
 Laboratory for Analysis and Architecture of Systems Toulouse
France

Continuously Evolving Systems: Towards On-Line Adaptation of Fault Tolerance Software

Jean-Charles Fabre



National Polytechnic Institute, University of Toulouse, France

Evolvability

The European Network of Excellence ReSIST
(<http://www.resist-noe.org/>)
defines evolvability during operation as :

“the ability to dynamically adjust system behaviour and potentially its functional architecture and deployment to cater for new operational contexts, including operational faults and attacks, new threats, and dynamic changes to the system environment”.

Resilient Computing

- Resilience is defined as “**persistence of dependability when facing changes**”.

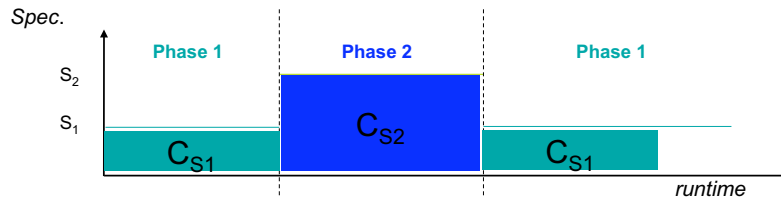
JC Laprie, “From Dependability to Resilience”, IEEE/IFIP International Conference on Dependable Systems and Networks, Anchorage, Alaska, USA, June 2008.

- These changes can be classified according to several dimensions:
 - the nature, functional, environmental, technological at both the hardware and software level;
 - their prospect considering new hardware platforms but also new threats, foreseeable or not;
 - the timing, from seconds to hours, even month for large scale systems;

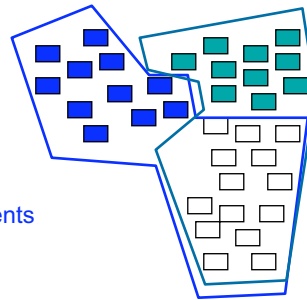
Context: Evolution of dependable systems in operation

- Specifications evolution
 - Application services and consequently structural and behavioural properties may change over the life time of the system (*successive versions*)
 - Fault tolerance software specifications may change and/or include additional requirements during operational life (*e.g. fault model, state handling issues*)
- Runtime environment
 - The configuration of the distributed infrastructure may evolve (*accidental loss of resources, configuration changes or additional resources available*)
 - Performance requirements (*e.g. response time*) must be fulfilled in operation in accordance with resource usage (*e.g. network bandwidth*), and these may vary according to various operational phases

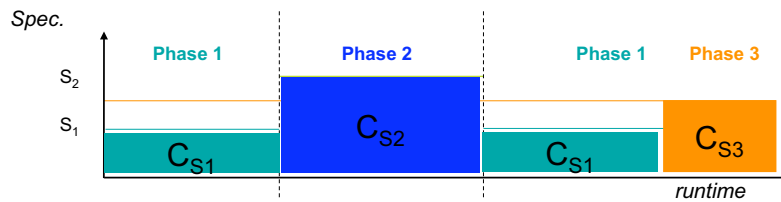
Evolution scenario: an example



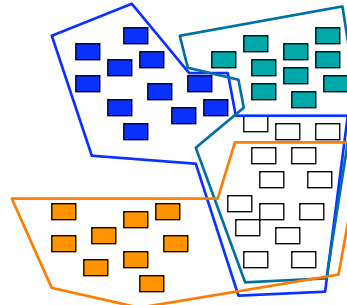
- **Software configuration: A+FTM**
 - Functional application software (A)
 - Fault Tolerance Mechanism (FTM)
- **Configurations vs operational phases**
 - Spec S1: soft. config. C_{S1}
 - Spec S2: soft. config. C_{S2}
- **FTMs share several services / components**
 - $CS1 \neq CS2$
 - $CS1 \cap CS2 \neq \emptyset$



