Architectural Style Requirements for Self-Healing Systems

Marija Mikic-Rakic, Nikunj Mehta, Nenad Medvidovic

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781 USA

{marija,mehta,neno}@usc.edu

What Does Self-Healing Mean to Us?

- Self-healing systems
  - Ability to adapt (e.g., reconfigure) in response to the:
    - Changes within the system
    - Changes in the execution environment
    - System faults

Targeted Applications

- Highly distributed
- Highly mobile
- Decentralized
- Resource constrained

Problem

- No general understanding of what constitutes an effective self-healing style
- How to evaluate or compare different self-healing styles
- Assessing the suitability of an existing style for the self-healing domain

Approach

- Identifying architectural style requirements for self healing
  - Monitoring
  - Planning the changes
  - Deploying the change descriptions
  - Enacting the changes

Awareness

Autonomy

Robustness

Mobility

Adaptability

Dynamicity

Distributability

Traceability

MAP

Style characteristics:

- Structure
- Topology
- Behavior
- Interaction
- Data flow

Characteristics of the style / Requirement (activities)	Structure	Topology	Behavior	Interaction	Data flow
Adaptability (enacting changes)	Separable components ✓ Explicit connectors ✓ Explicit entry/exit portals ✓ Primitive ducts ✓	Portals attached to a single component or connector ✓ Constrained number of component portals - limited component dependencies ✓ Expandable (number of) connector portals ✓ Connectivity pattern exclusively portal[a]-duct-portal[b], where both portals cannot belong to components ✓ Limited component dependencies (visibility) ✓	Exposed via named services only ✓	Asynchronous coordination ✓ Implicit invocation ✓ Event-based interaction ✓ Adjustable connector delivery policies ✓	Discrete events ✓ Data streams
Dynamicity (enacting changes)	Separable components ✓ Explicit connectors ✓ Explicit entry/exit portals ✓ Primitive ducts ✓ Dynamism effectors ✓ Dynamic change analysis agents ✓	Components and connectors dynamically created ✓ Modifiable portal-to-portal bindings (i.e. ducts) ✓ Dynamically expandable (number of) connector portals ✓ Less constrained topological rules for attaching dynamism effectors to application architecture (to add the possibility of more direct control over the architecture) ✓ Dynamism effectors attached to analysis agents to ensure desired system properties during the change ✓	State preservation/restoration Component quiescence Dynamism change analysis ✓ Data queueing and buffering by connectors ✓ Dynamism effecting functionality ✓	Different interaction categories (e.g., allowed, deferred, disallowed) ✓ Delivery guarantees (at least once, exactly once) ✓ Synchronous, possibly real-time, meta-level dynamic change requests ✓	Dynamic change events ✓ System architectural models used to analyze the validity of proposed changes ✓ State transfer events
Awareness (monitoring)	Introspection portals Introspection interfaces ✓ Meta-level components ✓ Meta-level connectors Self-monitors System monitors ✓	Less constrained topological rules for attaching to introspection facilities, to enable a more direct control over the architecture ✓ Direct binding of monitors to components, connectors, portals, or ducts ✓ Less constrained topological rules for attaching to environment facilities	Self-monitoring and assessment functionality ✓ Execution trace capture ✓ Execution trace analysis Environment-monitoring and assessment functionality Environment event analysis	Real-time system monitoring data delivery (to monitor correctness) ✓ Asynchronous system monitoring data delivery (to monitor statistical performance) Ongoing or intermittent environment monitoring	System monitoring events ✓ Critical system monitoring event patterns Environment-level events Critical environment-level event patterns
Autonomy (planning, deploying, enacting changes)	Autonomous meta-level components ✓ Explicit, autonomous connectors ✓	Meta level components connected to dynamism effectors and change analysis components ✓	Adjustable planning policies	Synchronous architectural dynamism (to ensure that the change is atomic)	Dynamic change events ✓ System architectural models ✓ State transfer events
Robustness (planning, deploying, enacting changes)	Autonomous components ✓ Explicit entry/exit portals ✓ Primitive ducts ✓	Connectivity only via (known) portals and ducts ✓	Exception handling Data queueing/buffering ✓	Asynchronous coordination ✓ Implicit invocation ✓ Event-based interaction ✓	Discrete events ✓ Exception propagation
Distributability (general requirement)	Autonomous components ✓ Explicit and distributed connectors ✓ Explicit portals to remote environments ✓ Distribution channels (ducts) ✓ Distributed node registries	Distributed topology rules same as local topology rules ✓	Data caching by distributed connectors ✓ Connection setup and teardown Multi-tasking mechanisms such as threads ✓	Remote procedure calls (RPC) Discrete event-based interaction ✓ Continuous stream-based interaction Data marshalling and unmarshalling by distributed connectors ✓ Adjustable scheduling policies ✓ Delivery guarantees ✓	Byte streams Discrete events ✓ Quality of interaction parameters (e.g., real-time constraints, delivery guarantees security, synchronicity) ✓
Mobility (deploying, enacting changes)	Separable components ✓ Distributed connectors ✓ Explicit portals to remote environments ✓ Modifiable portal-to-duct bindings ✓ Mobility effectors ✓ Mobility effect analysis agents ✓	Modifiable portal-to-duct bindings ✓ Limited component dependencies (visibility) ✓ Dynamically expandable (number of) connector portals ✓ Less constrained topological rules for attaching mobility effectors to application architecture ✓ Mobility effectors attached to analysis agents ✓	State transfer Component quiescence Data queueing and buffering by connectors ✓ Mobility effecting functionality ✓	Different interaction categories during migration (e.g., allowed, deferred, disallowed) ✓ Delivery guarantees (at least once, exactly once) ✓ Data rerouting to new destinations	Meta-level mobility requests ✓ Data tuples System components ✓ System architectural models ✓

Problems Outside of Our Scope

- Programming language support for
  - Exception handling
- Artificial intelligence
- Adaptive components
- Algorithms for system adaptation