Using Component Redundancy for adaptive, self-optimising and self-healing Component-Based Systems

Ada Diaconescu, John Murphy
Performance Engineering Laboratory
Dublin City University
Main Topics

• Targeted domain
• Motivation
• Our approach – component redundancy
  – What is component redundancy?
  – Example
  – How does component redundancy work?
  – Framework implementation
• Conclusions & future work
Targeted domain – Enterprise software applications

- We are targeting the business logic tier of enterprise software applications
- Quality characteristics - influenced by all tiers and layers involved
Motivation

• Enterprise software applications - Characteristics:
  – Complex, large-scale
  – Highly distributed and parallel
  – Non-real time, Soft quality requirements (performance, reliability)
  ⇒ Complicated & expensive design, testing, management processes
  ⇒ Reduced flexibility
  ⇒ Quality characteristics hard to control

• Component-Based Software Development (CBSD):
  – Benefits: modularity, reuse, shorter development time, lower costs
  – New challenges: lack of information
    • At component development: ?overall system, platform, resources?
    • At system integration: ?component insight information, changing resources/ requirements at runtime?

• Impossible to exhaustively test such software apps. offline
Component redundancy – what is?

- Multiple Software Component Variants, with:
  - Identical interfaces, Equivalent functionalities (i.e. offered services) and
  - Different design and/or implementation strategies are available at run-time
- Only one component variant is active at all times
  - instantiated for handling client requests -
- Variants are used alternatively, ‘complementing’ each other
- Variants are replaced in case of:
  - Poor/ non-optimal performance
  - Fault detection
  - Changing requirements, or running-context
Example

- Used EJB component technology
- Two component implementations:
  - Same functionality: retrieve information from a remote DB
  - Different design: Direct DB vs. Using Entity Bean

Response-time variations with Network load on the link to the DB

⇒ Alternating variants yields better performance, at all times
Component redundancy – how it works

**Monitoring**
- Monitor application
- Monitor environment
- Determine problems:
  - Causes
  - Affected components

**Evaluation**
- Determine optimal variant(s)
- Use:
  - Monitoring info
  - Descriptions
  - Decision policies
- Update descriptions
- Update decision policies

**Action**
- Activate / deactivate component variants
- Maintain application integrity

---

Using Component Redundancy for adaptive, self-optimising and self-healing Component-Based Systems

Ada Diaconescu
diacones@eeng.dcu.ie
Distributed adaptation mechanism

- Motivation: centralised adaptation mechanisms might:
  - Introduce unnecessary overhead
  - Not scale well
- Adaptation mechanisms with different scopes:
  - Component
  - Group of components
  - Entire application
- Hierarchical organisation
- Local problems:
  - Initially dealt with locally
  - Signalled to higher level adaptation mechanisms (if necessary)
- Periodic global optimisations
Framework Implementation

- Independent of specific applications
- Two options:
  
  a) Distributed platform level
  
  b) Software application level

- Maintain application integrity:
  
  - Component swapping implemented by means of client call indirection
  
  - No state transfer
  
  - Keep client references consistent using the proxy pattern
Conclusions & future work

• Component-based enterprise software
• Difficult to provide and maintain performance and dependability:
  – Lack of information
  – Changing requirements and execution contexts
• Our approach: using component redundancy (overview, general framework)
• Expected benefits:
  • Automatic performance optimisation
  • Recover from and avoid integration faults
  • Adapt to changing requirements, resources, workloads
• Future work:
  • Identify and implement relevant examples
  • Design and implement proof-of-concept framework
  • Identify and integrate work on monitoring, component descriptions, knowledge based management,…
Using Component Redundancy for adaptive, self-optimising and self-healing Component-Based Systems