Acme: a Language for Architecture Exchange and Analysis

Dave Wile
USC/ISI/CSE
wile@isi.edu
http://www.isi.edu/software-sciences/wile/home-page.html

Talk Outline

- What is architecture?
- The uses for formal ADLs
- Acme fundamentals
- Acme support tools
- Current work
  - Architecture dynamism
  - Architecture-based, domain-specific analysis and generation
What is Architecture?

“Boxes and Arrows”

- Convey structural information
  - Gross organization
    - Interacting, interconnected components
- Often require multiple views or perspectives
- Identify conceptual or actual artifacts
- Identify information or control flows
- Concern events relating the artifacts
- Foundation for constraints on behaviors and relatedness of entities and events
- Do not convey functionality
Architecture Design

- Current practice:
  - ad hoc
  - informal
  - picture-based
- Therefore,
  - Poorly understood by developers
  - Designs cannot be analyzed for consistency or completeness
  - Constraints are not enforced during system evolution
  - No tools to help designers with their tasks
- Hence, we need formal architecture description languages

The Uses for Formal ADLs
Uses for ADL Specifications

- Confluent terminology
- Structural specification for readers
- Application-independent analyses
  - connectors are connected:
    » all top level inputs are inputs to some subcomponent
    » all top level outputs are outputs from some subcomponent
  - contexts imposed on the same name are consistent
  - instantiations have the same number of input and output arguments as the generic
  - parts referenced by instances must be defined by generics

Uses for ADL Specifications

- Application-dependent analyses:
  - Interaction protocols
  - Bandwidths and latencies
  - Locations of resources
  - Anticipated dimensions of evolution
- Simulation
- Animation
- Instantiation to produce application code
- Traversal mechanisms for system programmers
  - apply to each
  - filter
  - choose subarchitecture
Impact
No duplication of architecture-related tools among EDCS Community
Shared Architecture and Generation Cluster community vision
Low buy-in, architecture-based analysis and prototyping support for specific problem domains

Schedule
ACME Language Specification
Web-based analyzers and tool activation
ACME-based performance analysis
Dynamic Architectures
Customization Packages
Constraints support

New Ideas
Community Consensus Architecture Interchange Language
Integration of independently-developed, architecture analysis tool sets.
Idiosyncrasies conveyed via annotated properties
Architecture Infrastructure Shared via the Web

Acme Fundamentals
Goals for ACME

• Interchange format for architectural development tools and environments
  – n * m problem -> m + n
  – tools
    » graphical interface tools
    » animation
    » analysis for deadlock, well-formedness
    » architecture style-specific tools

• Underlying representation for developing new tools for analyzing and visualizing architectures

Goals for ACME

• Vehicle for creating conventions: consensus building
  – Semantic foundations
    » refinement
    » event-based
    » temporal logic
  – Architecture families
    » Architecture evolution
    » Dynamic architectures

• Expressive descriptions that are easy for humans to read and write
ACME Kernel

- Components, with ports
- Connectors, with roles
- Attachments of particular ports to particular roles
- Aggregates: collections of components, connectors and attachments
- Properties of any of above

Acme Support Tools
Project Directions

- Deployment of Research Components
  - Development: AcmeLib, Webifier, GUI Tools, Layout Tools, Acme Language with (Types and Families)
  - Research: Families (styles), Dynamism, Semantics, Constraints (Armani), Domain-specific architecture analyzers and generators

- Deployment Platforms
  - Development: Java, C++
  - Research: Common Lisp (Popart/AP5/Flea), Haskell, Java++

- Community-wide Architecture ToolKit (ATK)

Automatic Tools

- Available:
  - Layout
  - Connectivity Analysis
  - Acme => Rapide, Wright, and UniCon's tool availability
  - Performance Analysis

- Under Development:
  - PC-based Acme graphical editor
  - Powerpoint-based Acme graphical editor
  - Static constraint checker and visualizer
  - Event sublanguage
Tools for Generation

- Available:
  - GUI Acme interface, generating Acme from Graphics and vise versa
  - Interchange via Acme generation among: Rapide, Wright, UniCon, Aesop

- Under development:
  - Acme (Armani) constraint enforcer
  - Acme semantics generator
  - MetaH in Acme
  - Constraint "residue" generation for constraint checks against dynamically maintained model of architecture

Incremental Value-Added

- Adapt and build on what you have
  - No implementation assumptions on what architecture is describing
  - Versions available on multiple platforms

- Reuse of effort
  - Customize architecture families for multiple product reuse

- Increased return on assets
  - Introduced automation where none existed
Certainty in Design and Development

- Ensure against mistakes
  - Architecture analysis up-front, before detailed design decisions, e.g. precise module decomposition
  - Dynamic verification of architectural constraints with potential for violation prevention
- Change with high confidence and predictability
  - Reanalysis is cheap
  - Interface assumptions constantly rechecked if using architectural composition mechanism
  - Design evolution constraints enforced
  - Dynamic constraint maintenance

