Review — Dynamic Architectures

- Safety- and mission-critical software systems often cannot be shut down and restarted for upgrades
  - undesirable delays
  - increased cost
  - unacceptable risk
- Architecture is not an executable artifact; two possibilities
  - *simulate* system execution at the architectural level
  - reflect architecture modifications in *executing system*
- Support for architectural dynamism
  - constrained dynamism
  - unconstrained architectural dynamism
    - → the validity of the changes must be ensured at runtime

Review — Aspects of Dynamic Architectures

- Modeling dynamic architectures
- Specifying dynamic changes
- Governing change
- Runtime tool support
  - → ArchStudio

Why Test and Analyze Architectures?

- A high-level model of a software system
  - many details abstracted away
    - → simpler task
  - only properties of interest modeled
    - → results less noisy
- Produced early in the lifecycle
  - minimizes costs of discovered errors
    - → prevents unneeded downstream effort
  - serves as a system integration blueprint
  - serves as a system acceptance contract
    - → initial software artifact that addresses customer needs

Approaches to Architecture-Based A&T

- Analysis
  - structural
  - functional
  - extra-functional
- Testing
  - integration testing
  - acceptance testing

Structural Analysis

- Establishes adherence to a style or reference architecture (DSSA)
  - effective structural patterns
  - proper connectivity
  - necessary (but not sufficient) for concurrent and distributed properties
- Typically possible from the description of architectural configuration
- Style and reference rules are heuristics
  - desired system properties are probable rather than ensured
- More severe impact of failing to comply may be due to support tools

Functional Analysis

- Involves analysis of formal architectural models
  - interfaces
  - behaviors
  - extra-functional properties
- Possible at several levels
  - pairwise conformance of interacting components
  - analysis of subsystems of interacting components
  - analysis of system-level properties
  - analysis of architectures at multiple refinement levels

Pairwise Conformance

- Requires formal specification of
  - interfaces
    - → all ADLs
  - behaviors
    - → Rapide, Wright, Darwin, C2SADEL
  - extra-functional properties
    - → UniCon, MetaH
  - conformance rules
    - → *interface:* e.g., parameter contravariance, result covariance
    - → *behavior:* e.g., pre- and postcondition matching, CSP protocol conformance, event pattern satisfaction
    - → *properties*: e.g., performance, throughput, ... (in)equality
- Easiest to establish

Pairwise Conformance Examples

- Interface conformance

  prov getFirstElement(q : Queue) : Natural;
  req getFirstElement(q : FIFOQueue) : Integer;
- Behavior conformance
  - pre- and postconditions

    prov pre q.size >= 1;
    post ~q.size = q.size - 1;
    req pre q.size >= 0;
    post ~q.size = q.size;
  - CSP protocol conformance

    DataRead = get $\rightarrow$ DataRead $\sqcap$ $\surd$ $\rightarrow$ STOP

    User = set $\rightarrow$ User $\sqcap$ get $\rightarrow$ User $\sqcap$ $\surd$ $\rightarrow$ STOP
- Property conformance

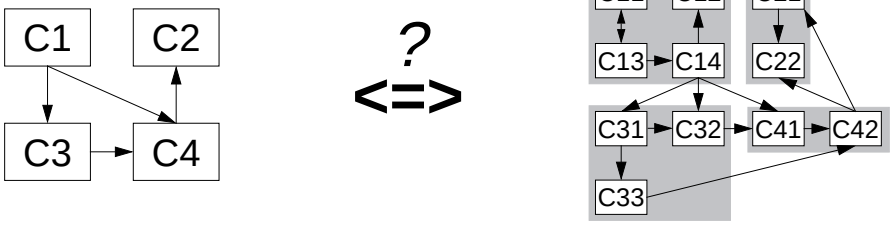
  Sender'Period := 1 sec;
  Receiver1'Period := 750 ms;
  Receiver2'Period := 1001 ms;

(sub)System Conformance

- Involves an entire system or a clearly identifiable part of the system
  - e.g., components clustered around a single connector
- Same requirements as pairwise conformance
  - modeling interface, behavior, and properties
- More meaningful results
  - e.g., interface mismatches, deadlocks, livelocks, starvation, violated properties
- More difficult to establish
  - non-monotonicity of results
  - “honey-baked ham” syndrome

Multiple Refinement Levels

- Level 1 architecture <=> Level 2 architecture



- Key questions
  - how much (if anything) can a lower-level architecture add?
    - how do we ensure the preservation of desired properties?
  - do we only require behavioral substitutability
  - do we need something stronger (e.g., conservative extension)?
  - how do we ensure refinement practicality/scalability?

Issues in Functional Analysis

- Critical vs. non-critical mismatches
  - architectures are not programs
  - some mismatches are “OK”
- Service provision
  - full provision is desired
    - anything any component needs is provided by some component in the architecture
  - partial provision may be more realistic
    - OTS reuse
- Service utilization
  - full utilization is desired
    - results in more efficient systems
  - partial utilization is more realistic
    - build components as general as possible

Extra-Functional Analysis

- Establishes satisfaction of system qualities
- Quantifiable vs. heuristic
  - quantifiable: throughput, performance, schedulability
  - heuristic: safety, modifiability, scalability
- Two approaches possible
  - analysis of individual architectures
  - comparison of architectures
  - → more challenging with heuristic qualities

Analyzing Individual Architectures

- Identify qualities of interest
  - “ilities”
- Identify application-independent conceptual features
  - dynamism
  - concurrency
  - types of connectors
  - layering
  - termination
  - preemption
  - control flow
- Relate features to qualities
- Identify the features applicable to the architecture

→ Establish architecture’s satisfaction of qualities

Comparison of Architectures

- Architecture can be broken down into
  - functionality
  - structure
  - allocation of functionality to structure
- Comparison requires
  - a well understood domain
    - common vocabulary
    - canonical structures
  - a classification of quality-related activities
    - inherently subjective
    - few qualities are well enough understood

  → the approach is analytical (and therefore subjective)

Example — SAAM

- Initial domain: UIMS
  - canonical vocabulary/structure — Seeheim
  - three systems with well-understood architectures
- Quality: modifiability
  - → extension of capabilities
  - deletion of capabilities
  - → adaptation to a new operating environment
  - restructuring
- Selection of benchmark tasks
  - demonstrate qualities of interest
  - do so for each relevant quality activity category

Architecture-Based Testing

- A variant of specification-based testing
  - black-box
- Requires a formal architectural model
  - simulations
  - “what if” scenarios
- Focus on architecture-level issues
  - reveal defects in dynamic interaction among components
- Testing up the abstraction hierarchy
  - unit/module
  - → integration
  - → system/acceptance

Possible Integration Coverage Criteria

- Each data element is exchanged
- All data dependencies are exercised
- Each component is executed
- Each connector is exercised
  - for each communication channel
- All possible (re)configurations are tested

Possible Acceptance Coverage Criteria

- Every functional requirement is exercised
- Every extra-functional requirement is satisfiable by the architecture