Overview of Adaptation

RAPIDware: Component-Based Design of Adaptive and Dependable Middleware

Project Investigators:
Philip McKinley, Kurt Stirewalt, Betty Cheng,
Laura Dillon, Sandeep Kulkarni

Software Engineering and Networking Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
(http://www.cse.msu.edu/RAPIDWARE)

Motivations

- Increasingly, distributed software must interact with both the physical environment and its execution environment
- Mobile computing software must adapt to dynamic conditions in several cross-cutting concerns:
  - Quality of Service
  - Platform characteristics
  - Security
  - Energy Consumption
  - Fault Tolerance
- How to support run-time adaptive behavior not envisioned during development?
Background

- Software Dynamic Adaptation
- Why?
  - Software runs in a changing environment
- What?
  - Software is considered to be dynamically adaptive if conditions in the executing environment cause new code to be introduced at runtime to achieve new behavior not previously possible with the original code [1].
- How?
  - Operating system level: 2K[3], K42[4]
  - Middleware level: dynamic TAO[5], Hadas[6]
  - Application level: Garlan[7]
  - Language support: AdaptiveJava[8], IguanaJ[9], SpartanJ[10]

Definitions

- Dynamic Adaptation:
  - To modify existing code at run time.
  - Remove or add (compose) code
  - Triggered by changing environmental conditions
    - Externally monitored
    - Internally monitored
- Application software: original system
- Associate with an application developer
- Adaptation code: new code to be added to application code
- Associate with an adaptation developer

Supporting Technologies

- AOP: Aspect-oriented programming
  - Separation of concerns
  - At compile or run-time, weave in cross-cutting aspects into core program
- Patterns: repeatable solutions for common problems (e.g., designs, architectures, requirements, models, specifications)
- Reflection: mechanism for program to observe its own state, identify problems and their locations
- Context-awareness: recognize different contexts to determine what policies and strategies should be applied
- Task-driven: system is aware of its intent
- Self-adaptation (transmutation): able to self-adjust capabilities in response to changing environment
Impact on Software Engineering [Garlan02]

- Manage Complexity through abstraction
  - Traditional SE: Use interfaces and specifications to describe functionality or property; hide details
  - Emerging: more subtle and flexible
    - Represent desired functionality, but tolerate "reasonable" deviations
    - Indicate resource requirements where necessary to setup user expectations

- Do not propagate errors to user
  - Traditional SE: Develop correct software
  - Emerging: Include monitoring and repair mechanisms to detect errors and repair when possible (at run time).

- Design systems with evolution in mind
  - Traditional SE: use configuration management to manage change for identified target execution environments
  - Emerging:
    - Support dynamic configuration
    - Support dynamic, automatic tuning for (unexpected) different environments
RAPIDware Project Goals...

- Develop adaptive middleware technologies that enable users to communicate seamlessly and securely across a diverse, dynamic, and evolving mobile computing and communication infrastructure.
- Target domain is collaborative computing
  - military command and control
  - crisis management systems
  - management of military/industrial installations
  - computer supported cooperative work
- The methods investigated can be applied to many other domains

Key Issues

- Separation of Concerns
  - Separating adaptive software functionality from the imperative code
- Adaptation Assurance
  - Guaranteeing that adaptations do not change the imperative behavior of the application in unexpected ways
- Migration Path for Development
  - Support adaptation through existing and new development paradigms.

Separation of Concerns

- Can adaptive behavior in “non-functional” concerns be separated from the (functional) application code?
- Doing so can facilitate the development, operation, and maintenance of adaptive systems
- How to implement “tradeoffs” among concerns
  - control changes in communication quality of service
  - manage energy consumption in battery-powered devices
  - provide fault tolerance according to specifications
  - actively monitor the system, execute security policies
Separating Adaptive Behavior

Application

Layer

Middleware

Layer

Network

Layer

Observers

Responders

Proxy node

(e.g., desktop)

Application

Host computer

(wired workstation)

Application

Host computer

(wireless laptop)

Application

Host computer

(wireless palmtop)

Assurance Issues

- How to support synchronization and atomicity during recomposition
- Use of "certified components"
- How to define and reach safe points
- System-wide integrity
  - Feature interaction
  - Global/system invariants

Migration Path

- Complexity of adaptive software is high
- Different types of developers:
  - Application developers
  - Adaptation developers
- Support existing programming models
- Transition to new programming techniques/languages/paradigms
Correctness Issues

- Functional and non-functional.
  - Functional: the adaptation preserve the correct functionality of the original program.
  - Non-functional: the new features do not interfere with each other.

- Invariants analysis.
- Feature interaction analysis.

Elements Supporting Adaptation

- Monitors (external and internal)
- Decision-making component
- Adaptation mechanisms:
  - Code insertion/replacement
  - Code removal
- Correctness components:
  - Adaptation safeness
  - Adaptation correctness