Some Software Technologies for Resilient Computing

Jean-Charles FABRE

LAAS-CNRS

Université de Toulouse

OSD Conference, Tokyo, Japan, March 7, 2012

Resilient computing

Resilience is defined as **"the persistence of dependability when facing changes"**.

JC Laprie, "From Dependability to Resilience", IEEE/IFIP International Conference on Dependable Systems and Networks, Anchorage, Alaska, USA, June 2008.

Resilient Computing vs Open System Dependability?

– Same motivation and same objectives….

Mastering changes and dependability properties…

- architecture and development process enabling changes
- technical means and tools for performing changes in a dependable way

– OSD encompasses openness and stakeholders implication (elicitation, consensus, etc.) together with supporting tools…

Resilient computing

- **Why it is important today?**
  - Most systems have to adapt and their adaptation cannot be determined a priori because of requirements evolutions, environment changes, etc.
  - Dependability properties should be maintained whatever the evolution is, functional or non-functional,
  - A functional modification may have an impact to its non-functional counterpart
- **True in the large…. but also in the small?**
  - Service Oriented Architecture*: "…. Evolution and openness is a fact…."*
  - Avionics: *"…. remote maintenance and partial evolution is essential…."*
  - Automotive: *"…. customization of embedded functions is of interest…."*

Core Concepts for Resilient Computing

– Componentization

Graph of autonomous components at built time, runtime

– Separation of concerns

Application software independent from FT software

– Architectural concepts

The architecture should enable control over a components graph

– Development process

Design for adaptation is a key concept & mapping to components

Software technologies and dimensions

- SOA and Web services

  Towards Resilient Computing in the large

- Component-based software engineering

  Architecting Resilient Computing Systems

- Aspect Oriented Programming

  Programming Resilient Computing Systems

From coarse grain …. to fine grain, from off-line… to on-line adaptation

Software technologies and dimensions

✔ SOA and Web services

Towards Resilient Computing in the large

✔ Component-based software engineering

Architecting Resilient Computing Systems

- Aspect Oriented Programming

Programming Resilient Computing Systems

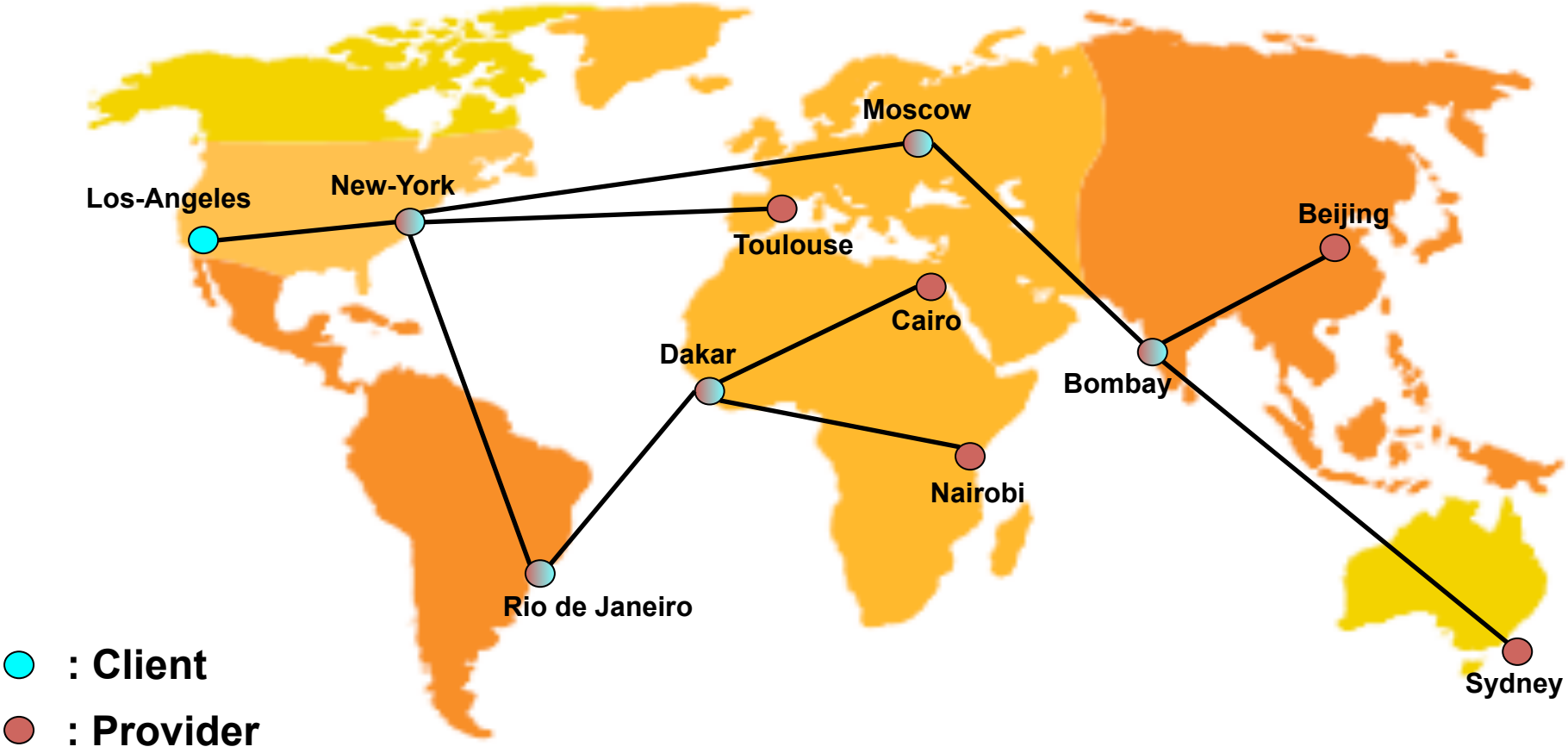
From coarse grain …. to fine grain, from off-line… to on-line adaptation

