A1-Architecture-based self-adaptation

INF9360
Seminar on dependable and adaptive distributed systems
(Slides has some derived contents from various sources)
Papers to present

● An Architecture-Based Approach to Self-Adaptive Software

● Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure
An Architecture-Based Approach to Self-Adaptive Software

- Discusses the fundamental role of software architecture in self-adaptive systems
- Topics covered:
  - Self-adaptive software introduction
  - Degrees of self-adaptability
  - Dynamic Software architecture definition
  - Adaptation management
Sample scenario

**mission**: fleet of unmanned air vehicles to disable an enemy airfield

replan their mission, dividing into two groups: SAM-suppression and airfield-suppression

surface-to-air missile (SAM) launchers now guards the airfield
What happened

- Specialized algorithms for **detecting** and recognizing SAM launchers.

- Replanning by **Analyses** that include feedback from new situation.

- **New software** components are dynamically loaded and integrated without requiring restart, or any downtime.

- Taking place **autonomously**.
Self-adaptive software

- **Definition:**
  
  Self-adaptive software is a software that modifies its own behavior at run-time in response to changes in its operating environment.

- **Behavior:** anything the software is expected to do

- **Run-Time:** do not need to be shut down to make the change

- **Changes in operating environment:** anything observable by the software system, e.g. end-user input, external hardware devices and sensors, or program instrumentation
Issues for adaptation

■ What Conditions?
  ■ Performance boost, failure recovery, re-configuration

■ Open or Closed adaptation
  ■ Is new application behavior can be added at runtime?

■ Type of autonomy
  ■ Fully autonomic, Self-contained, Human-in-the-loop

■ Frequency
  ■ Opportunistic, continuous, or lazy, as needed

■ Cost Effectiveness
  ■ Benefits should outweigh the cost of adaptation

■ Information Type and Accuracy
Degrees of adaptability

Algorithm selection

- Online algorithms (deterministic, randomized, or probabilistic)
- Generic or parameterized algorithms
- Evolutionary programming (algorithm generation, genetic algorithms, AI-based learning)
- Conditional expressions
Conditional expressions

- Program evaluates an expression and alters its behavior based on the outcome
- E.g. *if/*switch statements
Online algorithms

- assume future events are uncertain
- leverage knowledge about the problem and the input domain to improve efficiency
- E.g. memory-cache paging algorithm
Generic algorithms

- Provide behaviors that are parameterized
- E.g. polymorphic type in OOPLs, working with instances of new classes (derived from known classes or implement known interfaces)
Selection of the effective algorithm among a fixed set of available algorithms based on environment properties.

E.g., Self optimizing compiler uses program-profiling data collected during program execution to select another optimization algorithms.
● Using properties of the operating environment and knowledge gained from previous execution to generate new algorithms
Adaptation methodology
Architectural model

- Enact changes and collect observations
- Evolution management
- Maintain consistency and system integrity
- Implementation
Architecture-driven development

- From Architectural model to Implementation
- Consistency between model and Implementation
- **System**: network of coarse-grained *components* bound together by *connectors*
- Connectors are transport and routing services for messages or objects
- Components do not know how their inputs and outputs are delivered or transmitted or even what their sources or destinations might be
- Connectors know exactly who is talking to whom and how
- Separating computation from communication
Dynamic software architecture

- In dynamic systems: modification in behaviour at run-time by making the following architectural changes:
  - Adding new components
  - Removing existing components
  - Replacing existing components
  - Changing the connectivity structure between components

- Two approaches to dynamism at the architectural level:
  - C2
  - Weaves
C2 and Weaves

- Both
  - distinguish between components and connectors
  - no restriction on their implementation language
  - communication between components by exchanging asynchronous messages (C2) or objects (Weaves)
  - all communication between components must be via connectors
System as a hierarchy of concurrent components bound together by connectors.

Component is aware of components “above” it and unaware of components residing at the same level or “beneath” it.

Communication between a component and those below it is handled implicitly using events.

Whenever a component changes its state, it broadcasts this to all components below it.
Weaves

- Dynamic, objectflow-centric architecture
- Components consume objects as inputs and produce objects as outputs
- Components do not know the semantics of the connectors that delivered its input objects or transmitted its output objects
Maintaining consistency
Maintaining consistency and integrity

- Preserving an accurate and consistent model of components and connectors
- Maintain a strict correspondence between the architectural model and the executing implementation
- Architecture Evolution Manager (AEM)
  - maintains the consistency between architectural model and implementation
  - prevents changes from violating architectural constraints
Enacting changes
Enacting changes

- Possible sources of architectural change
- Architecture editor
  - To construct architecture and describe modifications
  - With analysis tools such as design analyzer or domain-dependent analyzer
- Modification interpreter
  - Tool to interpret change scripts written in a change-description language to primitive actions supported by the AEM
Adaptation methodology
Collecting observations

