ACME: an Interchange Language for Software Architecture Descriptions

David S. Wile
University of Southern California
Information Sciences Institute
wile@isi.edu

Goals of Presentation

- Convey technical understanding of ACME, a beginners architecture description language (ADL)
- Convince you that:
  - Adopting ACME is a low buy-in first step in architecture technology
  - ACME can be evolved to support specialized needs of problem areas
Talk Outline

- What is architecture?
- Why formalize it?
- ACME technical details
- When to expect what

What is Architecture?
Information Present

- Structural decomposition of SPIN meeting
- Connectors of various types:
  - data flow: slides to and from Jennifer
  - data flow: introductions
  - notice ternary connector
  - communication: presents
- Containment: SPIN members include Barry
Missing Information

- control flow: e.g., order of presentation
- additional and/or refined structure
  - some events: e.g., eating and intermission
  - structure: e.g., talk outlines
- constraints: e.g., not presenting while introducing

Architecture Descriptions

- 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
- Constrain 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

Need 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
Formal Architecture Description
Languages

- Aesop (Garlan at CMU): styles
- Adage (Coglionese at FSD): avionics navigation
- Meta-H (Vestal at Honeywell): real-time control
- C2 (Taylor at UCI): user interfaces
- Rapide (Luckham at Stanford): simulation and analysis
- SADL (Moriconi at SRI): refinement
- UniCon (Shaw at CMU): heterogenous styles
- Wright (Garlan at CMU): analysis of interactions between components
- Darwin (Kramer at Imperial): dynamic architectures

ADL Proliferation

- Plus side:
  - Exploring different facets of the overall problem
  - Tools developed for such exploration
- Minus side:
  - Stand-alone stovepipe systems
  - Cannot combine with others
  - Must reimplement:
    - graphical tools
    - persistent stores for designs
    - domain-independent forms of analysis
  - Investment to come on-board with a new application is heavy
ACME Technical Details
ACME: an Architecture Exchange Language

David Garlan (CMU)
Robert Monroe (CMU)
David Wile (ISI)

Goals for ACME

- Interchange format for architectural development tools and environments
  - $n \times m$ problem $\rightarrow m \times n$
  - 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
- Foundation for developing new, domain-specific ADLS
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

Your
Component
Name Here

Your
Connector
Name Here
Syntax Conventions

- Every major element is identified with a keyword before the declaration:
  - `component NoiseGenerator = {port out};`
- Groups of elements identified by plural keyword:
  - `component RadarTracking = {ports {signal-in, track-out}};`
- Properties allowed in all declarations:
  - `connector ErrorReport = {role report = {property datatye=Boolean}; role source}`
ACME Controller Specification

system controller = {
  component Idle = { ports { to-left, to-right, input}};
  component AwaitR = { ports { right-closed, right-opened,  
                              left-opened, timed-out, input}};
  component AwaitL = { ports { left-closed, right-opened,  
                              left-opened, timed-out, input}};
  component Activate = { ports { pressed-enough, input }};
}

ACME Controller Specification

connector I-CR--AL = { roles {in-CR, out--AL}  
                        properties {in: "CR"}};
connector I-CL--AR = { roles {in-CL, out--AR}  
                        properties {in: "CL"}};
connector AL-CL-CP-A = { roles {in-CL, out-CP-A }  
                        properties {in: "CL"; out: "CP"}};
connector AR-CR-CP-A = { roles {in-CR, out-CP-A }  
                        properties {in: "CR"; out: "CP"}};
connector A-T-OP-I = { roles {in-T, out-OP-I}  
                        properties {in: "T"; out: "OP"}}

***
ACME Controller Specification

**attachments**

\{ Idle.to-left to I-CR--AL.in-CR;
AwaitL.input to I-CR--AL.out--AL;
Idle.to-right to I-CL--AR.in-CL;
AwaitR.input to I-CL--AR.out--AR;

\*
\*
\*
\}

Translation between ADLs

![Diagram showing translation between ADLs]

**ACME Success Criterion:**
Narrow arrow between property languages.
Additional Kernel Concepts

- Need to extend Kernel to as large a language as is acceptable by the community
- Types
  - predicates, available for components, ports, connectors and roles
  - extendible
- Refinement
  - substructure specification
  - bindings of external interfaces to internal