SOA and Web Services

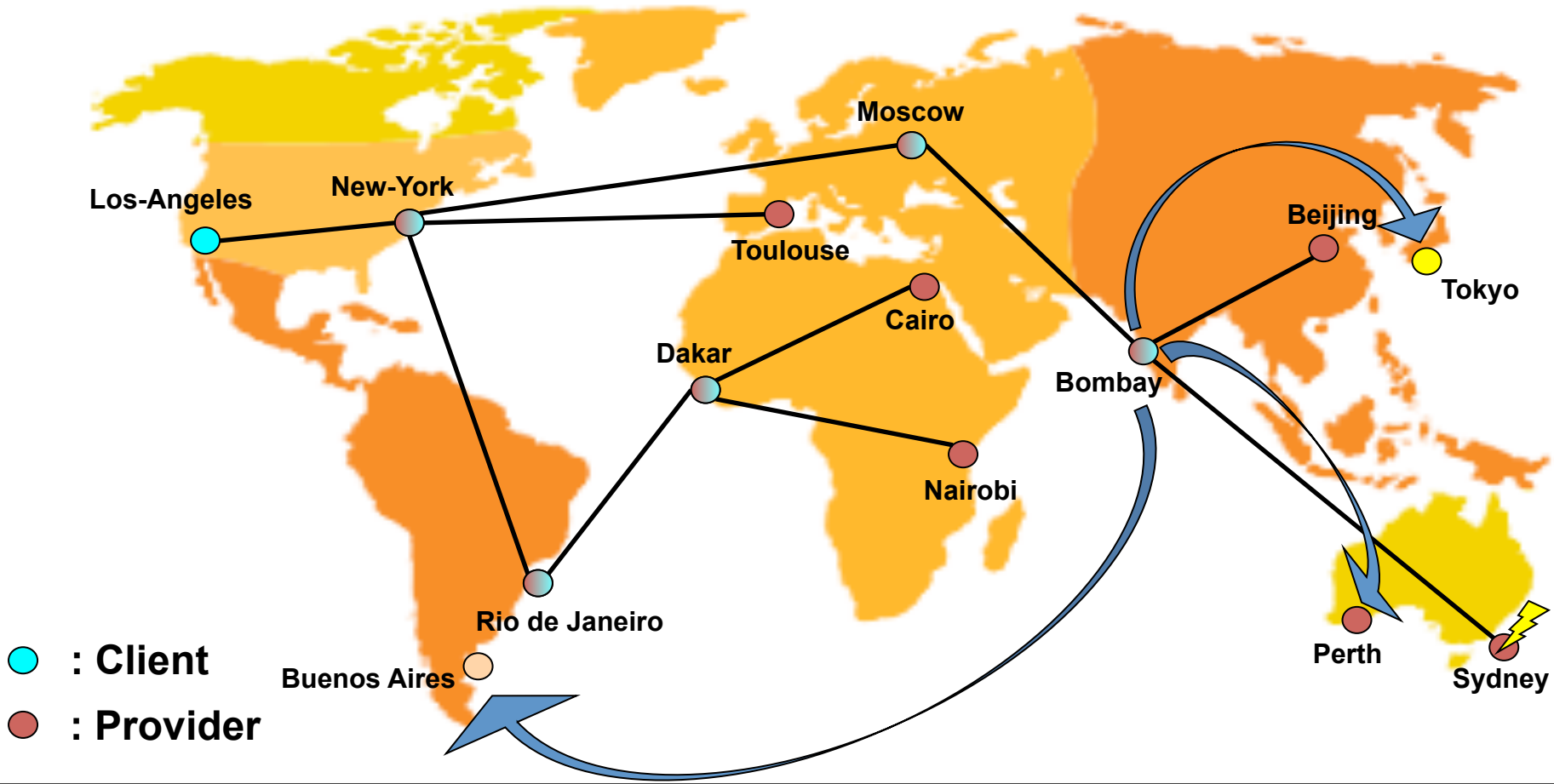
Fault Tolerance Connectors for Unreliable Web Services

Supported by ReSiST, Network of Excellence of the EC, *Resilience and Survivability in IST* (Project n°026764).

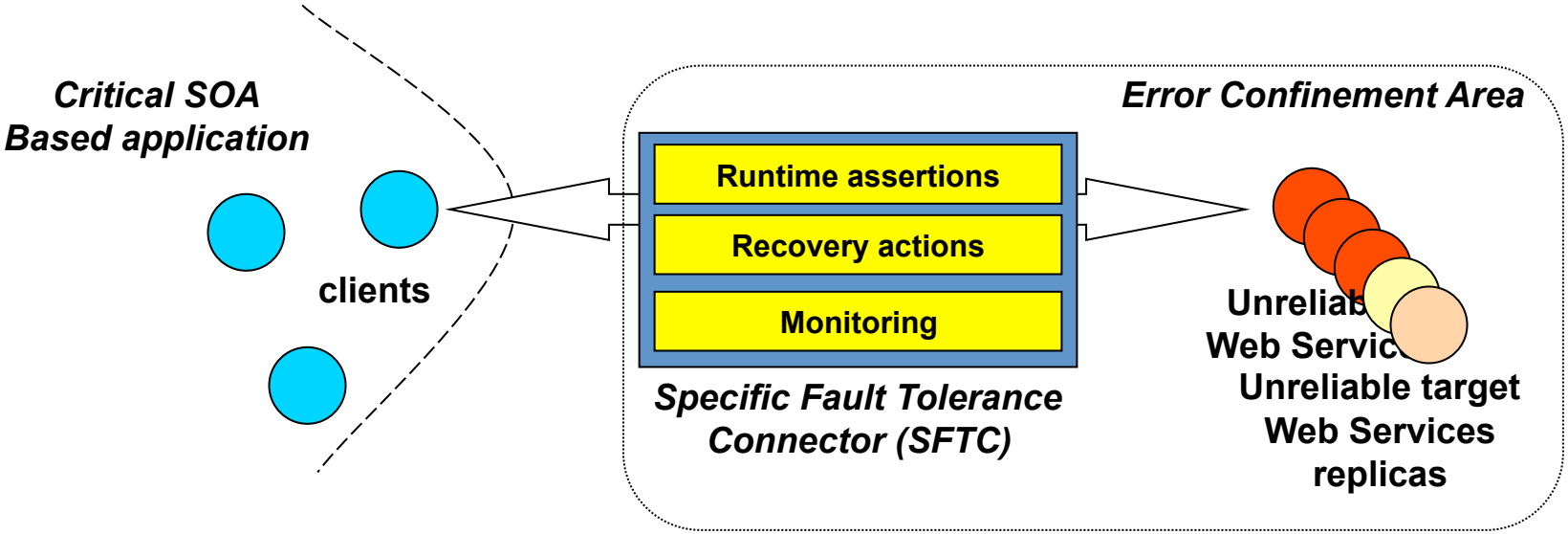
Service Oriented Architectures (SOA)