Evolution scenario: an example



- **Traditional approaches**
 - All FTMs defined a priori
 - A triggering parameter P
Switch (P) case
 $CS1$ or $CS2$ or $CS3$;
- **A new configuration**
 - FTM non anticipated a priori
 - Loading additional components



Requirements and technologies

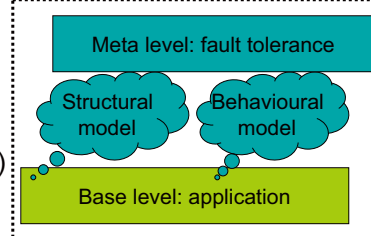
For on-line adaptation of unanticipated FTMs

- Requirements for adaptive FTMs

- Separation of concerns — Independence as much as possible
- Componentization — Fine-grain adaptation
- Runtime component model — Component graph description
- Behavioural model — States & adaptation synchronisation

- Supporting technologies

- Reflective computing / AOSD
- Component-Based Software Engineering (CBSE)
- Modelling techniques



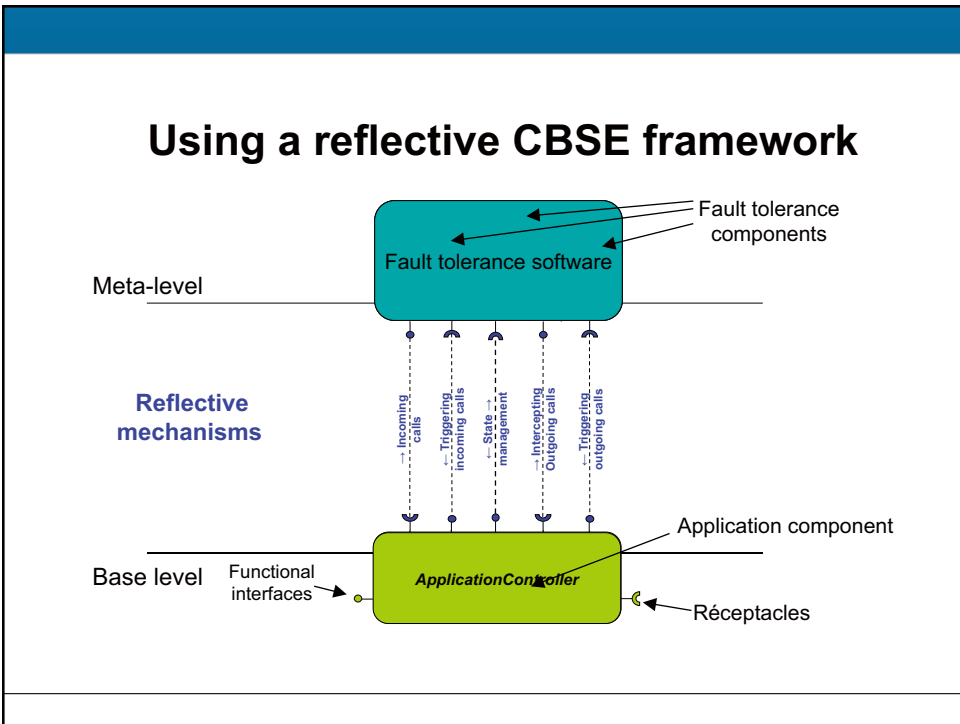
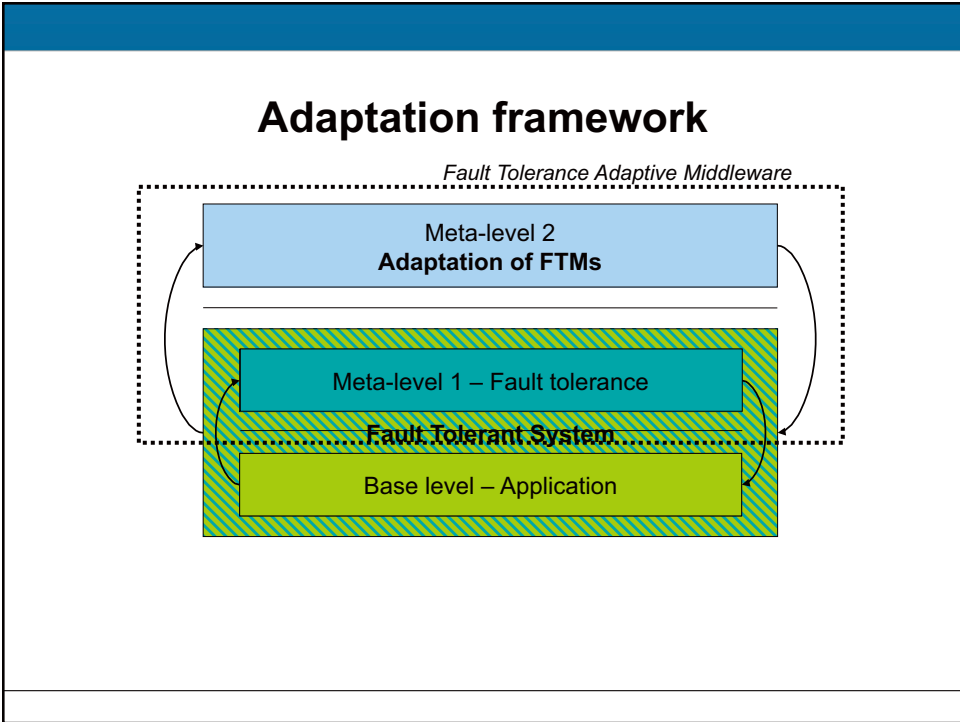
LAAS-CNRS

