Some Software Technologies for Resilient Computing

Jean-Charles FABRE

OSD Conference, Tokyo, Japan, March 7, 2012
Resilient computing

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


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

Critical SOA Based application

clients

Specific Fault Tolerance Connector (SFTC)

Runtime assertions
Recovery actions
Monitoring

Error Confinement Area

Unreliable Web Service
Unreliable target Web Services replicas
Development of connectors

Environnement of Development:

- Template Generator
  - Connector Description
  - Connector Generator
  - SFTC

Register

User

Web Service WSDL Contract

Insertion of fault tolerance actions

- Pre and Post processing
- Recovery strategy
Development of connectors

A Domain Specific Language, DEWEL
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](https://www.w3.org/TR/wsdl)), redundant implementations and the notion of a specific connector ([FTM](https://en.wikipedia.org/wiki/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](https://develoque.com/)) and its development tools to prevent (limit) the occurrence of software faults in connectors
  – A platform ([IWSD Framework](https://iwsd-framework.org/)) as a dependable execution support, including management and monitoring services for connectors and Web Services

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

- The system status is represented in a 3-axis space.
- System evolution w.r.t multivalued axes.
  - Crash fault
  - Value fault
  - Transient fault
  - Trajectory
- Each FTM covers a region (acceptable bounds).
- At a given point in time, an application is attached to an adequate FTM.
- Leaving a « validity space » implies changing the FTM!
Overall approach: FT design patterns

<table>
<thead>
<tr>
<th>Resource requirements/usage/performance issues and acceptable bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost function</td>
</tr>
<tr>
<td>• Deterministic parameters (e.g. number of sites)</td>
</tr>
<tr>
<td>• Application-dependent parameters (state)</td>
</tr>
<tr>
<td>Order relation between solutions</td>
</tr>
</tbody>
</table>

- Faust Tolerance
  - Fault-related models
    - Crash fault
      - Duplex
        - Assertion + duplex
        - Triple modular redundancy
        - Comparison + double duplex
    - Permanent value fault
    - Transient value fault
      - Time redundancy
      - Comparison + double duplex
      - Triple modular redundancy
  - Design faults
    - Recovery block
    - N-version programming
    - N-self-checking programming
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 FTM

- 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

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 $\Leftrightarrow$ 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*

- **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

---

A minimal API can be implemented on a CBSE middleware providing SW actuators on component lifecycle
On-line adaptation

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

Transition algorithms are run on-line

(A ∩ B and B\A and A\B)

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

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

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

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