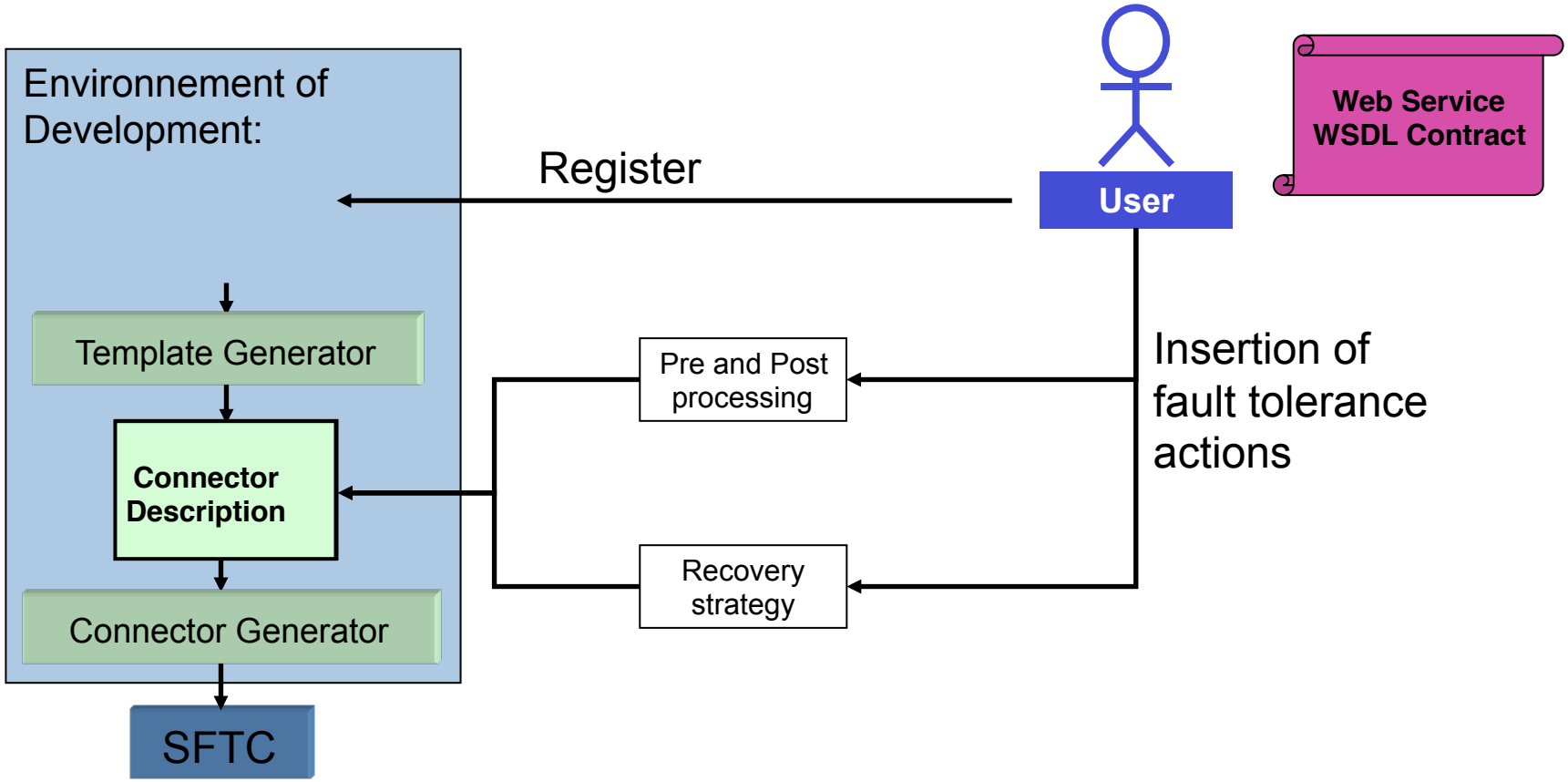
Service Oriented Architectures (SOA)



Specific Fault Tolerant Connector



Development of connectors



Development of connectors

A Domain Specific Language, DEWEL

```
SpecificFaultToleranceConnector myConnector extend Service.wsdl {
  type paramR   operation1(type param1, type param2, ..., type paramN) {
    RecoveryStrategy:
      // Definition of recovery strategy – Example:
      StatefulReplication("Rep1", "Rep2", "Rep3");
    Pre-Processing:
      // Assertion on service request
      return operation1;
    Post-Processing:
      // Assertion on service response
      return operation1;
    CommunicationException ce:
      // Communication error handler
      raise ce;
    ServiceException se:
      // Service failure
      raise se;
  };
  type paramR   operation2(type param1, type param2, ..., type paramN) {..}
  ...
  type paramR   operationN(type param1, type param2, ..., type paramN) {..}
}
```