Laboratory for Analysis and Architecture of Systems

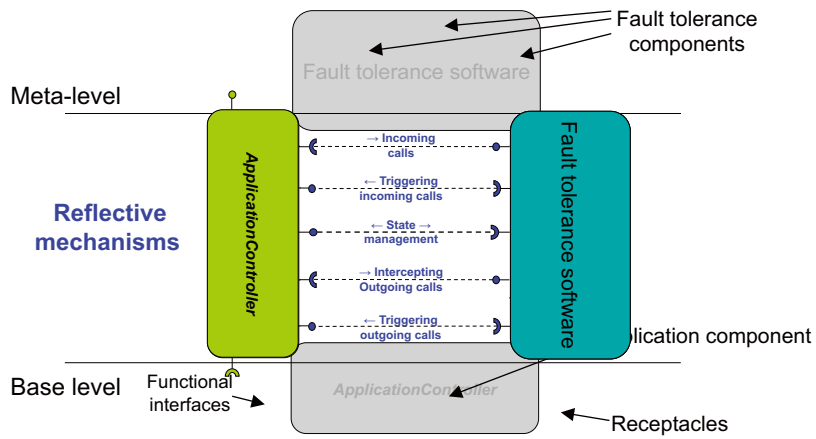
Toulouse
France

Outline

- Context and problem statement
- **Framework and basic elements**
- Definition of Suitable Adaptation States
- Application to a case study