Easily Assimilated

- Adoption barriers
  - Shallow learning curve for Acme specification language
  - Shallow learning curve on available tools (high leverage)
  - Multiple platform availability enables adoption in embedded systems
- Costs
  - Low buy-in to obtain tool use
  - Multiple platform availability avoids interoperation costs
Web-based ACME Tools

- www.cs.cmu.edu/~acme
- Webifier: produces web-viewable description of ACME
- Layout tool that packages graph layout algorithms that you can experiment with (used by webifier)

Research Directions

- PowerPoint Design Editor
- Architecture Dynamism
- Quality of Service Specifications
ACME is Stable

- I am about to describe our research at ISI
- Do not mistake the speculative nature of the research as instability in the language.
- Existing syntax and tools will be supported as long as there are users.
- We are extending Acme to include dynamism.
- Moreover, we are specifying Acme's semantics precisely to understand the dynamic effects and design problem-specific tool support

Research Directions

- ==> PowerPoint Design Editor
  - Architecture-based, domain-specific analysis and generation
- Architecture Dynamism
- Quality of Service Specifications
Architecture for Analysis

- Powerpoint interface to Acme
- All components, ports, connectors, and roles created using normal Powerpoint
- Mediating Connector technology used to keep a shadow AcmeLib-like model of activity
- Designer customizes the icons, colors, fonts, etc., used to create the architecture
- Designer decides what attributes can be attached to the various parts
- Designer writes problem-domain specific analysis packages
Designer’s Environment

- Component and Connector classes and abstract classes built up graphically
- Type structure conveyed by arrows
- Enumerated datatypes in addition to integers, float, boolean and string specified
- Legal attributes described on a per type basis
- Attributes inherited from abstract classes
- Analysis menus and error notification information described
Research Directions

- PowerPoint Design Editor
- Architecture Dynamism
- Quality of Service Specifications

Architecture Dynamism

- Variety of ways architecture can change:
  - Instantiation of nearly complete template
  - Logical refinement of components
  - Additions to skeleton or reference architecture
  - Implementation details
  - Self-modifying architecture
- The last is what is usually meant by "dynamism" in architecture, but the concerns are the same in all
Dynamic ACME

- Possible uses:
  - Design rules a la Armani
  - Law-governed systems a la Naftaly Minsky
  - Flea-enforced multiple stage dynamic constraints
  - Single state constraint checking a la Common Lisp ACME Analyzer
  - Adapt Wright dynamism (~ constraints on script)

Shared Concerns

- Stem from analyzability
  - Want to guarantee that certain properties hold or do not hold
  - These depend on the architecture, the topological structure

- Examples:
  - End-to-end throughput depends on interfering traffic
  - Security depends on knowing all connections through which secure information could leak
  - Accountability depends on intercepting all sources of information needing accounting
Closed World Assumptions

- Allow finer reasoning
  - structural closure is ideal
  - closure of means for changing/constructing
    may be just as good (induction needed to prove
    properties maintained)

- Alternatives:
  - model the dynamic architecture and check
    constraints on behavior dynamically
  - must detect before a system becomes unsafe

Declarative Dynamic ACME

- Semantic implications:
  - open elements, elements whose parts are not
    entirely specified
  - optional and multiple elements, whose presence
    is not necessarily guaranteed

- Constraints represented by specifications
  are conditionally existentially quantified

- Operational ACME mirrors in transactions
Acme Semantics

- Properties and predicates are either:
  - Assumed to hold -- the axioms
  - Derivative from those assumed to hold -- the theorems
- Implicitly, an Acme specification requires that topological assumptions be verified to hold in the artifact the specification purports to describe
- Assumed predicates must also be verified there
- Derived predicates are logical consequences

Example Open Component

- open component A-D
  - \{ port in = \{ assume property I = r \} \};
- Component A-D and port A-D.in can be identified in the artifact:
  - identifiable(A-D) and identifiable(A-D.in)
- They are instances of types corresponding to component and port in the implementation:
  - component(A-D) and port(A-D.in)
- Port A-D.in is a port of A-D:
  - port-of(A-D.in, A-D)
- A-D.in's properties are satisfied in the artifact:
  - I(A-D.in) = r
Same Component Closed

- open component A-D
  = \{ \text{port in} = \{ \text{property} \text{ I} = r \} \};

- semantics:
  identifiable(A-D) and identifiable(A-D.in)
  and component(A-D) and port(A-D.in)
  and port-of(A-D.in, A-D)
  and for all p .
    port-of(p, A-D) => p = A-D.in
  and I(A-D.in) = r

