A Framework for Using Component Redundancy for Self-Adapting and Self-Optimising Component-Based Enterprise Systems

Ada Diaconescu

Performance Engineering Laboratory

The Problem:
managing performance of complex enterprise systems

Complex systems are built by assembling (COTS) *components* and using *middleware* technologies

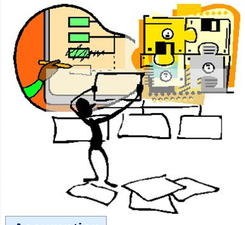
Problem factors:

- Lack of information on:
  - Component internal design and implementation (strategies)
  - Component running environment (e.g. workload, resources)
- Complexity:
  - Large scale applications (i.e. many components involved)
  - Dynamic nature of applications (dynamic inter-component binding, changing workloads and resources)

Problem:

The **emerging system performance** is **hard to analyse and predict**

Systems and their environment constantly change at runtime, making performance management a continuous task



Assumption

Different design and implementation strategies yield optimal performance (e.g. response time) in different execution contexts (e.g. workload)

Example:
validity of Assumption

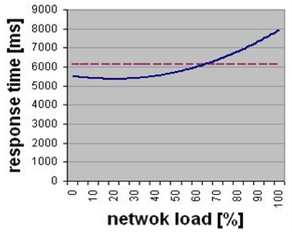
Functionality:

- repeated retrieval of information from a remote DB - used Enterprise Java Beans (EJB) technology

Two design strategies (two variants):

Stateless session beans only (direct access to the DB, for each request)

Session façade – session bean uses entity bean (local caching of data from DB)



Tested each variant for various amounts of available bandwidth on network link to remote database

Each strategy proves optimal in different running contexts

knowledgeably alternating the usage of redundant components, optimised for different running contexts, ensures better overall performance than either component variant could provide

Our Goal:
automating performance management

Enable component-based applications to automatically change their implementation at runtime, in order to:

- Self-optimise
- Adapt to changes in the system and in the running environment

Requirements for meeting our goal - the following must be provided:

1. Alternative design and implementation strategies available at runtime (1)
2. A mechanism for automatically alternating these strategies at runtime, and reach applications' high-level quality goals (2)

Our solution:

a framework (2)

that uses component redundancy (1)

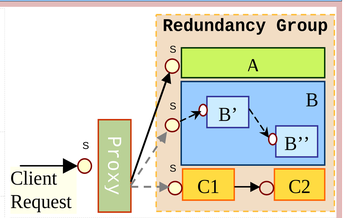
to automate performance management

Component Redundancy

Definition:
Availability at runtime of multiple component variants, providing identical or equivalent services, each variant with a different design and implementation strategy

Only one component variant providing a service is used, at any time, to handle a certain client request for that service

Aim:
have each component variant optimised for a different execution context (e.g. incoming load, usage patterns, available resources)



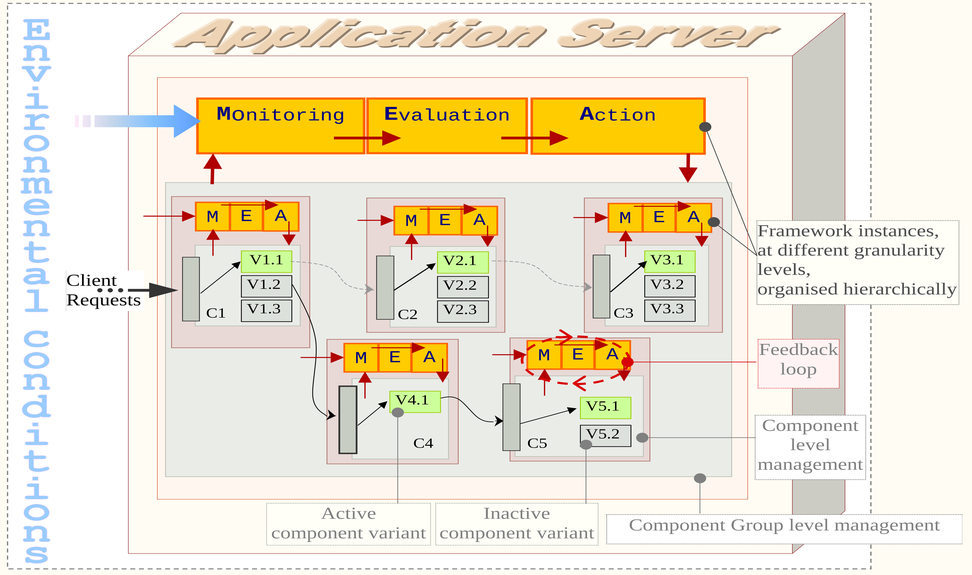
A, B and C: component variants providing service S

