Model-based Self Adaptation
David Garlan and Bradley Schmerl
Carnegie Mellon University

Workshop on
Self-Healing Systems
(WOSS’02)

November 18-19, 2002
Charleston, SC
Self-Healing Systems?

Systems have built-in mechanisms for

- **Monitoring**: observing run time behavior
- **Resolution**: interpreting the meaning of run time observations
- **Detection**: recognizing when there is a need or opportunity for adaptation
- **Prescription**: identifying a strategy to effect an improvement
- **Repair**: mechanisms to change the system as it is running
Architecture-based Adaptation

Architectural models are used as basis for resolution, detection & prescription

n Form the basis for control
n Represent the system in terms of gross decomposition into components and connectors
n Allow repairs at the global system level for qualities such as reliability, performance, interoperability

These models are external to the system

n Separate from the system itself
n Require bridging mechanisms
n Provide clean separation of concerns
Architecture-Driven Adaptation

Architecture Layer

- Repair Handler
- Interpreter
- Translator
- Runtime Manager

Implementation Layer

- Arch. Evaluator
- Generic API
- Arch. Model

Monitoring Mechanisms

Executing System
Key Challenge: One size does not fit all

Self-healing must be tailorable to different

- Implementation styles
- Architecture styles
- Qualities of concern
- Repair expertise

Examples:

- A data flow system with end-to-end latency concerns (e.g., video teleconferencing), versus
- A real-time shared variable system with schedulability concerns (e.g., automotive systems), versus
- A blackboard-based planning system with reliability and resource consumption concerns (e.g., a NASA Mars Rover)
Our Approach: Parameterized Framework

- **Architecture Layer**
  - Repair Handler
  - Interpreter
  - Arch. Evaluator
  - Arch. Model
  - Generic API
  - Architectural Style
  - Architectural Constraints & Heuristics
  - Monitoring Mechanisms
  - Probes & Gauges

- **Implementation Layer**
  - Translator
  - Runtime Manager
  - Executing System
  - System-specific Effectors
  - Style-specific Operations
  - Style-specific Repair strategies

**Generic Framework**
Specific Adaptations

Current:
- Performance adaptation for distributed client-server systems
- Protocol monitoring

Under development:
- Service coalitions for pervasive computing environments
- Shared variable systems (Ford, NASA)

Currently not addressing:
- Probe technologies
- Implementation change mechanisms
- Constraint inference
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

Timing issues
  - Non-deterministic arrival of system observations
  - Change latencies

Avoiding thrashing
Adapting the adaptation strategies
Last Slide
Example

Req-queue: ftp.server.com

Client 1

Client 2

Client 3

Client 4

Client 5

Client 6

ServerGrp1

srv1.server.com

srv2.server.com

srv3.server.com

Spare Servers

Spare Servers

Spare Servers
Software Architecture

Graph of interacting components
- Components
- Connectors

Properties capture semantics
- E.g., performance properties, protocols

Tools to analyze
- Style conformance
- QoS conformance

Assumptions
- System can be monitored
- System can be modified at run time
Making Repairs

False! Find the right tactic

Client6.moveClient (ServerGrp2)

True?: avg_latency <= max_latency

Client6.avg_latency = 3.1

Move impl

4 second latency

– 12 –
Today

Users

Applications

OS/Network

Physical Devices

Tomorrow

Users

Tasks

Services

OS/Network

Physical Devices