A Systems Engineering Perspective of Aspect-oriented Software Architectural Analysis

Phillip Schmidt, Ph.D.
Phillip.P.Schmidt@aero.org
Agenda

- Our World
- Why a Systems Engineering Perspective?
- REACT
- Architectural Representation Challenges
- Evolving REACT
- Closing Remarks
Our World is Rocket Science!

Space system architectures growing increasingly complex
- Highly interdependent legacy subsystems
- Manual inspection of hardcopy designs ineffective in finding subtle design flaws
- Increasingly difficult to make technical tradeoff decisions based solely on qualitative judgments (e.g. within Integrated Product Teams)
- Architectural representation issues, object-oriented design technologies applied to legacy RT embedded systems not well understood

Space system architectures exhibit pressure to evolve
- Desire to improve performance, functionality, and program success
- New environments
- New services
- New contexts

Complexity and evolution raise risk

How do we manage architectural risk?
Challenges
(Early Discovery of Architectural Risks)

- Weak architectural tools
- Unconventional, inconsistent tool usage
- Actual
- Tools
- Intended
- Built-to
- Design Incompleteness
- Ascertaining derived architectural information
- Ambiguous Interpretation of design intent
- Incompatible design
- Design evolution errors
- Limited performance insight; flawed implementation
- Incomplete testing
- On-board failure
Why a System Engineering Perspective?

- Disconnect between **Vision and Reality**:
  - **Vision**: Architecture is central to supporting program evolution
  - **Reality**: Software architectural representations often incomplete and inconsistent

- A **systems engineering perspective** is needed to recognize and deal with the disconnect

- **Architecture is more than**
  - what **UML** is today
  - what **Aspect-oriented programming** is today (and will likely become)
  - questions about code

- **Architectural representation challenges await**
- **Aspect-oriented architectural analysis** is being used to tackle these challenges
Real-time Embedded Architecture-Centric Testbed (REACT)

- **Architecture-Centric**
  - Recognize importance of architectural representation
    - Many forms
    - Frequent access
  - Early discovery/feedback

- **Aspect-Oriented Architectural Assessment**
  - Architectural development exhibits concerns that cut across object decomposition boundaries
  - Support for automated management of concerns
Architecture-Centric

- Receive contractor-provided architecture artifacts
  - Unified Modeling Language (UML)
  - Other electronic representations
- Automatically extract architectural information
- Conduct architectural assessments
  - Prior to code development
  - Static Assessment
    - consistency/completeness
    - Compare “as-designed” to “as-built” representations
  - Dynamic Assessment
    - Focus on critical execution issues (synchronization, priority tasking, sizing)
    - Create simulations of well-formed models
    - Understand logical execution behavior of architecture
    - Refine/re-parameterize models
- Work closely with program office/contractor
- Work closely with UML vendors
## Aspect-Oriented Architectural Analysis

**Idea:** Apply aspects over UML architectural domain

<table>
<thead>
<tr>
<th>AOP</th>
<th>AOAA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leverage expression of cross-cutting concerns</td>
<td>Leverage expression of cross-cutting concerns</td>
</tr>
<tr>
<td>Programming language domain (e.g. Java)</td>
<td>Architectural domain (e.g. UML and other artifacts)</td>
</tr>
<tr>
<td>Solutions architecturally intrusive (completeness)</td>
<td>Architecturally non-intrusive; separable via simulation</td>
</tr>
<tr>
<td>Address dynamic, execution impacts</td>
<td>Address static or dynamic aspects</td>
</tr>
</tbody>
</table>
## Architectural Aspect Types

<table>
<thead>
<tr>
<th>Aspect Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Analysis Aspects</td>
<td>Perform integrity, consistency checks over UML space</td>
<td>Find all examples of destroy object usages</td>
</tr>
<tr>
<td>Derivation Aspects</td>
<td>Derive new or customized architectural information from UML space</td>
<td>Collect all event related information</td>
</tr>
<tr>
<td>Augmentation Aspects</td>
<td>Add new architectural informational detail</td>
<td>Supply model information based on ICDs, other analysis</td>
</tr>
<tr>
<td>Dynamic Assessment Aspects</td>
<td>Define cross-cutting concerns that need to be monitored</td>
<td>Log all raised exceptions; evaluate pre/post conditions</td>
</tr>
</tbody>
</table>
Real-time Embedded Architecture-Centric Testbed (REACT)
Aspect-Oriented Architectural Assessment

UML Model Evolution

UML Architectural Information

Model Extractor

Architectural Representation

Architectural Representation

Architectural Representation

Model Executor

Dynamic Assessment

Static Assessment Results

Aspect Translator

Model Generator

Model Configuration

Dynamic Assessment Results

Static Analysis Aspects

Derivation Aspects

Augmentation Aspects

Dynamic Analysis Aspects

Static Assessment Results

Static Assessment Results

Aspect Translator

Model Generator

Model Configuration

Dynamic Assessment Results

Static Assessment Results

Dynamic Assessment Results

Static Assessment Results
Aspects useful in exploring quality concerns

Message interaction sets too large for manual inspection
Dynamic Architectural Analysis

Model Configuration File

Simulation Model

Sample Model Outputs

Animated State Execution

UCI Research Forum
There still are problems...