More Constrained Openness

- Allow optional parts in specification (ports, components, connectors, roles)
- Semantics is a predicate conditioned on identifying the optional parts
- Some closure statements are still possible
- Done with "multiplicity" notation of mathematics adopted by UML
Example Optional Port

- component A-D
  = {  
    port in;
    0..1 port scale=property P=q
  }

- semantics:
  identifiable(A-D) and identifiable(A-D.in)
  and component(A-D) and port(A-D.in)
  and port-of(A-D.in, A-D)
  and (identifiable(A-D.scale)
      => P(A-D.scale)=q)
  and for all p · (port-of(p,A-D) =>
      p=A-D.in or p=A-D.scale)

Optional and Multiple Parts

- open component A-D
  = { 
    port in;
    open 0..1 port scale;
    0.. port out
  }

- Informally:
  • A-D may have more ports
  • If port scale is present, it may be defined further
  • Port out is multiple and may be referred to as out[j],
    but it need not appear
  • Ports in and out are "closed"
  • Essentially, these provide idioms for refined openness
Real Use for Semantics

- Specify styles using dynamism constructs
- Instantiate style types in a specification
- Derive semantics automatically: assumptions and proof obligations
- Pass off semantics to a theorem prover
  - Theorem prover may prove some things inconsistent; must rewrite specification
  - Theorem prover may be able to derive some derivatives
  - Take residue and check dynamically in a simulation keeping track of the abstract architectural structure (using a Base Acme facility)
Research Directions

- PowerPoint Design Editor
- Architecture Dynamism
- => Quality of Service Specifications

QoS Problem

- Construct and run
  - many complex applications
  - whose computing resource needs vary over time
  - sometimes conflicting with one another
  - on a single computing platform
- Guaranteeing QoS attributes to the applications
  - resource utilization
  - robustness
Allow Applications to:

- Place varying demands on system resources over time.
- Communicate changes in requirements to a QoS infrastructure.
- Respond to changes in resource availability.
- Negotiate with providers for the appropriate level of resource support.

![Diagram](image)

Figure 1: Image Processing Architecture
Figure 2a: Station 1 Architecture
Figure 2b: Station 2 Architecture

Figure 3: Implementation Architecture
Technology: each application

- The logical topology (architecture) of the application;
- A state-space characterization of its dynamic execution;
- The topological elements' dependencies on these states and resource requests associated with them;
- Transitions between states in this space;
- Models governing the timing and extent of resource consumption (to be transmitted to the QoS manager);
- Policies reacting to choices that could be presented by different platform models;
- Platform and state-based implementation choices for topological elements;
- Platform and state-based descriptions for QoS model monitoring.

Each Implementation

- Its topology (architecture), normally a modification of an already-existing design;
- QoS performance and cost models for the resources it controls.
Generate Automatically

- An analysis of the well-formedness, consistency, completeness, and conformance between the various specifications;
- An implementation harness for communicating QoS information between nodes in the platform topology;
- A distributed QoS manager that runs on the platform;
- A set of surrogate model managers for the applications, collecting QoS-relevant information and transmitting it when it differs from the distributed manager's model;
- A QoS model manager, reporting discrepancies in QoS resource availabilities or performance costs from those previously known by the application.

Station 1 Specification

<table>
<thead>
<tr>
<th>States</th>
<th>Monitoring, Seeking, Coping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components:</td>
<td></td>
</tr>
<tr>
<td>IR:</td>
<td></td>
</tr>
<tr>
<td>When Seeking or Coping: present = true</td>
<td></td>
</tr>
<tr>
<td>Recognize:</td>
<td></td>
</tr>
<tr>
<td>When Monitoring: Execution module is &quot;c:/egg/scan.exe&quot;</td>
<td></td>
</tr>
<tr>
<td>When Seeking: Execution module is &quot;c:/egg/hot-spot.exe&quot;</td>
<td></td>
</tr>
<tr>
<td>When Coping: Execution module is &quot;c:/egg/snap.exe&quot;</td>
<td></td>
</tr>
<tr>
<td>Requests:</td>
<td></td>
</tr>
<tr>
<td>When Monitoring &amp; Result.movement: allocate IR</td>
<td></td>
</tr>
<tr>
<td>When Seeking &amp; Operator.cancel: release IR</td>
<td></td>
</tr>
<tr>
<td>Transitions:</td>
<td></td>
</tr>
<tr>
<td>When Monitoring: Result.movement → Seeking</td>
<td></td>
</tr>
<tr>
<td>Main_Camera.failure → Coping</td>
<td></td>
</tr>
<tr>
<td>When Seeking: Result.quiescent → Monitoring</td>
<td></td>
</tr>
<tr>
<td>When Coping: Main_Camera.on_line → Monitoring</td>
<td></td>
</tr>
<tr>
<td>Result.movement → Seeking</td>
<td></td>
</tr>
</tbody>
</table>