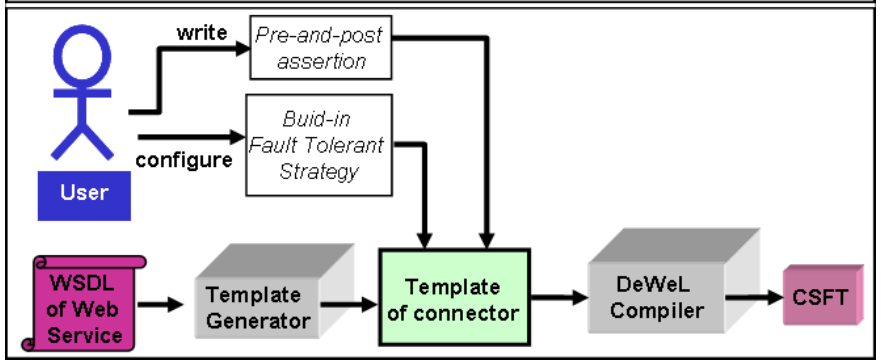
User-selected Fault-Tolerant Strategy

User-defined pre-processing assertion

User-defined post-processing assertion

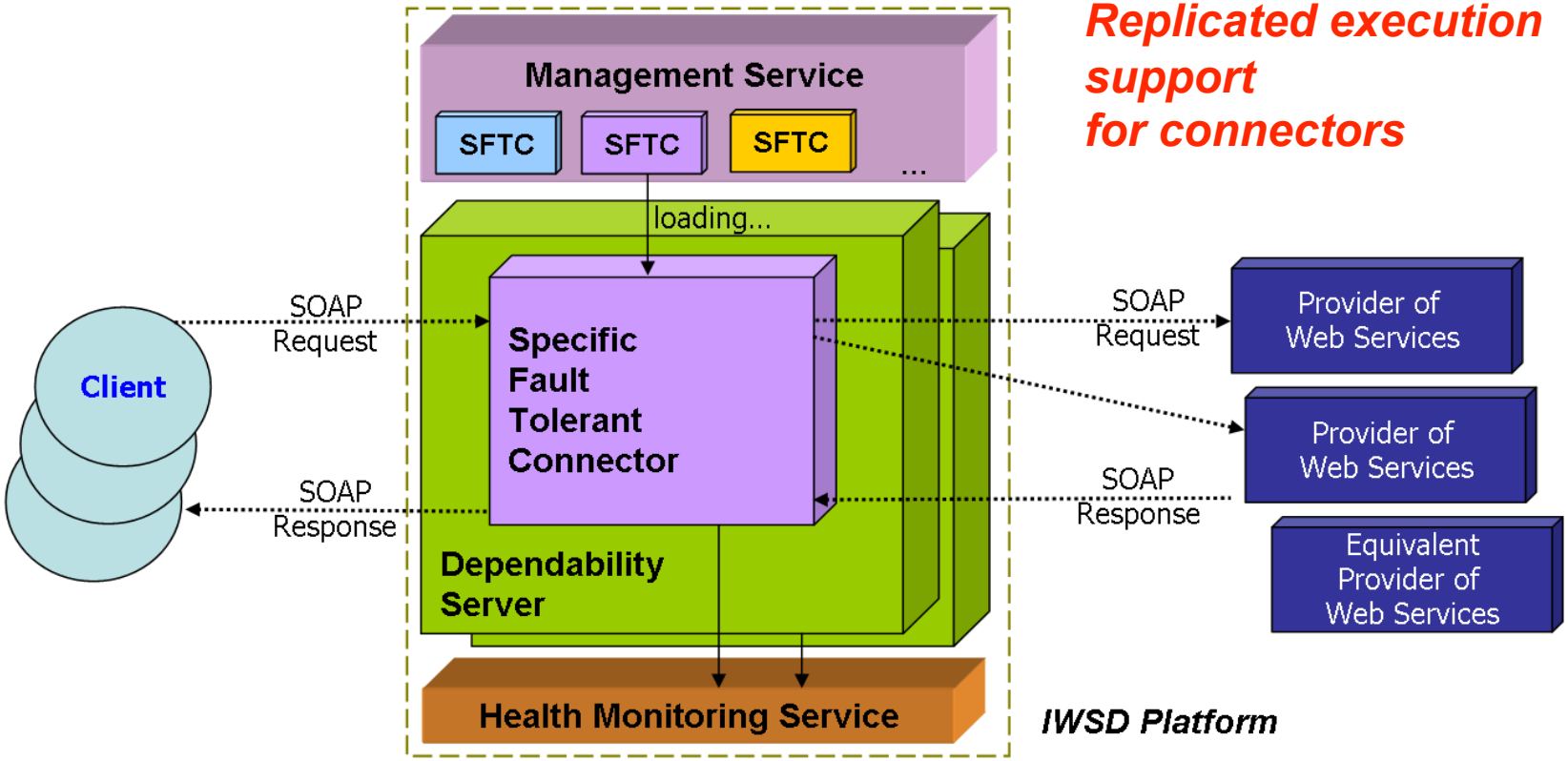
Unreachable service provider handler

Wrong service reply handler



The IWSD Platform

An Infrastructure for Web Service Dependability

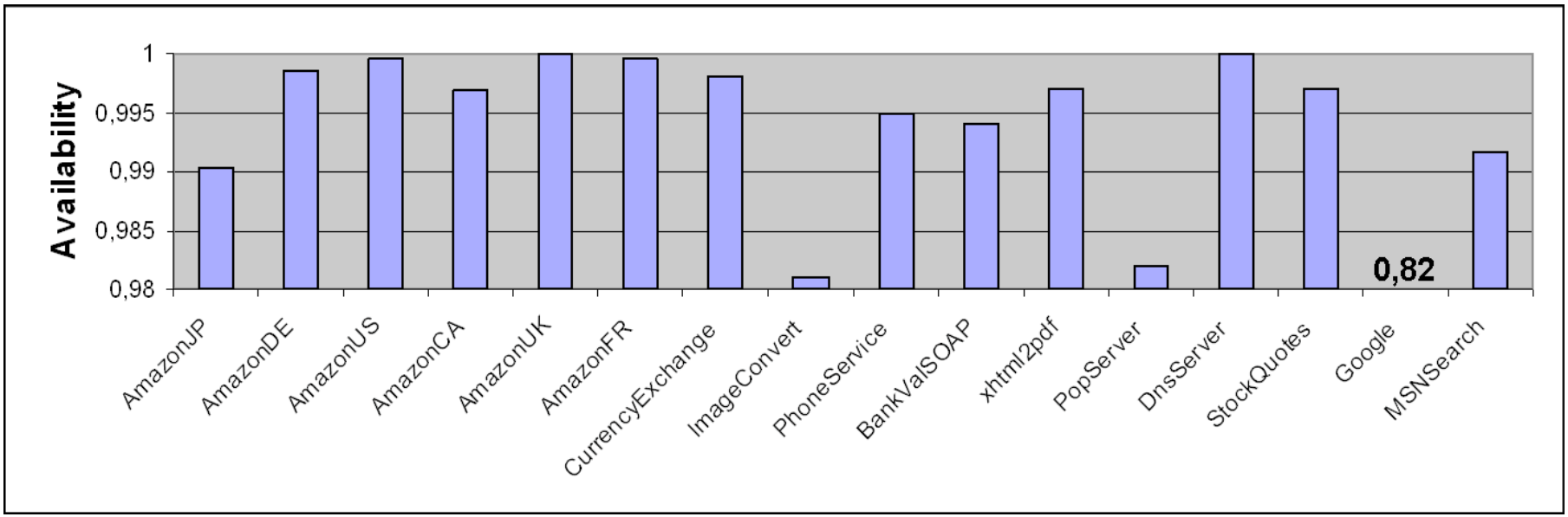


Replicated execution support for connectors

Measurements provided by connectors

- Objectives:
  Collect operational information (availability, response time, failures rate, etc.)