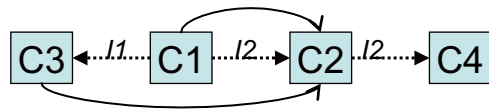


Using a reflective CBSE framework



Software configuration manipulation

Structural model

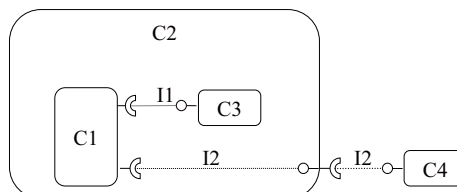


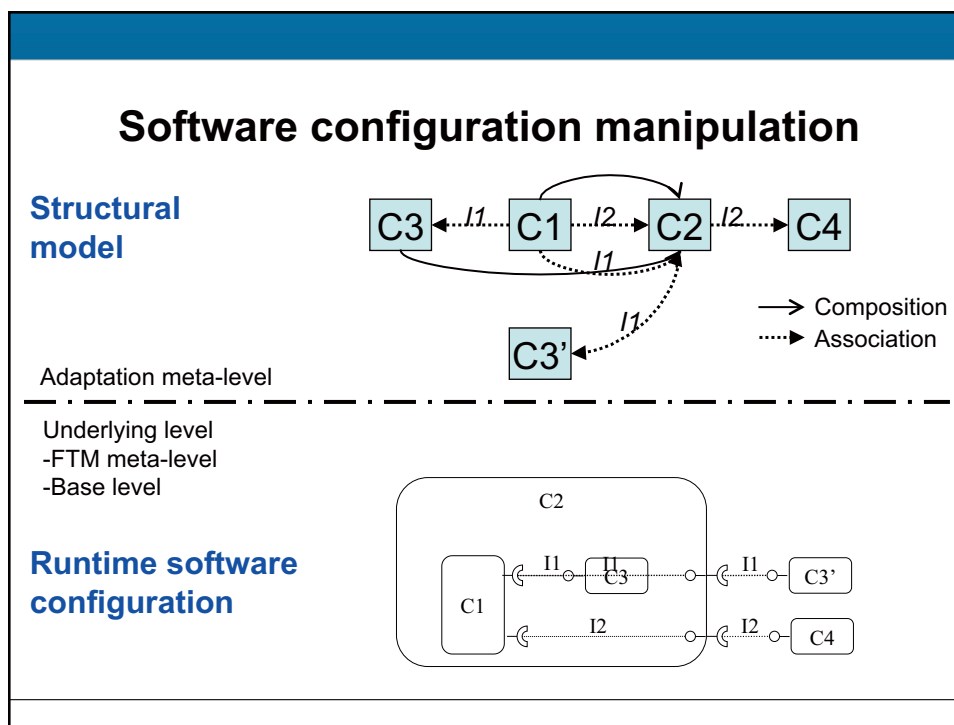
→ Composition
 Association

Meta-level

Base level

Runtime software configuration





Laboratory for Analysis and Architecture of Systems

Toulouse
France

Outline

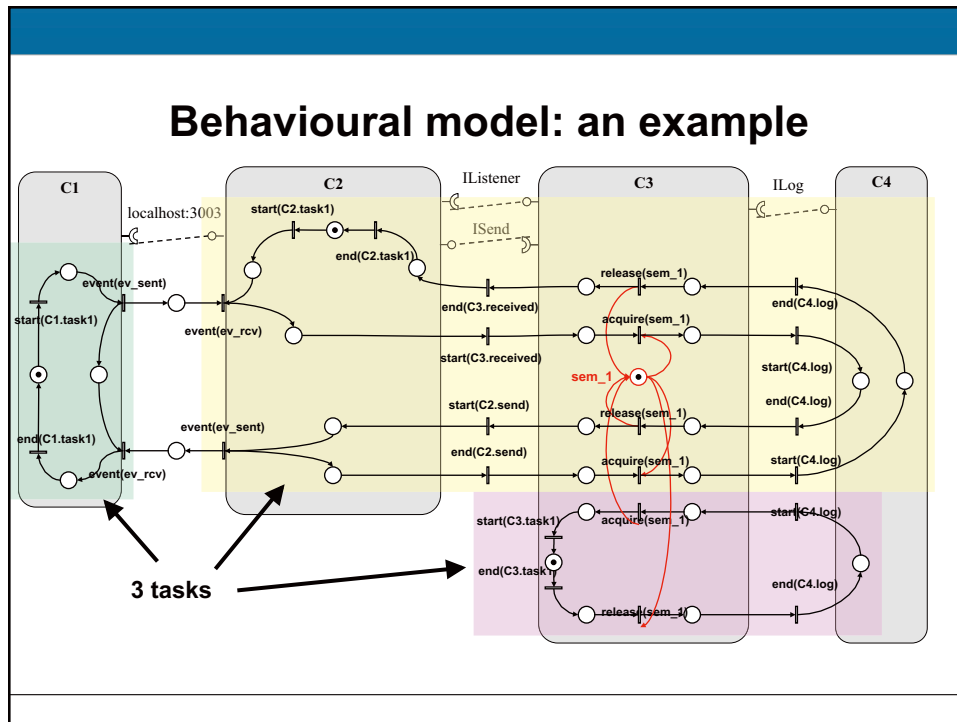
- Context and problem statement
- Framework and basic elements
- **Definition of Suitable Adaptation States**
- Application to a case study

