Architecture-Aware Autonomic Adaptations within the Common Component Architecture

J. Ray, M. Parashar, H. Liu, B. Allen, S. Lefantzi
Emerging Next-Generation of Scientific Applications

Physical phenomena
- Large
- Multi-phased / multi-scale
- Dynamic
- Heterogeneous

Scientific applications
- Long running
- Assembled from independently developed computational units
- Geographically distributed

Environments
- Large
- Heterogeneous
- Dynamical
Challenges

• The emerging computing systems introduce a new set of challenges due to their scale and complexity.

• The emerging simulations and the phenomena they model are similarly large, complex, multi-phased/multi-scale, dynamic, and heterogeneous (in time, space, and state).
Solution: Autonomic Computing

- Computing systems that can manage themselves given high-level objectives from administrators.
  - Self-configuration
    - Autonomic systems will configure themselves automatically in accordance with high-level policies.
  - Self-optimization
    - Autonomic systems will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost.
  - Self-healing
    - Autonomic computing systems will detect, diagnose, and repair localized problems resulting from bugs or failures in software.
  - Self-protecting
    - Autonomic computing systems will automatically defend against malicious attacks or cascading failures.
The behaviors of individual managed elements are adaptable.

Interactions between elements are adaptable.

• Source: IBM
Overview of CCA

• Components
  – They are peers
  – They interact through provide/use ports

• A CCA framework (e.g., Ccaffeine)
  – It holds components and composes them into applications via connecting their ports
  – It supports SCMD and MCMD models
Extending Ccaffeine to Enable Autonomic Adaptations

• Component Manager
  – monitor and control the computational behaviors of individual components at runtime (e.g., dynamically selecting optimal algorithms, modifying internal states)
  – dynamically replace components
  – Dynamically coordinate with other component managers or composition managers to add / delete components
  – perform tasks assigned by composition managers

• Composition Manager
  – manage, adapt and optimize the overall execution of an application at runtime.

• Performance Toolkit (e.g., TAU)
  – Monitor resource utilization
Extending Ccaffeine to Enable Autonomic Adaptations

CCA framework + TAU
Node x

CCA framework + TAU
Node y

CCA framework + TAU
Node z

- user component
- Component manager
- Composition manager
Management Hierarchy

Composition Manager
- Decompose complex rules and inject them to component managers
- Evaluate and execute component/composition rules and decomposed rules

Component Manager
- Application level (across multiple components)

Components
Workflow

Component

Initialization

Export sensors and actuators, register events

Component Manager

Register sensors, actuators, and events

Load in component rules

Store decomposed sensors and actuators

Composition Manager

Load in and decompose composition rules

Computation

Query sensors

Forward sensor values

Rule evaluation

Rule evaluation

Rule evaluation

Conflict resolution and reconciliation

Actions to be invoked

Invoke actuators or replace components

Management
Self-optimizing

1. export actuator “algorithm”
2. register memory event
3. collect memory usage information
4. evaluate the rule
5. invoke algorithm with x

Component Manager

Algorithm x will be used from the next computation

Performance toolkit

IF memory < threshold
THEN algorithm x
Self-healing

1. register communication event
2. collect communication channel information
3. evaluate the rule
   IF communication_in_old FAILED THEN REPLACE old new
4. Replace the old component with the new component

The new component will be used from the next computation
Self-configuring

1. Triggered by self-optimizing and self-healing process

2. Delete sensors, actuators, rules related to the old component

3. Delete the connections and destroy the old component

5. Initialize the new component and get exposed sensors/actuators

6. Load in new rules

7. Connect the new component to interacting components

4. Load in the new component from component pool and instantiate it

Component Manager

New component

User component

Old component
Experimental Evaluations

Fig 1. The runtime overhead introduced in the minimal model

Fig 2. The percentage of overhead in the overall execution time
Fig 3. The average execution time of component rules and composition rules on parallel processors
Conclusion

• Challenges of the next generation of scientific applications
• Solution
• Extending Ccaffeine to Enable Autonomic Adaptations
• Self-management scenario
• Experimental evaluations