- Example of experimental results :
  2000 requests sent on each Web Service



Synthesis

- ***Resilient computing*** relies on WS (*WSDL*), redundant implementations and the notion of a specific connector (*FTM*) :
  - *Error Detection*: runtime assertions (pre and post-processing).
  - *Error Signaling*: user-defined exceptions raised to the client
  - *Error recovery*: built-in (parameterized) recovery strategies
  - *Monitoring/Diagnosis*: behavior of connector and Web Services

- ***Supporting Tools***
  - A DSL (*DeWeL*) and its development tools to prevent (limit) the occurrence of software faults in connectors
  - A platform (*IWSD Framework*) as a dependable execution support, including management and monitoring services for connectors and Web Services

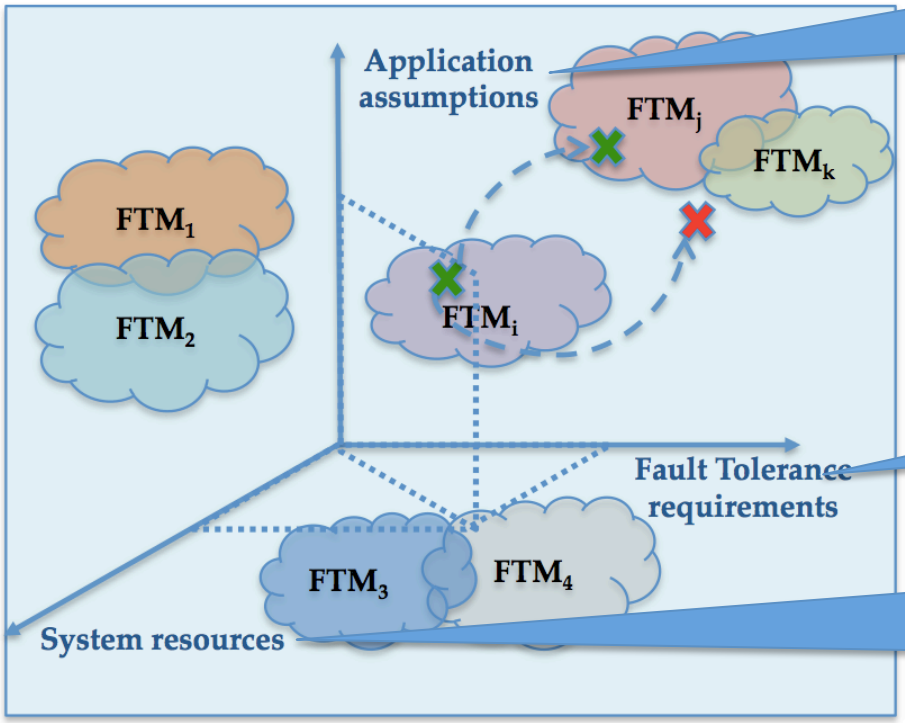
Nicolas Salatge, Jean-Charles Fabre, "Fault Tolerance Connectors for Unreliable Web Services". DSN 2007: 51-60 (IEEE *Xplore*)

Component-based software engineering

Design for adaptation and architecting resilient systems

Supported by MURPHY, a project funded by the French Science Foundation (ANR) on Dependable Sensor Networks (project n° ANR-BLAN-SIM/10-LS-100618-6-01).