<table>
<thead>
<tr>
<th>Problem Areas</th>
<th>Solution Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human factors, Architecture-Centric philosophy not always embraced</td>
<td>Improve trust, education, tools, methodologies, research</td>
</tr>
<tr>
<td>UML Usages</td>
<td>UML profile, improved architectural semantics</td>
</tr>
<tr>
<td><strong>Inconsistency</strong></td>
<td>Early discovery, Static analysis</td>
</tr>
<tr>
<td><strong>Behavioral incompleteness</strong></td>
<td>Augmentation, auto-generation, re-parameterization</td>
</tr>
<tr>
<td>Dynamic Assessment</td>
<td>Multi-level modeling techniques</td>
</tr>
<tr>
<td>Architectural Evolution, Cross-cutting concern analysis, etc</td>
<td>Better model representation/analysis techniques. Aspects</td>
</tr>
</tbody>
</table>

Ignoring these does not reduce architectural risk
## Different UML Usages

<table>
<thead>
<tr>
<th>Focus</th>
<th>UML Artifacts</th>
<th>PIM</th>
<th>PSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual system-level models (goals, objectives, system dependencies, constraints)</td>
<td>High-level sequence diagrams High-level state/activity diagrams Class/actor as subsystems Role relationships between components</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Requirements analysis and traceability (reqt ids, subsystem, build, test info)</td>
<td>Use case/functional requirement descriptions (nominal, alternative, exception, preconditions, postconditions, triggers)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Architectural/detailed design Level (active/passive objects interfaces, tasks, OS models, concurrency,)</td>
<td>Class diagrams as SW classes Detailed sequence diagrams (messages/methods, class participants) State behavior (class, method) Deployment info</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
REACT Example: 
Class Coverage in Sequence Diagrams

- 41% of classes are referenced in some sequence diagram
- 59% of classes are referenced in no sequence diagram

UCI Research Forum
Usage of Class Diagrams
REACT Early Discovery Example: Consistency and Completeness

- Consistency and Completeness
- Non-standard UML usage
- Inconsistent Classes, methods
- Traceability incomplete
- Less than 50% methods described

Graph showing various metrics such as Classes, Methods, Participants, Guards in Seq, etc., over a timeline from 1 to 10.
Dynamic Assessment

- **Goal:** Perform dynamic assessment when model behavioral information is missing

- **Approach:**
  - Multiple levels of modeling abstraction
  - Augmentation aspects
  - Monitoring aspects
Architectural Evolution

- Representations must support frequent change (mandatory/optional components)
- Not all features will be preplanned and separable
- Need to look backward, forward, and elsewhere! (e.g. old design decisions, new usage scenarios, other ICDs, changing requirements)
- Expand features to study concerns we don’t want! (e.g. design conflicts, deadlocks, unreachable states)
- Architectural complexities/dependencies will make feature interactions difficult to manage
- Separation/integration of multiple UML models
- Any given OO decomposition will eventually be reexamined
- There are cross cutting concerns that the programming domain alone cannot answer (e.g. version impacts, requirements evolution changes, workload)
Evolving REACT

- Improve Architectural Representation
- Improve Assessment Techniques
Expanding Architectural Representations

- Requirement Representation
- Environment Representation
- Workload Representation
- Aux
- Reports

- UML Model
- UML Profile
- Profile Interpreter
- Aspects
- Code Analyzer
- Reverse Engineer
- Code
- Artifacts
- ICDs
- Deployment Info
- Use cases

- Aspect Processor
- Model Generator
- Model Executor
- Model Configuration
- Dynamic Assessment
- Dynamic Assessment Results
- Static Assessment
- Prep Tools
- Extractor
- Dynamic Assessment Results
- Dynamic Assessment Results

UCI Research Forum
Expanding Assessment Techniques

- Develop tools/techniques to improve context and semantics
  - XML schemas represent/share architectural artifacts
  - Support augmentation from various sources
  - Support interpretation aspects (e.g. UML profiles of use)
- Augment representations with parameters derived from reverse-engineered code
  - Capture missing behaviors to improve evolution success
- Manage planned scenarios as analyzable use cases
- Manage planned features as aspects over entire representation space
  - Dependencies too difficult otherwise
- Move toward automating analysis and aspect-oriented impact analysis
- Develop architectural analysis techniques to discover design patterns and refactoring opportunities
The holy grail of architecture is not efficient software code generation but managing architectural risk during its evolution.

A systems engineering perspective supporting architectural assessments and impacts to change is desired.

Architecture is a core asset that goes beyond UML and AOP.

Architectural representation challenges remain.

UCI is a meeting the challenge!
Backup Charts
References

- **Scenario-based**

- **Feature-oriented**

- **Architecture-centric Design**
References (cont’d)

- **Aspect-oriented Programming**

- **Aspect-oriented Architectural Analysis**

- **Maintenance/Product Line Studies**
Definitions

- Architectural variability, refers to the ability to identify and flexibly reshape aspects of an architecture
  - Aspects identify points of variation

- Program evolution refers to the ability of an architecture, over its lifecycle, to undergo change
Augmentation Aspects

- Example: Initially model missing information as a “black box”
  - An aspect identifies
    - Area/context of interest (e.g. methods with no state behavior)
    - Some action to be taken (associate some default black box action state with that method)
  - Later another aspect could replace/revise the black box behavior
- Example: Identify all COTS tool interfaces
Monitoring Aspects

- Monitor defines an action to take and the condition under which to enable it.
- Currently monitoring is independent of system under study. E.g. monitoring does not force adaptive behavior.
- Augmentation aspects can tag areas and enable monitoring. E.g. All interrupt handler methods.
- Monitoring can provide directives to the simulator (e.g. report Task msg queue size)
Multi-Level Modeling Types

- Method-level Modeling
- Participant-level Modeling
- Use-case level Modeling