Types

- Can declare types for all major categories: components, connectors, roles, ports, and properties
  - connector type UnixPipe = {roles (source, sink);
    property protocol: Wright };
  - property type Viz = Record [x,y:Integer; color: Color]
- Can then instantiate:
  - connector dump-excess = new UnixPipe;
  - property loc.Viz=[3,5,blue]
- Types can be extended:
  - connector type ErrorPipe
    = extend UnixPipe with (port error);
- May allow parameterized types.
Types Example

system Plant = {

component type ND-state
= { ports [in-state, next-state]};

connector type transition (in-state, next-state: component,
input, output: name)
= { roles [in-state, next-state];
properties [in: input; out: output]};

component Closed-left-button = new ND-state;
component Open-left-button = new ND-state;
component Closed-right-button = new ND-state;

connector OL = transition( Closed-left-button, Open-left-button, R, OL);
connector OR = transition( Closed-right-button, Open-right-button, R, OR);
connector CL = transition( Open-right-button, Closed-right-button, R, CL);

attachments • • •
}

Refrainements

* Want to express the internal structure of a
  component, connector, port, or role
  - must relate the outer structure to the inner structure
  - via "bindings"
Refinements

\[
\text{system HydraulicControl=}\{
\begin{align*}
\text{component } & \text{ctl} = \{ \text{ports } \{ \text{input, output} \} \} \\
& \text{representation } \{ \text{Controller} \\
& \hspace{1cm} \text{bindings } = \{ \ast \ast \} \}; \\
\text{component } & \text{hdw} = \{ \text{ports } \{ \text{input, output} \} \} \\
& \text{representation } \{ \text{Plant} \\
& \hspace{1cm} \text{bindings } = \{ \ast \ast \} \}; \\
\text{connector } & \text{sensors } = \{ \text{roles } \{ \text{send, receive} \} \} \\
\text{connector } & \text{actuators } = \{ \text{roles } \{ \text{send, receive} \} \}; \\
\text{attachments } & \{ \text{ctl.input } = \text{sensors.receive}; \} \\
& \hspace{1cm} \text{ctl.output } = \text{actuators.send}; \\
& \hspace{1cm} \text{hdw.input } = \text{actuators.receive}; \\
& \hspace{1cm} \text{hdw.output } = \text{sensors.send}; \}
\end{align*}
\]

ACME Extensions to Kernel

- **Templates**
  - typed macros
  - with typed arguments
- **Families: styles and other constrained aggregates**
  - specification as a set of templates and types
  - declaration of restriction to family enforces template usage
Introduce Community Consensus Styles

Narrows the arrow between property languages.

Templates and Families

- **Templates are like macros:**
  - `template connection (a,b:component) defining (connector c)`
    = (component a = (port out);
           component b = (port in);
           connector c = (roles into, outof));
    attachments (a.out to c.into;
                 c.outof to b.in ));
  - `connector rpc = connection (lisp1, unix3);`

- **Can type fully:**
  - `connector template connect (a,b: component)`
    = (roles into, outof); *** )

- **Can define multiple elements:**
  - `template BigCon defining (connector c; components a,b) = ***`
  - multiple (connector cinst; component ainst; component binst)
    = BigCon
Families

family StateCharts = {
    component type ND-state () = {ports(in, out)};
    connector template transition (s1, s2; ND-state, stimulus, response; Name) =
        (roles {input, output};
         attachments {s1.out to transition.input;
            property transits-on=stimulus;
            s2.in to transition.output;
            property causes-response = response});

    system Plant : StateCharts =
        {components Closed-left-button, Open-left-button, Closed-right-button,
        Open-right-button, Closed-press, Open-press = new ND-state;
        connector OL = transition(Closed-left-button, Open-left-button, rand, OL);
        connector OR = transition(Closed-right-button, Open-right-button, rand, OR);
        connector CL = transition(Open-right-button, Closed-right-button, rand, CL);
        connector CR = transition(Open-right-button, Closed-right-button, rand, CR);
        connector CP = transition(Open-press, Closed-press, CP, none);
        connector OP = transition(Closed-press, Open-press, OP, none)};

Types and Templates

• Types and Templates are complementary
  - Types define sets of values
  - Templates are a convenient macro facility
  - Templates may be constructors for types.
  - Types make a strong claim about structure.
  - Templates make no such claim
ACME Semantics

- Problem: Not all box and arrow diagrams model what we would consider to be "architecture descriptions"
- Solution: Give topology a predicate calculus interpretation, called its prescription
  - components and connectors are types
  - attachments relate ports of components to roles of connectors
- Designer is required to check that the artifact indeed satisfies the predicate

Sample Semantics

Specification:

```plaintext
system contemplate = {
    component c = [parts {in,out}];
    connector x = [roles {i.o}];
    attachments {c.in to x.o, c.out to x.i};
}
```

Prescription:

```plaintext
exists c, x 1
    component(c) ^
    connector(x) ^
    attached(c.in,x,o) ^
    attached(c.out,x,i)
```
Closure and Unique Identification