- Varieties of observations: event generation, ...
- *Observers* for notification of exceptional events
- *Expectation agent*
  - detecting and noting single events is not enough
  - responds to the occurrence of event patterns
- New techniques for reducing the monitoring overhead
Evaluate and monitor observations

- Adaptive demands arise from inconsistencies or suboptimal behavior
- Evaluating and observing an application’s execution, including, performance monitoring, constraint verification, …
Plan changes

- The task of accepting the evaluations, defining an appropriate adaptation, and constructing a blueprint for executing that adaptation

- Two forms
  - Observation planning: determines which observations are necessary for deciding
  - Adaptation planning: determines exactly which adaptations to make and when
Deploy change descriptions

- Change descriptions
  - Included are any new required components or connectors and their affiliated annotations
  - Interact with the AEM to translates the change in descriptions into specific updates of implementation
Quiz and Summary

- What kind of dynamic architecture model will be most suitable for dynamic link library?
  
  **Answer:**

- Does evolution managers directly interfere in architecture editing?
  
  **Answer**

- Which programming model (In the spectrum) captures the result of previous execution?
  
  **Answer**
Summary

● Paper main goal
  ● Introducing an architecture-based approach to managing self-adaptive software

● To achieve this goal
  ● Describe dynamic software architecture
  ● Explain how architectural model eases software adaptation
Key points

- Software spectrum from self adaptation
- Presented dynamic software architectures: C2 and Weaves
- Making effort to connect software development process with adaptation process
Questions

- Methods for constraint matching?
  - Conditional operator
  - Black box testing: Based on I/O

- Methods for architecture enforcing?

- Difference in the architecture change requirements for the run-time changes and static changes?
Rainbow: Architecture-based Self-adaptation
By David Garlan et al, 2004
Motivating University Grade System

- Students using university web
  - University aims to provide timely and ubiquitous access
  - One student tries to hack in and change his grades

- Possible (escalating) responses:
  - Turn on auditing
  - Switch authentication scheme
  - Sandboxing
  - Move grades data
  - Close off connections
  - Partition network
  - Turn off services
Many Things Can Go Wrong

- Resource variability
- Changing environments
- Shifting user needs
- System faults

The system should dynamically adapt to these problems.
Traditional, Internal Mechanisms

Limitations
- Detection limited to localized view of system
- Outcome difficult to reason about
- Costly or infeasible to modify existing system
- Difficult to reuse logic for new system

Video Conferencing
- Data formatting in DB causes exception
- Network failure causes time-outs
- One application failure causes sig-HUP on the socket of another
- Garbage collection

- Exception handling
- Network time-outs
- Signal and interrupt
- Memory management
External Adaptation

- Global system perspective
- Important system-level behaviors and properties
- Explicit system integrity constraints
- Proven trade-off analysis techniques

Architectural model & Adaptation mechanism

University Grade Sys

Architecture-based self-adaptation
Desirable condition for Solution

- Ideally, we’d like a solution that
  - enables software engineers
  - to use architectural models
  - to adapt existing systems

- Key Challenge: One size does not fit all
- Solution should
  - apply to many architecture and implementation styles — general
  - facilitate adding self-adaptation capabilities — cost-effective
  - support run-time trade-off between multiple adaptation goals — composable

A family of systems with common element types (e.g., client-server, pipe-filter)

Effort to add self-adaptation is small, e.g., one person within days or weeks

Choice among competing goals based on stakeholder preference
Rainbow Approach

Architecture Layer

- Adaptation Engine
- Adaptation Executor
- Arch Evaluator
- Model Manager

Translation Infrastructure

- Gauges

Monitoring mechanisms

Effector mechanisms

System Layer

- Effectors
- System API
- Resource Discovery
- Probes

Running System
Our Rainbow Approach (2)

Architecture Layer

- Adaptation Engine
- Arch Evaluator
- Model Manager
- Gauges
- Translation Infrastructure

System Layer

- Effectors
- Resource Discovery
- Probes
- Running System
Architectural style

Characterizes a family of systems related by **shared structural** and **semantic properties**

- **Component and connector types** provide a vocabulary of element
  - e.g. **components**: Database, Client, Server, and Filter; **connectors**: SQL, HTTP, RPC,
- **Component Constraints** determine the permitted composition
  - e.g. constraints might prohibit cycles in a particular pipe-filter style
- **Properties** are attributes of the component and connector types.
  - e.g., load and service time properties of servers in a performance-specific client-server style
- **Analyses** can be performed on systems built in an appropriate architectural style.
  - e.g. **performance analysis** using queueing theory in a client-server system
Rainbow extension