Context-aware fault tolerant computing

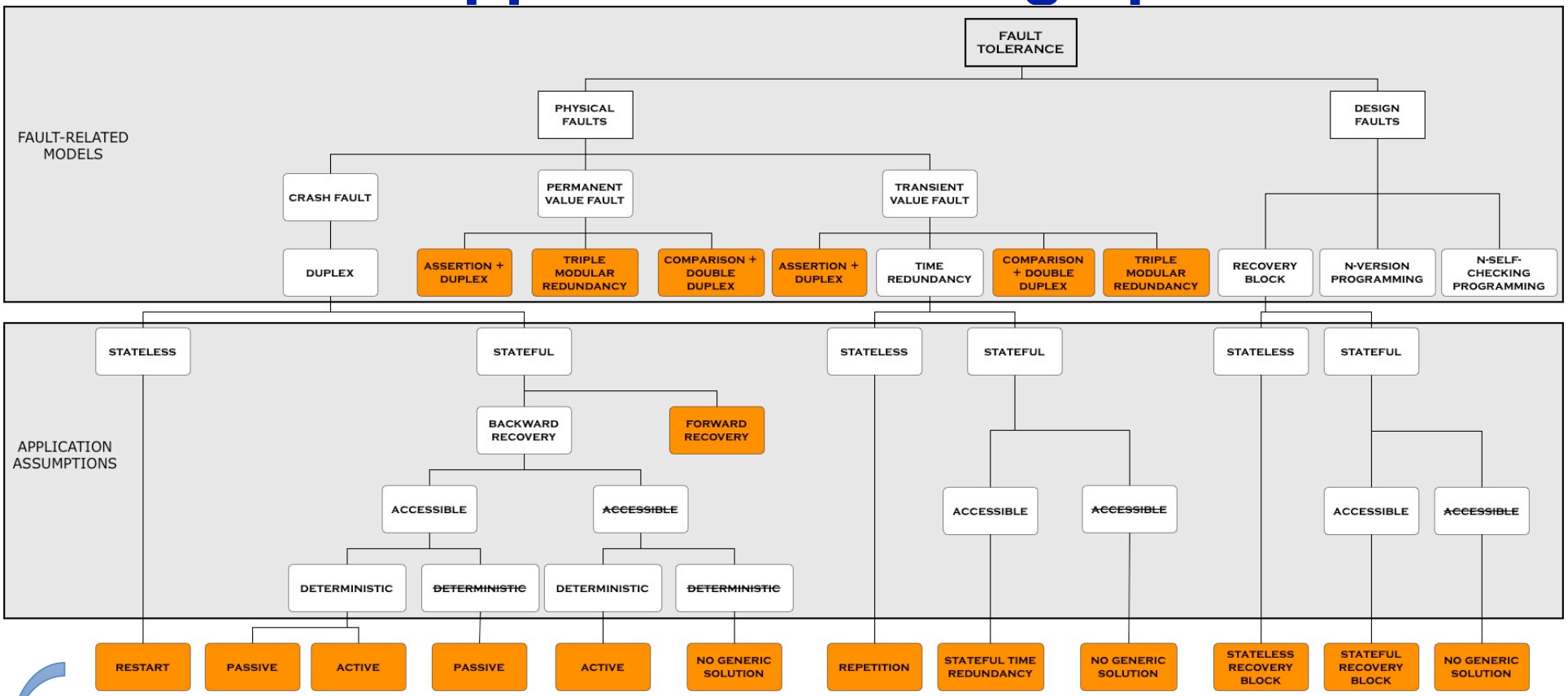


Change Model Frame of Reference

- The system status is represented in a 3-axis space
  - Stateless/stateful
  - Accessible or not
  - Deterministic or not
- System evolution w.r.t multivalued axes

  ⇔Trajectory
  - Crash fault
  - Value fault
  - Transient fault
  - …
- Each FTM covers a region (acceptable bounds)
- At a given point in time, an application is attached to an adequate FTM
  - No of processors
  - Bandwidth
  - Memory
  - Battery life
- Leaving a « validity space » implies changing the FTM!

Overall approach: FT design patterns



Resource requirements/usage/performance issues and acceptable bounds
Cost function
- Deterministic parameters (e.g. number of sites)
- Application-dependent parameters (state)

} **Order relation between solutions**

Development process

FTM Design Pattern development :

UML design tool and C++ implementation

Tools enabling runtime reconfiguration:

component-based middleware: FraSCAti

Transition algorithms:

FScript + script execution engine

Monitoring service :

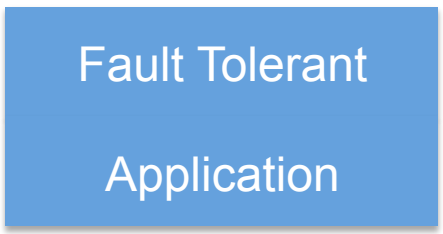
resource availability and property verification

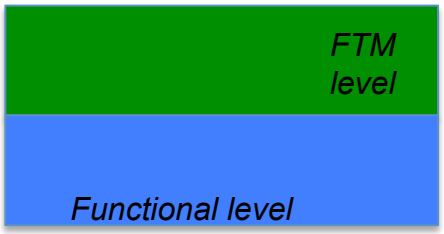
Design for adaptation of FTMs

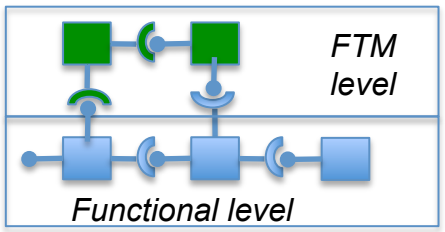
- Adaptation in mind in the design
- A tool box of reusable FT building blocks
- Minimizing component state

Componentization of FTMs

- **Decomposition for adaptation of the FT software**

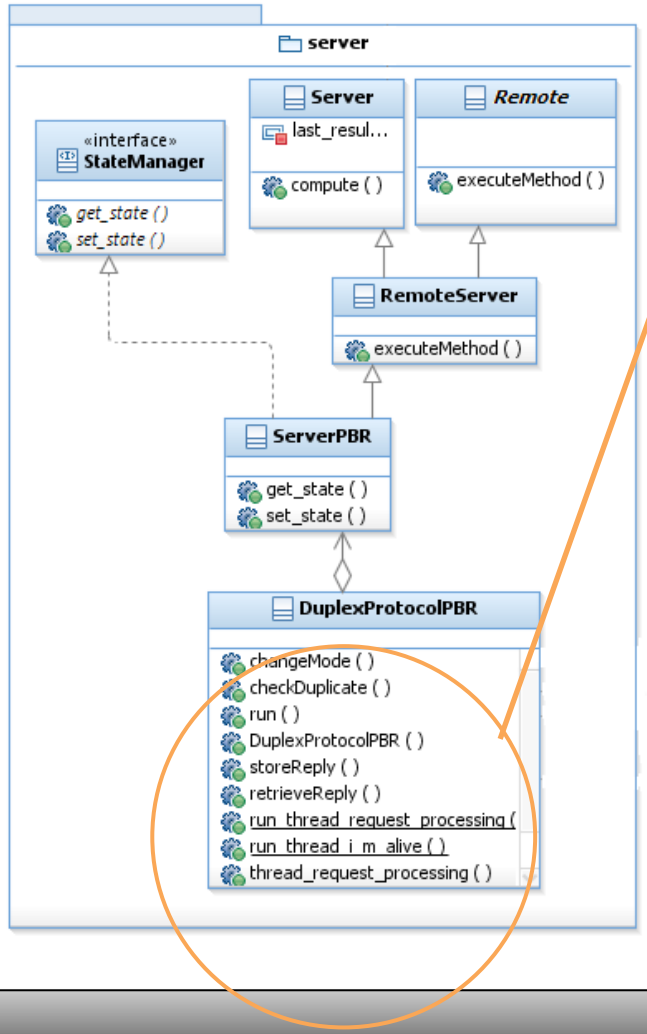






- **CBSE middleware support – Reflective Middleware**
  - *Observe and modify* the state of the components
  - *Observe and control* the interactions between components at runtime (dynamic binding)
  - *Observe and modify* the component architecture (creation, destruction, insertion and removal of components)

Design for adaptation: Initial design



Crash fault model

- Communication support
- Request processing
- Check duplicate
- Inter-Replica Protocol for PBR