Specification: system contemplate = {
    component c = {ports {in, out}};
    connector x = {roles {i, o}};
    attachments {c.in to x.o; c.out to x.i};
}

Prescription: exists c, x |
  component(c) ^
  connector(x) ^
  attached(c.in, x.o) ^
  attached(c.out, x.i) ^
  c != x ^
  (for all y: component(y) => y = c) ^
  (for all y: connector(y) => y = x) ^
  (for all p, q: attached(p, q) => (p = c.in ^ q = x.o) I (p = c.out ^ q = x.i))

Property Syntax

- Datatypes:
  - property type position = Record [x, y: Integer; color: Color]
  - property p:position = [55, 66, red]

- External languages:
  - property type FOITL external language
  - property FOITL: exists x l p(x) => afterwards ~p(x)
Additional Properties and Derivatives

Specification:

```plaintext
system test = {
  component c = [ports {in, out}];
  connector x = [roles {i, o}];
  attachments{
    c.in to x.o;
    c.out to x.i;
  }
  property FOITL: delay-line(x),
  derivative FOITL: (o value-at(c.in)) = value-at(c.out)
};
```

Derivative Properties

Prescription:

```plaintext
exists c, x t | 
  component( c ) ^
  connector(x) ^
  attached(c.in, x.o) ^
  attached(c.out, x.i) ^
  c /= x ^
  (for all y:component(y) => y = c) ^
  (for all y:connector(y) => y = x) ^
  (for all p,q: attached(p,q) => (p=c.in ^ q=x.o) l (p=c.out ^ q=x.i)) ^
  delay-line(x) 

Derivative: => (afterwards value-at(c.in)) = value-at(c.out)
```
Low Buy-in Infrastructure

Domain-Specific Software Architectures

- ARPA program advocated investment in infrastructure
- Software Architecture is the key to programming in the large:
  - It forms a contract between the component developers
  - "Reference Architecture" is the basis for refinement of individual components
  - When formalized, forms the specification for a composition mechanism
Three Levels of Environments

Environment Activities
Status

- Exchange architectural information between a diverse set of architectural development and analysis tools

Interchange Experience

- **Wright -> Rapide translation**
  - Initial translation technology developed
  - One-way translation (not round trip)

- **Aesop <-> ACME <-> UniCon**
  - Aesop <-> ACME 1.0 works
  - Aesop <-> ACME 3.0 underway
  - UniCon <-> ACME 3.0 underway
  - Aesop <-> ACME <-> UniCon eventually
Interchange Observations

- One-way translation easier than round-trip
- Subtle semantic differences still a concern
- Expected properties problems not such a big deal
- ACME-based analysis tools perhaps more promising than ADL round-trip translation

Future Interchange Directions

- Translation into ACME
- ACME-based analysis, animation, simulation
  - UniCon "super make" system, providing automatic architecture compilation.
  - Rapide POSet analysis providing event-based deadlock, starvation, etc., analysis.
  - Rapide animator, given an architecture specification and event trace, providing flow visualization.
  - ACME translation to skeletal versions of UniCon, Aesop, Rapide, Wright, etc., as higher buy-in becomes warranted.
Status

- Web access to architectural descriptions
- Baseline tools for
  - manipulation,
  - analysis,
  - change-impact analysis of architectural structures
- that can be universally and transparently invoked from existing ADL platforms.

Infrastructure

- ACME-Lib infrastructure
  - Extensible ACME parsers and unparsers
  - Extensible ACME translation tools
  - Native-ADL embeddable support
  - Support for design traversal, manipulation, and type-checking in ACME-naive tools.
Ongoing Work

- Prototypes for several ACME tools to be provided to the Architecture and Generation EDCS Cluster:
  - an ACME description repository,
  - various analyzers for connectedness and completeness,
  - and a translator from ACME into a predicate calculus - based semantics.
- Prototypes for tools that allow others to provide domain - specific analyzers, such as a
  - code walker
  - ACME elaborator—a tool that translates extended, style - based ACME descriptions into the kernel language.
- Promised
  - ACME type checker
  - Tool to visualize ACME specifications graphically

Status

- Complete ACME language specification, with rationale for its inclusions and exclusions.
- Vehicle for community consensus about architectural representation
  - that evolves over time
  - to accommodate new understandings about requirements for architectural modeling and change.
- Provide a starting point for developing new ADLs
Ongoing Research

- Open semantics (carefully remove closure assumptions)
- Formal definition of the type checking rules
- Tooling infrastructure
- Broader release of TR and user's guide
- Event modeling: temporal logic
- Refinement, evolution, and dynamic behavior represented using graph transformations
- http://www.cs.cmu.edu/~able
  -- click on ACME-web