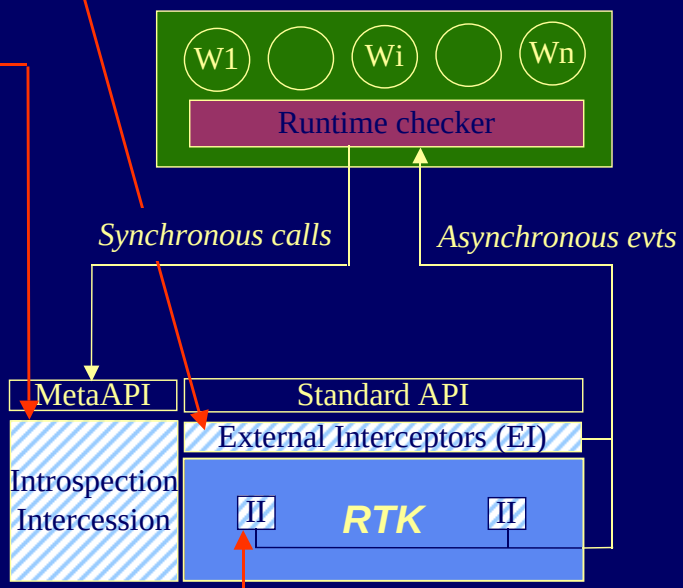
([Flow] = ↑Create (th_b) ∧ th_a = [Running]

∧ sometime([Flow] = ↑signal (th_b)

∧ [Running] = th_a ∧ prio (th_b) ≤ prio (th_a))

⇒ $Next_{event}$

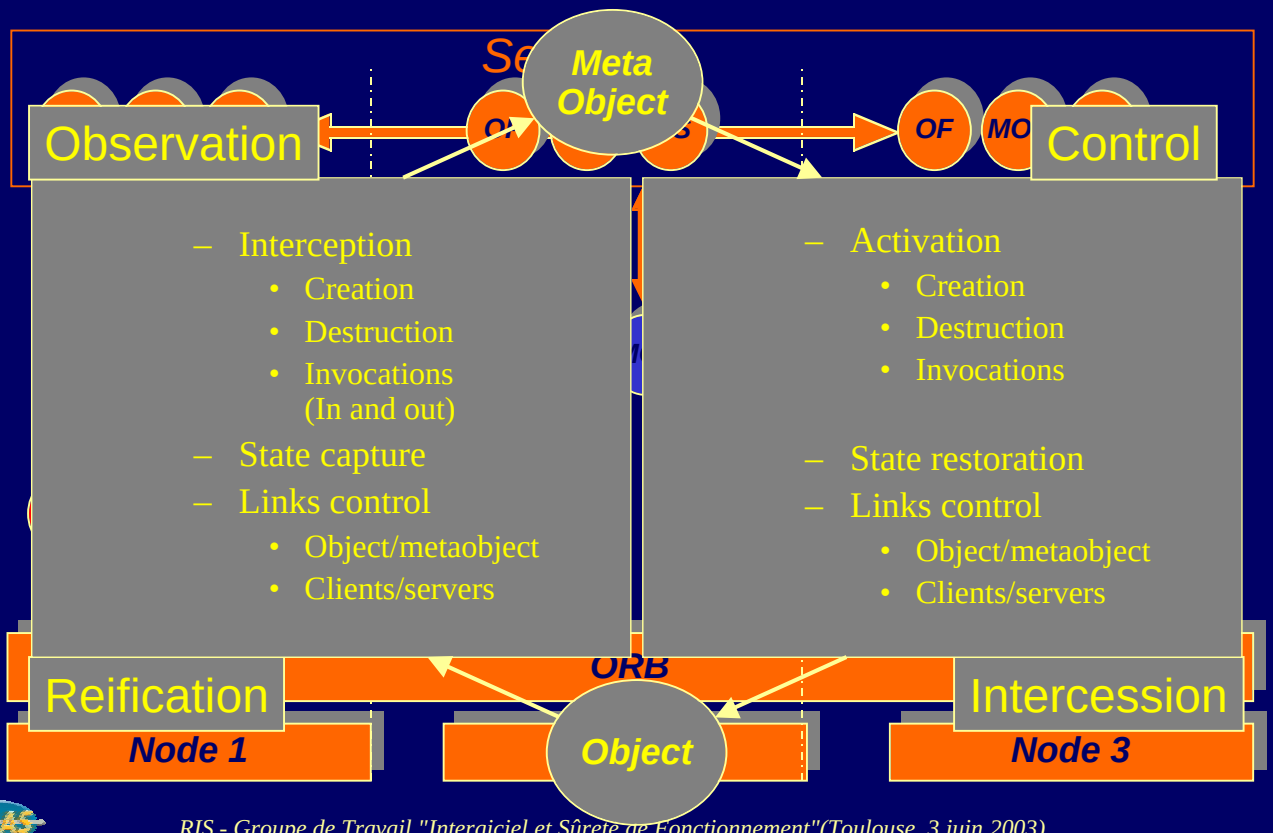
([Running] = th_a ∧ th_b ∈ [$ReadyQueue_{prio(thb)}$]))



Example

Based on reflective languages and open compilers

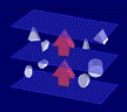
FRIENDS V2



Work in Progress: Context

- **OpenORB (Lancaster University)**
  - Component based, highly adaptive CORBA middleware
  - Goals: Static and dynamic configurability, Extensibility, Efficiency, Compliance to Standards
  - Approach: Component Technology + Reflection
- **Think (INRIA)**
  - A framework for building component-based operating system kernels
  - Approach: kit composed of components, interfaces, naming, binding…
- **Friends (LAAS-CNRS)**
  - Architecture and components for building fault tolerant systems
  - Approach: (Multi-level) Reflection, Open compilers and validation

Work in Progress: Topics



- **Lightweight component models**
  - Combination of existing models at Lancaster and Inria
  - Appropriate description of components & interactions
- **Reconfiguration and cross-cutting concerns**
  - Fault isolation, state & control capture
  - Fault tolerance and dynamic reconfiguration
- **Deep middleware and optimisation**
  - Middleware than spans multiple levels, including OS
  - In-lining techniques and other optimisation techniques.