Towards On-Line Adaptation of Fault Tolerance Mechanisms

Jean-Charles Fabre, Marc-Olivier Killijian and Thomas Pareaud

INPT-ENSEEIHT    CNRS    EADS-ASTRIUM Satellites

University of Toulouse, France
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
    
    $C_{S1}$ or $C_{S2}$ or $C_{S3}$;

- **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
  - Componentization
  - Runtime component model
  - Behavioural model

- Supporting technologies
  - Reflective computing
  - Component-Based Software Engineering (CBSE)
  - Modelling techniques

Independence as much as possible
Fine-grain adaptation
Component graph description
States & adaptation synchronisation
Meta level: fault tolerance
Structural model
Behavioural model
Base level: application
Outline

• Context and problem statement

• Framework and basic elements

• Definition of Suitable Adaptation States

• Application to a case study
Adaptation framework

Fault Tolerance Adaptive Middleware

Meta-level 2
Adaptation of FTMs

Meta-level 1 – Fault tolerance

Fault Tolerant System

Base level – Application
Using a reflective CBSE framework

Meta-level

Fault tolerance software

Fault tolerance components

Reflective mechanisms

Base level

Fault tolerance software

Functional interfaces

Application component

Réceptacles
Using a reflective CBSE framework

Meta-level

Reflective mechanisms

Base level

Functional interfaces

ApplicationController

Fault tolerance software

Fault tolerance components

→ Incoming calls

← Triggering incoming calls

→ State management

→ Intercepting outgoing calls

← Triggering outgoing calls

Receptacles

Application component
Software configuration manipulation

Structural model

Meta-level

Base level

Runtime software configuration
Software configuration manipulation

Structural model

Adaptation meta-level

Underlying level
- FTM meta-level
- Base level

Runtime software configuration
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 ⇒ stateless or simple state handling

• **Modelling behaviour: Petri Nets**
  – On-line tracing of on-going execution
  – Modelling causality and concurrency
Behavioural model: an example

3 tasks
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 (E1 or E2)
  – 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 ⇒ adaptables states w.r.t. $S_{after}$
  - E2 ⇒ adaptables 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 to 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 new 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
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
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!