Our Framework: three-tiered architecture

Three tiers: *Monitoring*, *Evaluation* and *Action* operate in a feedback-loop manner:

- application performance is **monitored**;
- application is **evaluated** and optimisation decisions taken;
- redundant component variant(s) identified as optimal are activated (**action**); *and*
- the resulting application is (re-)monitored and (re-)evaluated

Features: - Determine and activate optimal individual component variants, at component level
- Determine and activate optimal combinations of variants, at component assembly (or application) level



Main contribution: Unique Framework Features

- Decentralised framework:
  - Framework instances manage running application at different granularities (e.g. individual component, component group, entire application)
  - Framework instances exchange quality requirements and feedback information
  - Emerging behaviour of collaborating framework instances results in globally optimised application
  - Framework instances at different granularities can be switched on and off to reduce unnecessary overhead

Benefits of framework decentralisation:

- Increased scalability and reduced management overhead (e.g. local problems can be solved locally)
- Increased flexibility

- Learning-based approach (rather than based on static information)
  - Framework does not impose that component providers/deployers provide initial performance information on components (variants)
  - Framework is designed to collect performance information, learn in time about the component based application it has to manage and dynamically improve itself

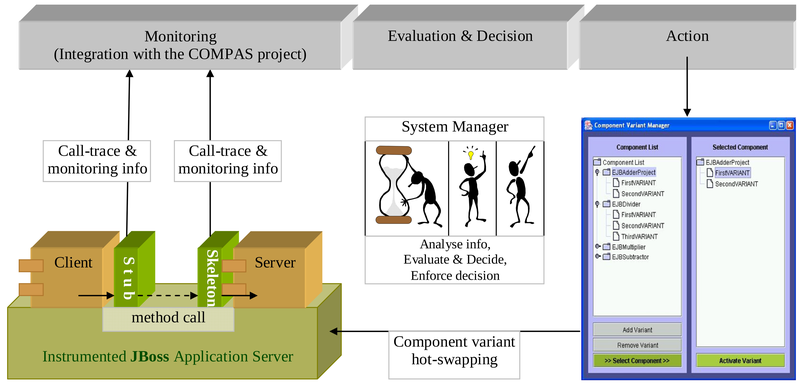
- Reusable framework:
  - Framework is implemented once and then used for any deployed application

Monitoring (&testing)
- Monitors application (response time, throughput)
- Monitors environment (CPU, memory, I/O usage)
- Determines problems:
  - Causes
  - Affected components

Evaluation
- Decides *which* component variants to replace and *when*:
  - Obtains information:
    - Monitoring information
    - Component descriptions (characteristics and history)
  - Uses decision policies (configured for each application)
- Updates descriptions and decision policies (Learn)

Action
- Activates component variants: redirect client requests to active variant
  - No state transfer (variants do not change per client request)
  - Consistent client references
- Maintains application integrity

Current Framework Implementation Status



Current Status:

- **Automated Monitoring tier**
- Role of Evaluation tier played by human System Manager
- **Automated Action tier**

Future work:

- Automate Evaluation tier (iteratively)

This work is funded by Enterprise Ireland Informatics Research Initiative 2002

Ada Diaconescu diacones@eeng.dcu.ie
Performance Engineering Laboratory (PEL)
Dublin City University www.eeng.dcu.ie/~diacones/

Leave comments, ideas, or questions here