Execution and adaptation properties

- **Isolation**
 - The components to be modified must be suspended and detached from the SC, and new ones initialized
- **Liveness**
 - The adaptation process must not introduce any deadlock or livelock in the current fault tolerant application execution
- **Conformance**
 - The observed behaviour of the fault tolerant application must be consistent w.r.t:
 - i) the specifications before adaptation: a task runs in the former configuration
 - ii) the specifications after adaptation: a task runs in the target configuration
- **Convergence**
 - The fault tolerant application must eventually reach a state where the modification can be done while respecting the above defined properties

Assumptions and Models

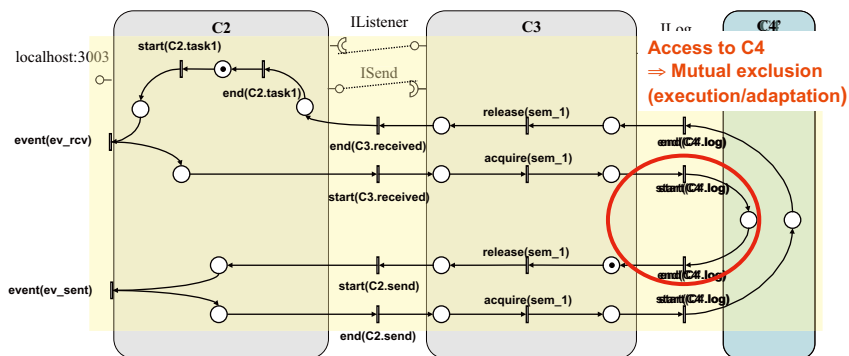
- **Suitable Adaptation State (SAS)**
 - An SAS is defined as a state where adaptation does not violate the previously defined properties
- **System model assumptions**
 - Cyclic tasks, the initial state of a single task being an SAS
 - State mgt functions are user-defined within every component
 - Fine-grain components \Rightarrow stateless or simple state handling
- **Modelling behaviour (e.g. Petri Nets)**
 - On-line tracing of on-going execution
 - Modelling causality and concurrency



Suitable Adaptation States

- SAS for a unique task:
 - Mutual exclusion between the adaptation process and the system tasks (*application tasks going through both application components and fault tolerance components*)
 - Local conformance of a task execution trace w.r.t one configuration (*current configuration → target configuration*)
- SAS for the system:
 - All tasks are within an adaptable state (*before or after adaptation*)
 - Global conformance:
 - causality between two transactions imposing to obey the same specification (*before or after adaptation*)
 - example: a message sent in one configuration must be processed within the same configuration

Impact of modifications and SAS

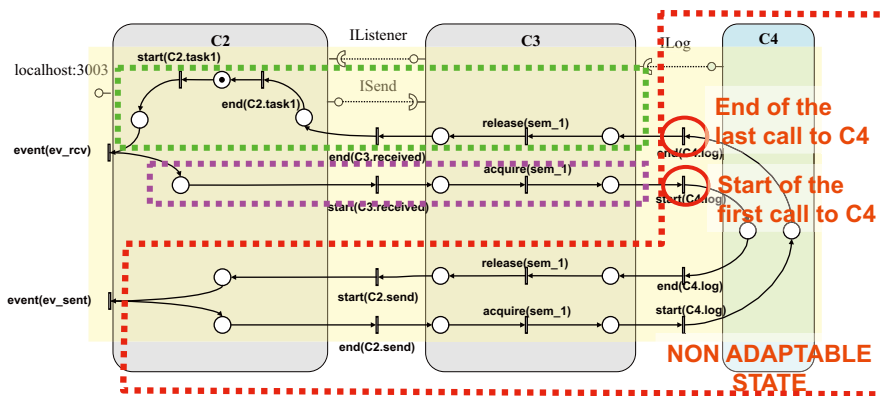


- When C4 can be replaced by C4'?

