Using Statecharts in Software Architecture Analysis

Marcio Dias  Marlon Vieira  Debra Richardson

(mdias,mvieira,djr)@ics.uci.edu

Information and Computer Science, UC Irvine

Rosatea: Research Organization on Specification-based and Architecture-based Testing & Analysis

http://www.ics.uci.edu/~rosatea/
Why Software Architecture Analysis?

- Component based Software Development
- Software Quality Assurance:
  - quality of the components and of their configurations
  - analysis in the context of connections and interactions between components
- Architecture Level of Abstraction:
  - components, connectors and configurations are better understood and intellectually tractable
  - analysis may address both behavioral and structural qualities, such as functional correctness, timing, performance, style, portability, maintainability, etc
Why Specification-based Analysis for Software Architecture?

**Specification states what system should do**
- this information should be used to drive testing
- analysis is more likely to detect errors of omission
- Specifications enable formalized automation
- Specification-based analysis augments code-based testing

**Architecture-based Testing and Analysis**
- analyze architectural design against requirements
- test structure for conformance to architectural design
- test system and/or components against specified properties
- analyze components without knowing where they will be used
- test component-based system consisting of OTS components
- monitoring deployed software
Software Architecture Specification

- Architecture Description Languages (ADLs):
  - formal specification languages for describing architectures at a high level of abstraction
  - many ADLs fail to provide a mechanism for behavioral semantics

- Statecharts semantics augmenting ADLs:
  - specifying component and connector behavior
  - support to real time and event based systems
  - graphical notation with a precise formal semantics

- Experiment: Argus-I analysis capabilities by augmenting C2 ADL with statecharts
Argus-I

All-Seeing Architecture-based Analysis

- Environment for architecture specification, analysis and testing
- Specification-based architecture analysis toolkit
  - structural and behavioral analysis
  - component and architecture levels
  - combination of static and dynamic techniques
- Current version
  - architectures in the C2-style (structure specification)
  - component behavior specification described by statecharts
- Future versions
  - generalize to other architectural styles and ADLs
C2 Architecture Style

- Network of concurrent components connected by message routing devices
- Key elements: components and connectors
  - A configuration of components/connectors is an architecture
  - C2 components may have state and their own thread(s) of control
- Principles governing how the components and connectors may be legally composed
  - Limited visibility (substrate independence)
  - Components communicate by messages: notifications travel down an architecture and requests travel up
  - Components act on three situations only:
    - to react to a notification that it receives from the connector above it
    - to execute a request received from the connector below it
    - to maintain some constraint
Behavioral Extension to C2

- Lack of behavioral semantics in C2 limits both architecture and component analysis
  - C2SADL expresses component semantics in first-order logic using invariants and operation pre/post conditions

- Extend C2 with statechart semantics to describe component behavior
  - Map component conditions to statechart state definition of possible states/substates
  - Map component requests, notifications, and variables to entry/exit actions
  - Map component condition changes to statechart transition

- Related languages / approaches
  - Wright: CSP
  - Darwin: $\pi$-calculus or LTS (labeled transition system)
  - Rapide: posets (partially ordered sets of events)
Argus-I Process

- Iterative, evolvable analysis during architecture specification and implementation

- **Architectural Element (Component) Specification**
  - Create / Evolve
  - Reuse / Import
  - Analysis
    - Structure Analysis
    - Static Behavior Analysis
    - Dynamic Behavior Analysis

- **Component Implementation**
  - Develop / Deploy
  - Analysis
    - State-Based Testing (DAS-BOOT)

- **Architecture (Configuration) Specification**
  - Create / Evolve
  - Reuse / Import
  - Analysis
    - Dependency Analysis
    - Interface Consistency
    - Model Checking
    - Simulation

- **Architecture Implementation**
  - Compose / Integrate
  - Analysis
    - Debugging / Monitoring
    - Conformance Verification
Specification: Architecture (Configuration) and Component Structure

Elevator System simplest architecture: one elevator, one clock

Elevator component structural interface
Elevator States refined

Statecharts support concurrent substates, timing, and broadcast communication
Argus-I: Architecture Specification Analysis

**Structural Analysis**
- **Static**
  - interface consistency checking for connected components
  - dependence analysis between components
  - anomaly detection: components/interfaces defined but not used or used but not defined in architectural configuration

