
Matthew J. Hawthorne and Dewayne E. Perry
Dept. of Electrical and Computer Engineering
The University of Texas at Austin

ACM SIGSOFT
Workshop on Self-Managed Systems (WOSS’04)
Oct. 31-Nov. 1, 2004
Requirements for Self-Adaptive Systems

- **Automatic reconfiguration**
  - Reflection (self-awareness)
  - Loosely coupled component architectures (e.g., component-connector, service provider architectures)

- **Learning and planning**
  - Correlation coefficient, Bayesian inference algorithms
  - Neural nets

- **Support for incomplete specifications**
  - Default / inherited implementations
  - *Intent*-based implementation taxonomies

- **Scalability**
  - Distributed architectures
  - Decentralized configuration
  - Distributed service discovery

- **Heterogeneity**
  - Platform-independent component frameworks
  - Adapters

- **Security**
  - Component-connector-monitor architectures
  - Opt-in component registration
General Approach

• Use constraint- and intent-based architectural prescriptions to bridge the gap between requirements and implementation architecture

• Model the problem domain as goals and constraints

• Model the application domain as a set of application activities and roles

• Model the solution domain using implementation object intents
Domain Models

• **Problem Domain** (System Requirements)
  - **Goals**: Functional requirements
  - **Constraints**: Non-functional prescriptions

• **Application Domain** (System Design)
  - **Activities**:
    • Model system functionality in application domain terms
    • Non-functional properties prescribed by constraints on activities
  - **Roles**:
    • Model role of implementation objects in a given application context

• **Solution Domain** (Implementation Architecture)
  - **Intents**:
    • Define implementation object function/purpose
    • Fulfill role(s) in an application
Intents

• Intuition: What if we could model system requirements and implementation objects so rigorously that the system would “know” how any object is intended to be used?

  – Awareness of object intent would be very useful for self-configuring systems
  
  – *Intent awareness is a necessary prerequisite* for architecture-level self-configuration
Requirements for Intent-based Architectures

• The *intent* framework must unambiguously model the function of implementation objects

• System requirements must be rigorously modeled in compatible terms

• The system (*or system architect*) must be able to map functional requirements to implementation object *intents* that can satisfy the requirements
  – Must also ensure conformance to non-functional system goals (*constraints*)

→ Any two objects with the same *intent* may be used interchangeably
Prescriptive Architecture Approach

• Prescribe system functionality in problem domain terms *(goals & constraints)*

• Model application *activities* to fulfill system goals, conform to constraints

• Define application *roles* for which objects are needed to implement activities

• Model the *functional intent* of all implementation objects & assign *intents* to application *roles* they can fill
Conceptual Model
Our Approach Addresses

- **Conformance to requirements specifications** during self-configuration
  - $\rightarrow$ *Architectural prescriptions* expressed in *problem domain terms*

- **Basic requirements** for self-evolving systems
  - $\rightarrow$ Useful *design features and techniques*

- **Architectural frameworks** for self-configuring systems

- **Self-configuration algorithms**

- **Architectural specification** using Architectural Prescription Languages (APLs)
Not Addressed (or Incompletely Addressed)

• **Component-level configuration**
  – Approach does not preclude component-level configuration
  – Not specifically addressed (focus is on *architecture-level* configuration)

• **Architectural constraints**
  – Addressed in general terms (as *non-functional system constraints*)
  – Semantics not specifically defined (*topic for further research*)
Discussion

• Architectural constraints and self-configuring systems
  – Architectural constraints are at odds with *architecture-level* self-configuration
  – Do we need to constrain the architecture *if the system is capable of reasoning about conformance* to system goals & constraints?
  – Is it possible that architectural constraints are only needed to overcome limitations inherent in current self-configuration approaches?
Discussion (continued)

• **Formal vs. ad-hoc intents**
  – How can a system implement self-configuring architecture without some notion of component intent (i.e., the functionality/purpose/role of available components)?
  *
  * **Intents formalize what is typically done in an ad-hoc manner**

• **General-purpose vs. application-specific intents**
  – Common general-purpose intent taxonomies would be nice to have for component interoperability; however:
  *
  * **Self-configuring systems can benefit from a more formal intent-based approach regardless of whether the intents are common/general-purpose or proprietary/application domain-specific in nature**

• **The essence of intent-based architectural configuration:**
  *
  * Rigorously model system requirements and implementation object functionality such that the system can select from the latter to fulfill the former
Main Contributions

• **Intent-based architectural configuration**
  – *Goal* and *constraint*-based *problem domain* model
  – *Intent*-based *solution domain* model

• **Distributed Configuration Routing (DCR)**
  – *Architectural framework* for self-configuring systems
    • Distributed network of loosely coupled *intent*-aware service provider components
  – *Self-organizing algorithm* for self-configuring systems
    • *State change-based intents* allow components to build candidate configurations without global system knowledge using only local information + information sent in the service request
    • *System can discover new unplanned architectural configurations* in response to unanticipated events, including *emergent system properties*

• **Propose Architectural Prescription Language (APL)**
  – Combine best features of ADLs and RDLs
  – Better support for *intent*-based architectures
Supplemental Material

• The following slides contain supplemental supporting material, including the remaining diagrams from the paper
Component-Connector-Monitor/Adapter Architecture
Self-Configuration Algorithm

CRR: Candidate Configuration Routes Returned to Origin

1. Origin transmits a CRR to DCR components requesting state change service from start state S to target state T.

2. R1 and R2 can provide state change to target state T (from F and E), so they transmit CRR config. paths with new target states F and E.

3. R4 in turn transmits its own CRR requesting state change service from internal states SR4 to TR4.

4. R12 transmits a CRR requesting state change from SR12 to TR12.

5. Configuration route discovery continues backwards from target state T toward start state S.

6. Final path components send configuration route candidates to the CRR origin.

7. The origin selects the best route and informs components.
Constraint-based Configuration Selection

Configuration Route Selection Using a Cost Metric

Note that configuration selection is based on tunable parameters, which may be expressed as a composite "cost", or "optimality" value.

Selected Routes: →
Unselected Routes: --→
Hierarchical View of Selected Architectural Configuration

Architectural Configuration Dependency View

Origin

- R8
- R6
- R7
- R4
- R2

  - R13
  - R12
  - R11

  - R15
  - R14

(*Selected configuration only)