start(C2.task1), event(ev_rcv), start(C3.received), acquire(sem_1), start(C4.log), end(C4.log),
 release(sem_1), start(C2.send), event(ev_send), end(C2.send), acquire(sem_1),
 start(C4'.log), end(C4'.log), release(sem_1), end(C3.received). end(C2.task1)

Conformance violation ⇔ no specification satisfied

Impact of modifications and SAS



- Execution states classification:

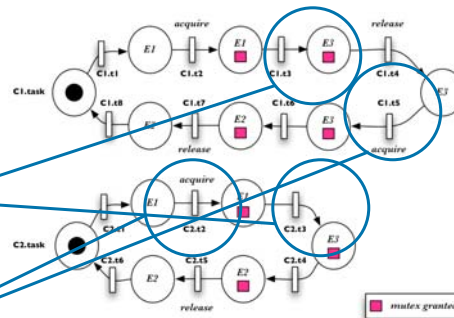
- E1 ⇒ adaptable states w.r.t. S_{after}
- E2 ⇒ adaptable states w.r.t. S_{before}
- E3 ⇒ non-adaptable states

The task has performed actions within components to be modified and will perform actions within newly inserted component

SAS and concurrency control

- **Principle of adaptation locks**
 - Used by the adaptation process to control task execution
 - Stop the task execution to perform the adaptation
- **Durability lock**
 - prevent execution to enter non adaptable state (E3)
 - But....
- **Convergence lock**
 - guiding the system leave E3 and reach an SAS (E1 or E2)
 - So...


Durability warning: A task cannot be locked by the adaptation process when it holds a mutex requested by another task.



Convergence warning: when a task is in a Non adaptable state E3, it has priority to acquire a mutex and leave E3

Implementation steps of the adaptation process

- **Describing software configurations**
 - An ADL can be used to produce XML documents (off-line)
 - New configurations can be developed during the life time of the system
- **Building architectural representations**
 - Use of a CBSE frameworks (e.g. OpenCOM) to manipulate components on-line
 - Simple graph of components representing a configuration to determine the modifications
- **Building behavioural representation**
 - A Petri net is associated with all configurations (off-line)
 - New configuration implies updates of the Petri net representation on-line.
- **Event monitoring**
 - Use of an interception mechanisms: interface interceptors, connection interceptors
- **Controlling execution towards an SAS**
 - Comparison of two configurations before and after
 - Insertion of "contextual" durability and convergence locks at appropriate places
- **Initialization of component state**
 - Activation of user-defined methods to update the state of the newly inserted components
 - Release of the locks

 **LAAS-CNRS**Laboratory for Analysis and Architecture of SystemsToulouse
France

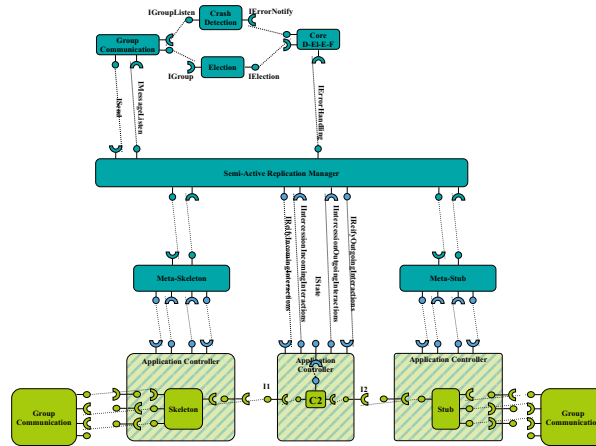
Outline

- Context and problem statement
- Framework and basic elements
- Definition of Suitable Adaptation States
- **Application to a case study**