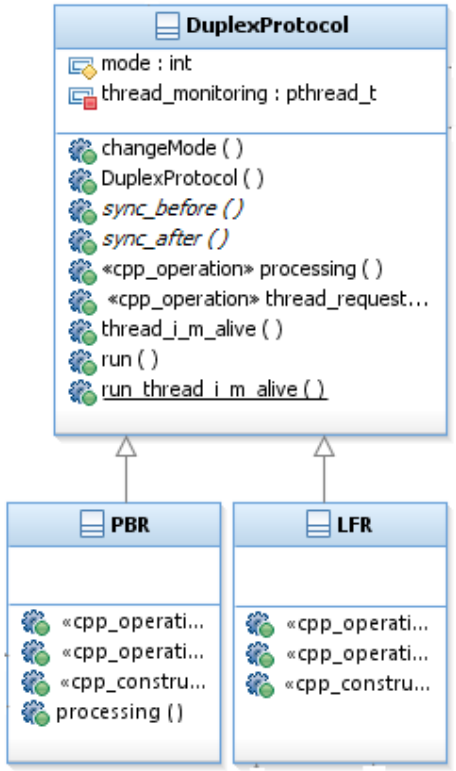
Mixed concerns

New protocol development

⇔

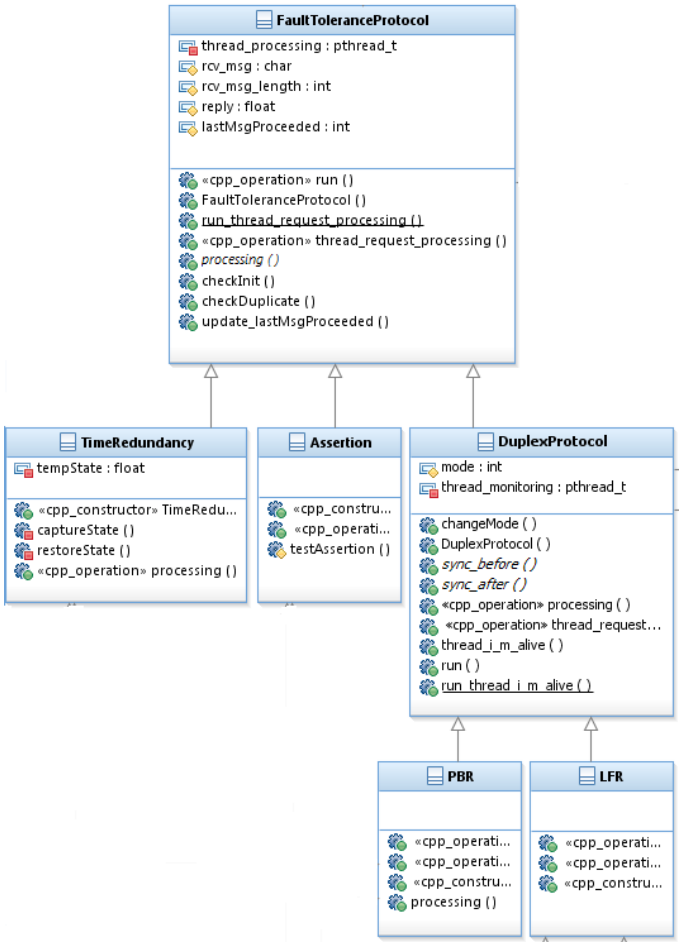
new «DuplexProtocol*» class

Design for Adaptation through Iteration



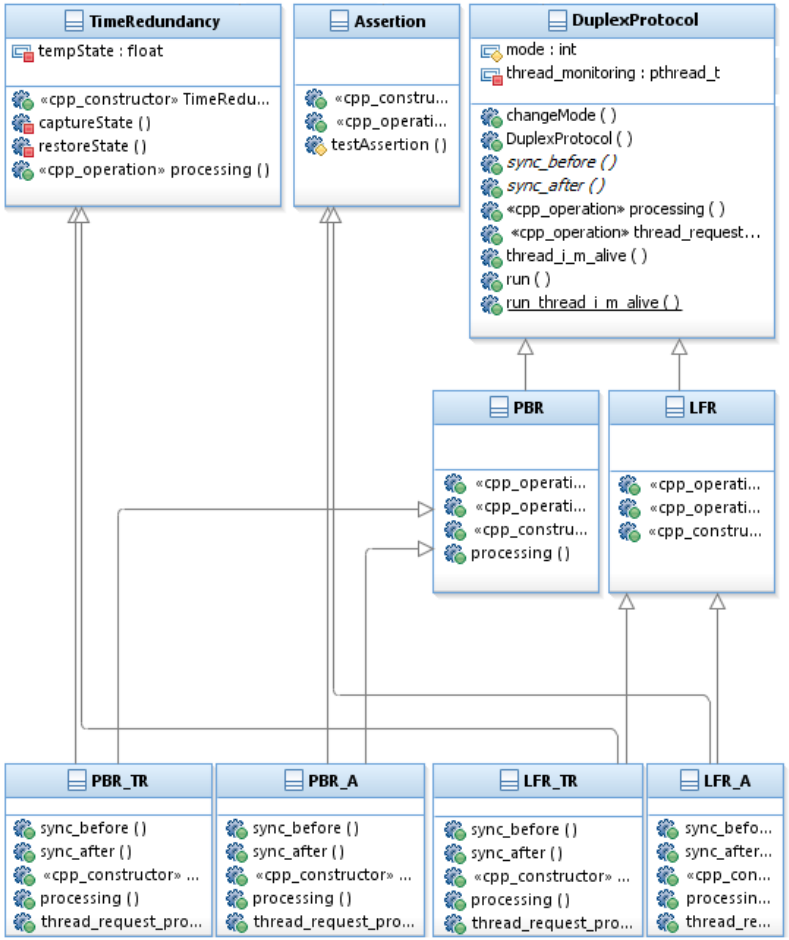
- **Step 1:**
  - Externalise common features of all Duplex Protocols
  - Break the Inter-Replica Protocol into:
    - * *sync_before*
    - * *processing*
    - * *sync_after*
  - Evolutions
    - ⇒ Facilitate duplex variants development (e.g. LFR), no code duplication
    - ⇒ new fault model (transient value fault) => Combination of a duplex protocol pattern and the time redundancy pattern

Design for Adaptation through Iteration



- **Step 2:**
  - Externalise request processing and the communication with the client
  - Implementation of Time Redundancy (transient value fault)
  - Easy composition of Fault Tolerance Protocols

