ABC's requirements of an ADL

- Style must be a first class object
- No deviations from style
  - within a given view
  - multiple views constitute the architecture
- Styles widened to include implicit meanings
  - e.g. exceptions are ubiquitous; should not need to portray them explicitly in the architecture
  - similarly, communication with environment
- ADLs expanded to allow more program specification, scaffolding constructs

ACME:
an Architecture Exchange Language

David Garlan (CMU)
Robert Monroe (CMU)
David Wile (ISI)
Why Interchange?

- **Community consensus**
  - wide variety of tools being invented by the architecture community
  - considerable overlap and redundancy

- **Tool/Capability exchange**
  - n² problem $\rightarrow 2n$
  - tools
    - graphical interface tools
    - animation
    - analysis for deadlock, well-formedness
    - architecture style-specific tools

- **Semantic foundations**
  - refinement
  - event-based

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

![Diagram of ACME Kernel components and connectors](Image)
Hydraulic Press Controller

ACME Controller
Specification

```plaintext
system controller = {
  Idle = component { ports: { to-left, to-right, input} },
  AwaitR = component { ports: { right-closed, right-opened, 
                                left-opened, timed-out, input} },
  AwaitL = component { ports: { left-closed, right-opened, 
                                left-opened, timed-out, input} },
  Activate = component { ports: { pressed-enough, input} },
}
```
ACME Controller
Specification

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

- Refinement
  - substructure specification
  - bindings of external interfaces to internal
- Templates
  - with typed arguments
  - can pass names into the template which are declared during template expansion
  - (or denote "genyms" if not provided at the call site)
- Styles
  - specification as a set of templates and constraints
  - declaration that an architecture obeys a style implies enforcement of style template usage and a subset of the constraints
  - ‘harness’ specification allows provision of style-specific components and connectors

Refinements

```plaintext
system HydraulicControl =
  ctl = component (ports: {input, output})
    representations: [Controller with
      bindings: (* * *)];
  hw = component (ports: {input, output})
    representations: [Plant with
      bindings: (* * *)];
  sensors = connector (roles: {send, receive})
  actuators = connector (roles: {send, receive})
  attachments: {ctl.input = sensors.send;
      ctl.output = actuators.send;
      hw.input = actuators.receive;
      hw.output = sensors.send;}
Styles

**style** StateCharts : {
  template ND-state () = component (ports());
  template transition [s1,s2:state,i,o:name]
    defining c
    = (c=connector. (roles:(input,output)));
    augment $s1 with port:($o);
    augment $s2 with port:($i);
    attachments: ($s1.o to c.input;
                   $s2.i to c.output));

system Plant : StateCharts =
(Closed-left-button = ND-state;
 Open-left-button = ND-state;
 Closed-right-button = ND-state;
 Open-right-button = ND-state;
 Closed-press = ND-state;
 Open-press = ND-state;
 CL = transition Closed-right-button Open-right-button R CL;
 OR = transition Closed-left-button Open-left-button R OL;
 CP = transition Open-press Closed-press CP;
 OP = transition Closed-press Open-press OP)

ACME Future Directions

- **Dynamic architectures**
  - component creation and removal
  - attachment creation and removal

- **Constraints**
  - semantics-based
  - syntax-based

- **Community Demonstration**