Some Challenges in Adaptive Fault Tolerant Computing

F. Taiani & JC. Fabre
Lancaster Univ. (Lancaster, UK), LAAS-CNRS (Toulouse, F)
The Problem

- Evolution implies adaptation, e.g.
  - rapidly changing operational conditions
  - evolving requirements
  - evolving threats
The Problem

- Evolution implies adaptation, e.g.
  - rapidly changing operational conditions
  - evolving requirements
  - evolving threats

- **Dependability** mechanisms must evolve:
  - to meet new user requirements.
  - to provide same level of dependability under different conditions
The Problem

- Evolution implies adaptation, e.g.
  - rapidly changing operational conditions
  - evolving requirements
  - evolving threats

- **Dependability** mechanisms must evolve:
  - to meet new user requirements.
  - to provide same level of dependability under different conditions

Our goal: **facilitate** the design, analysis and implementation of adaptable fault-tolerant systems
Requirements and properties

- **Design-level** requirements for adaptation
  - separation b. functional / non-functional concern
  - adaptable software structures
  - appropriate runtime modelling
  - on-line assessment of execution & dependability
Requirements and properties

- **Design-level** requirements for adaptation
  - separation b. functional / non-functional concern
  - adaptable software structures
  - appropriate runtime modelling
  - on-line assessment of execution & dependability

- **Adaptation mechanisms**: ideal properties
  - minimal impact on system execution (isolation)
  - consistent w/ (i) past activity (ii) new requirements
  - ensure dependability of:
    - new software configuration
    - adaptation process itself
Our (advocated) Strategy

- **Reflection** for separation of concerns between
  - the functional level
  - fault-tolerance
  - adaptation itself.

- **Components** for programmability
  - framework rather than specific / on-dimensional scenario
  - principled high-level abstractions
  - encapsulate best practices adaptation and fault tolerance

- Scope control using **fine-grained decomposition**
  - simple tasks only incur simple costs
ASAP Framework

- A Reflective architecture separating:
  - Fault tolerance mechanisms
  - Adaptation of FT application
  - System attributes assessment

- Architectural principles
  - Fine-grain decomposition to minimize adaptation cost
  - Adaptation Triggers ⇔ (i) Adverse operational context, (ii) Side effects of application software modifications
Fault tolerance software design

- Decomposition for adaptation of the fault tolerant Software using CBSE
A Smart update example

[Laas, City]
SWOT Analysis

- **Strengths**
  - reflective computing and component technologies

- **Weaknesses (so far)**
  - statically defined set of configurations (closed world)

- **Opportunities**
  - novel software technologies (CBSE, AOP)…

- **Threats**
  - on-line assessment & validation of dynamic systems
  - performance overheads
What the next step?

- Qualitative sensitivity analysis of approach features
  - complexity of runtime modelling, performances, etc.

- Better model-based control of execution
  - automation of model construction [Pareaud 2009]
  - more powerful in particular considering timing issues
  - hierarchical models to master complexity

- State
  - adaptable state wrt to liveness and safety properties?
  - mastering and handling the capture/ recovery of state of the computation
To Conclude: Challenges (1/2)

1. Design for adaptation
   ➔ Fine grain components

2. Synchronisation of modifications
   ➔ Runtime modelling of execution

3. Mastering distributed state
   ➔ How to capture semantic?

4. On-line assessment
   ➔ How to assess impact of adaptation on dependability?

5. On-line adaptation
   ➔ Reactive (traditional)
   ➔ Or proactive (much harder)
Thanks

Question?