- Rainbow extends notion of *architectural style* to support *runtime adaptation* by capturing the *system’s dynamic attributes*,

- In terms of the *primitive operations* that can be performed on the system to change it dynamically, and

- How the system can *combine* those operations to achieve desired effect.
Rainbow as a Tailorable Framework

- General framework with
  - *Reusable* infrastructure + *tailorable* mechanisms
  Specialized to targeted
  - system + adaptation goals

- Main components
  - Monitoring mechanisms
  - Model manager
  - Architectural evaluator
  - Adaptation engine
  - Effector mechanisms

What’s tailored
  - Properties
  - Vocabulary of model
  - Architectural constraints
  - Strategies & tactics
  - Operators
Demo: University Grade System

- Students using university web
  - University aims to provide timely and ubiquitous access
  - One student tries to hack in and change his grades

- Possible (escalating) responses:
  - Turn on auditing
  - Switch authentication scheme
  - Sandboxing
  - Move grades data
  - Close off connections
  - Partition network
  - Turn off services
Demo: University Grade System

- Composite system style:
  - Client-server + data repository

- Adaptation goals investigated:
  - Performance + security

counterIntrusion, counterDoS...

GradeServer.addService(), DbT.audit()

Db ⇔ MySQL DB, etc.

Architecture Layer

Adaptation Engine

Adaptation Executor

Arch Evaluator

Model Manager

Gauges

Translation Infrastructure

System Layer

System API

Resource Discovery

Probes

Effectors

Running System

Invariant intrusionProb ≤ threatThreshold

Firewall, Db, ServerT intrusion%, load

intrusion behavior patterns; specific IDS
Case Study 2: Video Conferencing

- System style: Service-coalition
- Adaptation goals investigated:
  - Performance + cost

- Case study showed:
  - > 90% of framework reuse
  - Need for principled coordination

Strategies:
- fixHHBandwidth, fixGatewayCost

Events:
- HandheldT.move()
- NetMeetingT.move()

GatewayT ↔ sysGateway, etc.

Invariant:
HH.availBandwidth > minHHBandwidth

Vict, NetMeetingT, HandheldT, GatewayT, ConnT
cost, load, bandwidth

Gateway/proxy
cost and load,
link bandwidth
Case Study 3: University Grade System

- Composite system style:
  - Client-server + data repository

- Adaptation goals investigated:
  - Performance + security

Strategy: counterIntrusion, counterDoS...

GradeServerT.addService(), DbT.audit()

DbT ⇔ MySQL DB, etc.

Invariant: intrusionProb <= threatThreshold

FirewallT, DbT, ServerT, SSHT

intrusion%, load

intrusion behavior patterns; specific IDS

Diagram:
- Architecture Layer
- Adaptation Engine
- Arch Evaluator
- Model Manager
- System API
- Resource Discovery
- Probes
- Effectors
- DbT
- GradeServerT
- Running System
- Translation Infrastructure
- System Layer
Preliminary Work Shows Promise

- Rainbow prototype
  - Integrated mechanisms and tested control cycle
  - Demonstrated usefulness for specific adaptation scenarios

- Two case studies
  - Three styles of system
    - Client-server, service-coalition, data repository
  - Three kinds of adaptation goals
    - Performance + security + cost

- Adaptation language under development
Conclusion

- Rainbow relies on existing capabilities in the managed system to allow system states to be extracted and changes to be effected.
- Using a software architecture model allows the adaptation engineer to abstract away unnecessary details of the managed system.
- The software architecture of a system is the structure of its **components**, their **interrelationships**, and **principles**
- Allows an engineer to
  - obtain a global system perspective,
  - explicitly capture system properties and constraints, and
  - leverage existing, proven architectural analysis techniques to determine problems and remedies
Some Research Problems

- Architectural “recovery” at run time.
- Efficient, scalable constraint evaluation
- Environment modeling and scoping
- Handling multiple models and dimensions of concern
- Reasoning about the correctness of a repair strategy
- Non-deterministic arrival of system observations
- Avoiding thrashing
- Adapting the adaptation strategies
Quiz

- What does Rainbow add in Architectural evaluation?
- Answer
- Can Adaptation executor directly take inputs from Probes? Why it can or it can’t?
Critics

- Architectural changes, can be highly effective, but **very hard to implement**, test, and manage
  - Specially in complex large-scale distributed systems.

- **Fallback**: Large-scale applications stability is major challenge. Specially under unanticipated conditions.
  - Dynamic change may lead to unanticipated conditions.
  - No recovery design

- **Evolution** is not clear.
Future direction

- Multi objective adaption
- Collaborative Adaptation cycle
- Dynamic update of strategies and strategy selection information
- Parallel adaptation
- Adaptation re-emption
- Real time adaption
The END