Design for Adaptation through Iteration

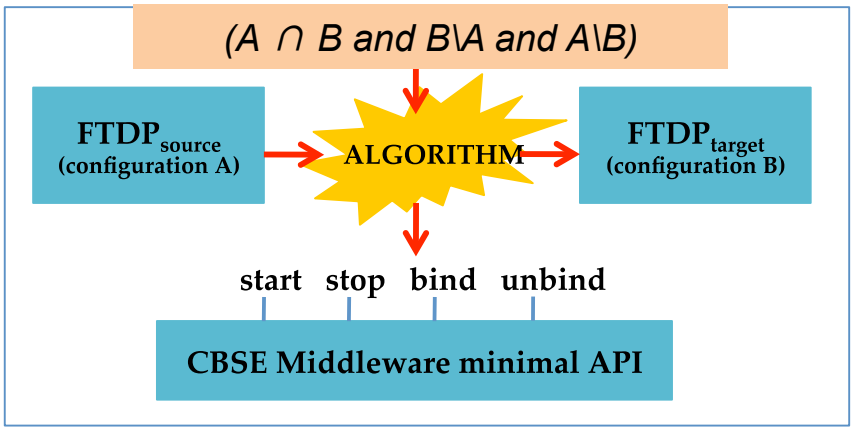


- Facilitate protocol composition
- Composed protocols ⇔ fault model extension
- Fine-grained classes:
  - facilitate mapping to components on a CBSE middleware
  - Minimizes state handling during software updating

On-line adaptation

FTM variants and transitions algorithms are defined off-line (outer loop)

Transition algorithms are run on-line



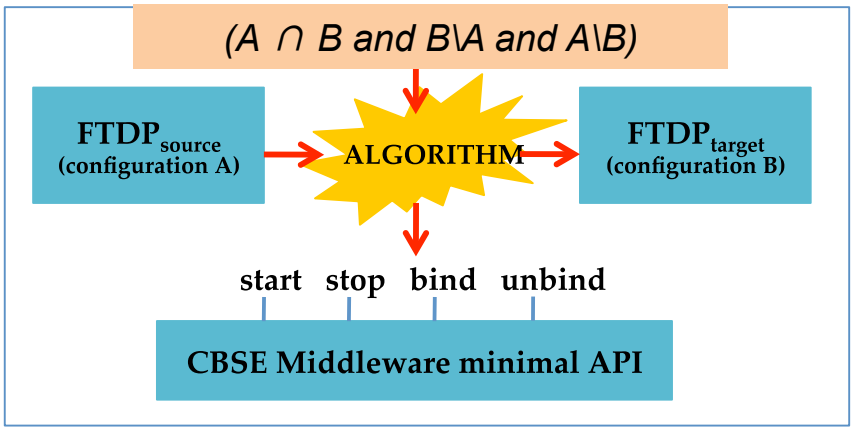
A minimal API can be implemented on a CBSE middleware providing SW actuators on component lifecycle

- **CBSE middleware in action**
  - Stopping a component
    - All requests in progress must be processed
    - Queuing input requests
    - Triggering component state storage
  - Starting a component and triggering its initialization
  - Binding and unbinding components

On-line adaptation

FTM variants and transitions algorithms are defined off-line (outer loop)

Transition algorithms are run on-line



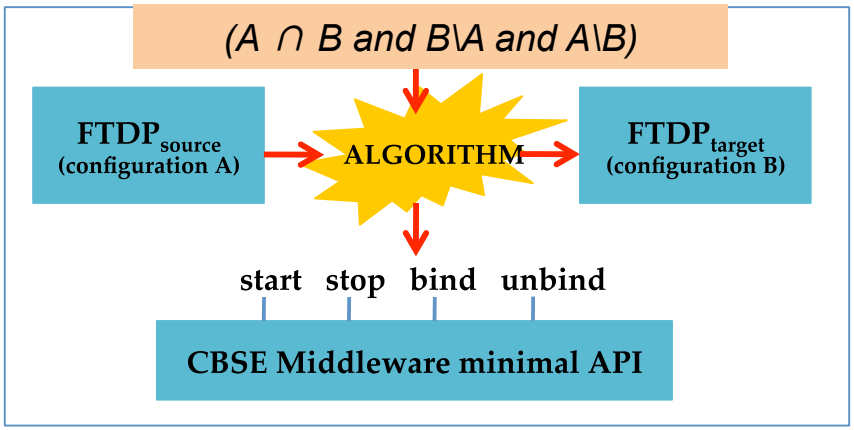
A minimal API can be implemented on a CBSE middleware providing SW actuators on component lifecycle

- **Transition algorithm template**
  - Identification of components to be changed
  - Script definition off-line
    - Stop components in a suitable adaptation state *(locks)*
    - Unbind components
    - Install new components
    - Bind components
    - Start components

On-line adaptation

FTM variants and transitions algorithms are defined off-line (outer loop)

Transition algorithms are run on-line



A minimal API can be implemented on a CBSE middleware providing SW actuators on component lifecycle

Experiments in progress

- Scenario
  - *T0: crash fault model for A*
    Passive replication
  - *T1: cash fault model for A*
    Semi active replication
  - *T2: cash+transient fault model for A*
    Semiactive replication
    + Time redundancy
- Working steps
  - Mapping design to components
  - Few components to be changed
  - Minimal state, easy transfer function
  - Perform the updates at runtime
- Frascati middleware support
  - The script can be tested by hand through a interactive interface
  - Script definition can be tuned before running

Synthesis and working steps

- Design for adaptation is essential
  - UML based design for FTM design patterns
  - Code generation in C++ for proof of concepts
  - Practical evaluation on a simple case study
  - Feedback to the design following adaptation criteria
- Experiments
  - Experiments on the Frascati CBSE/SCA middleware
  - adhoc implementation of a component runtime support for Smart Sensors
- Transitions between FTMs is very much assisted by the CBSE runtime support (suspend/activate, bind/unbind,….)

M.Stoicescu , J.C.Fabre , M.Roy, « Architecting Resilient Computing Systems: Overall Approach And Open Issues », Int. Workshop On Software Engineering For Resilient Systems (SERENE 2011), Geneva (Switzerland), Septembre 2011, pp.48-62

Conclusion

Concluding remarks

- Many SE technologies are available to ***implement resilient computing*** in the large but also in the small.
- In large systems, resilient computing is mandatory since individual elements (e.g. web services) have unknown dependability properties and change in an uncontrolled manner.
- Component-based software engineering helps developing resilient systems
  - Design time
    - Design and analysis of FT mechanisms wrt frame of reference
    - Designing FTMs for adaptation and easy mapping to components
  - Runtime
    - Identification of a middleware support to perform modifications on-line
    - Definition of transition algorithms for runtime adaptation
    - Synchronization of changes to ensure consistent behaviour