**Behavior Analysis**
- **Static**
  - model checking (SPIN & SMV) checks user-defined safety and liveness properties against super-statechart model
- **Dynamic**
  - simulation produces event trace generation/animation & and evaluation (error and warning notifications)
  - performance prediction/evaluation supported by statistics
Architecture Simulation

Simulation speed controls

Counts show number of events

Running: sim time = 5.9525

Speed: 10
Architecture Monitoring:
Event Trace (Generated by Simulation)

errors and warnings indicate unexpected events and inappropriate actions
C2 Dependence Analysis

- **Control dependence is not explicit, rather it is expressed by message passing**
  - discovered by what messages one component sends and what other components understand along with their location in the architecture

- **Data dependence implicit within messages**
  - determined by what parameters one component passes in messages to others along with where parameters are defined and used (in which components)

- **C2DG embodies all possible information flow among C2 architectural components (as an FSA abstraction)**
  - nodes represent the components in a specific architecture
  - edges labeled with the message represent the possible relationships between components
Architecture Dependence Analysis

Control is dependent on Elevator for AddCall, RemoveCall, CallAttended
Structural Analysis
- Static
  - type checking rules determine well-specified components

Behavior Analysis
- Static
  - syntactic and semantic checking of statecharts
    - identification of conflicting transitions (nondeterminism)
  - consistency checking between component interface and statechart operations
  - model checking (SPIN & SMV) checks user-defined properties against statechart model
- Dynamic
  - simulation of statechart given an input event sequence
## Component Simulation

### Simulation History

<table>
<thead>
<tr>
<th>States</th>
<th>Trigger</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>StoppedClosed, Idle, Ready</td>
<td>&lt;start&gt;</td>
<td></td>
</tr>
<tr>
<td>StoppedClosed, Idle, Ready</td>
<td>GetID</td>
<td>ElevID</td>
</tr>
<tr>
<td>StoppedClosed, Idle, Ready</td>
<td>GetStatus</td>
<td>Status</td>
</tr>
<tr>
<td>StoppedClosed, Idle, Ready</td>
<td>AddCall</td>
<td>CallAdded</td>
</tr>
<tr>
<td>Moving, Up, Ready</td>
<td>Step</td>
<td>ElevMoving ElevUp Status</td>
</tr>
<tr>
<td>Moving, Up, Ready</td>
<td>Step</td>
<td>ElevMoving ElevUp Status</td>
</tr>
<tr>
<td>Moving, Up, Ready</td>
<td>AddCall</td>
<td>CallAdded</td>
</tr>
<tr>
<td>StoppedClosed, Up, Ready</td>
<td>Step</td>
<td>ElevStopClose ElevUp Status</td>
</tr>
<tr>
<td>StoppedOpened, Up, Ready</td>
<td>Step</td>
<td>CallAttended ElevStopOpen ElevUp Status</td>
</tr>
</tbody>
</table>

### User-defined Event Sequence

- GetID
- GetStatus
- AddCall
- Step
- Step
- AddCall
- Step
- RemoveCall
- Step

### Statechart Debug: ElevatorADT2

- Event Sequence:
  - GetID
  - GetStatus
  - AddCall
  - Step
  - Step
  - AddCall
  - Step
  - RemoveCall
  - Step
Argus-I: Architecture Implementation Analysis

**Structural Analysis**
- Dynamic
  - conformance verification of component interfaces, parameters, events, ...

**Behavior Analysis**
- Dynamic
  - parallel execution of components’ statecharts
  - monitoring captures and visualizes event traces for architectural configuration and individual components
  - debugging provides architecture-level control of execution (breakpoints)
  - conformance verification between component execution and statecharts specification
Architecture Conformance Verification

- Elevator component
- IN/OUT events
- unspecified messages sent
- current possible transitions
- fired transition history
Argus-I: Component Implementation Analysis

- Structural Analysis
  - Dynamic
    - object inspection and modification of component state and messages
    - monitoring internal data structure
    - component / message inspection and modification

- Behavior Analysis
  - Dynamic: DAS-BOOT
    - testing of Java classes based upon and against statechart specifications
Arch/Component Debugging: Component & Message Inspection

Execution stopped at breakpoint set for Elevator component

User editing message before resuming execution
Conclusions

- Architecture- and Component-based software development relies on the quality of components and configuration.
- Analysis and testing guided by formal specifications at these levels is key for critical software systems.
- The behavioral semantics of Statecharts allows sophisticated architecture-based analysis and testing, as demonstrated by Argus-I.