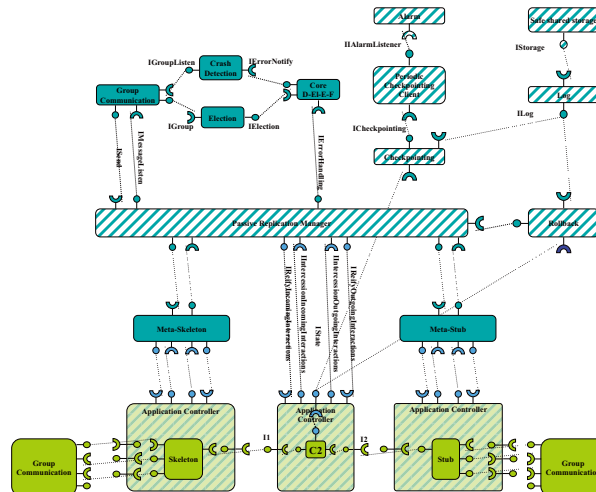
Case study: a simple FT application

- **Objective**
 - Simple case study as first proof of concepts
 - Application of the overall approach, analysis and feasibility
- **Application**
 - Automatic control application of an inverse pendulum on a cart
 - Two processors running an identical copy of the control software
 - Sensors and actuator:
 - Sensors capture the current position of the pendulum and the cart
 - An actuator delivers the acceleration figure to electrical engines
- **Fault Tolerance**
 - Crash fault assumption of a processor, the pendulum must not fall
 - Two variants of a duplex strategy: semi-active and primary backup replication protocol

Semi-Active Replication



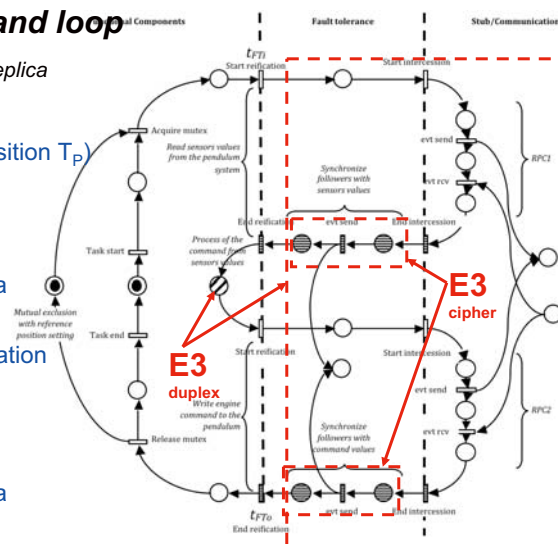
Primary Backup Replication



Behavioural model of the control-command loop

(simplification of real RdP of a replica
few locks to control adaptation)

- Periodic task (target position T_P)
- RPC1 : read sensor C_p
- Synchronize with replica
- Computation of acceleration
- RPC2 : write command
- Synchronize with replica



LAAS-CNRS

Laboratory for Analysis and Architecture of Systems

Toulouse
France

Conclusion

Concluding remarks

- **Systematic analysis of the adaptation process**
 - Separation of concerns: application, FTMs, adaptation
 - Properties: isolation, convergence, liveness, conformance
 - Structural and behavioural modelling available on-line
- **Lessons learnt**
 - Very positive use of CBSE technologies / reflective framework
 - Design for adaptation / Variability of SAS
 - Core components updates ⇒ large “non adaptable state”
 - Side components or additional FTMs ⇒ small “non adaptable state”
 - Fine-grain adaptation on-line, preventing stopping the whole system and loading a new one introduces some complexity ☹ ... tradeoffs ☺!
- **Next**
 - Engineering of adaptation (framework, process, application, cost)
 - Proactive fault tolerance vs a reactive approach

The end

Thank you!