ADL Roles

- Provide models, notations, and tools to describe components and their interactions
- Support for large-scale, high-level designs
- Support for principled selection and application of architectural paradigms
- Support for abstractions
  - user-defined
  - application-specific
- Support for implementing designs
  - systematic
  - possibly automated

→ Close interplay between language and environment
  - language enables precise specifications
  - environment makes them (re)usable

An Example ADL Description

- An ACME architecture

```
System simple_cs = {
  Component client = {Port send-request}
  Component server = {Port receive-request}
  Connector rpc = {Port {caller, callee}}
  Attachments : {
    client.send-request to rpc.caller;
    server.receive-request to rpc.callee
  }
}
```
ADL Definition

- **ADL Definition**
  - An ADL is a language that provides features for modeling a software system’s *conceptual* architecture.

- **Essential features:** explicit specification of
  - components
    - interfaces
  - connectors
  - configurations

- **Desirable features**
  - specific aspects of components, connectors, and configurations
  - tool support

Differentiating ADLs

- **Approaches to modeling configurations**
  - implicit configuration
  - in-line configuration
  - explicit configuration

- **Approaches to associating architecture with implementation**
  - implementation constraining
  - implementation independent

- **Related Notations**
  - high-level design notations
  - module interconnection languages (MIL)
  - object-oriented notations
  - programming languages
  - formal specification languages
Example ADL — Wright

- Developed at CMU
- An architecture (**system**) consists of
  - **component** type specifications
    - a set of **ports**
    - specification of component behavior (**comp spec**)
  - **connector** type specifications
    - a set of **roles**
    - specification of role coordination (**glue**)
  - **instances** of types
  - **attachments** among instances
- All specifications are CSP protocols
- Used for detecting deadlocks in an architecture

Example ADL — UniCon

- Developed at CMU
- **Universal Connector language**
- Predefines supported component and connector types
  - **component types**: module, computation, filter, process, ...
  - **connector types**: pipe, file IO, procedure call, data access, remote proc call, ...
- Generates glue code for interconnecting components
  - implementation constraining
- Allows specification of non-functional properties
UniCon syntax

- An architecture is a composite **component** consisting of
  - atomic or composite component types consisting of
    - **interface**
      - predefined set of **players** and attributes
      - predefined set of attributes for each player
    - **implementation**
      - connector types
    - **protocol**
      - predefined set of **roles** and attributes
      - predefined set of attributes for each role
    - **implementation**
      - their instantiations (**uses**)
      - their configurations (**connect**)

---

UniCon Component Player

PLAYER input IS StreamIn
MAXASSOCS (1)
MINASSOCS (1)
SIGNATURE ("line")
PORTBINDING (stdin)
END input

UniCon Connector Role Constraint

ROLE output IS Source
MAXCONNS (1)
ACCEPT (Filter.StreamIn)
END input
UniCon Configuration

USES p1 PROTOCOL Unix-pipe
USES sorter INTERFACE Sort-filter
CONNECT sorter.output TO p1.source
USES p2 PROTOCOL Unix-pipe
USES printer INTERFACE Print-filter
CONNECT sorter.input TO p2.sink

Example ADL — Rapide

- Developed at Stanford
- Focuses on modeling and simulation of dynamic behavior described by an architecture
  - partially ordered event sets (posets)
  - poset-based constraints
  - event patterns to recognize posets
- Large language with several sublanguages
  - types
  - constraint
  - pattern
  - architecture
  - executable
Rapide Event Pattern Operations

- Dependent
- Both
- Distinct
- Either
- Independent
- After
- Iteration
- Guarded pattern
  → useful in specifying dynamic reconfigurations

Rapide Syntax

An architecture consists of
- component types (interfaces)
  - synchronous communication interface elements — functions
    - provides
    - requires
  - asynchronous communication interface elements — events
    - in action
    - out action
- connectors (connections)
- constraints
Rapide Event Patterns

Evt1(?prm) and Evt2(?prm);
Evt1 \rightarrow Evt2;
Evt1 < Evt2;
Evt1(?prm) where ?prm < 0;

Rapide Component

type Application is interface
   in action Request(p : params);
   out action Results(p : params);
behavior
   (?M in String) Receive(?M) => Results(?M);;
end Application;

Example ADL — C2SADEL

- Developed at UC Irvine
- Balances formality and simplicity
  - small number of language constructs
  - semantics described in first-order logic
- Describes C2 components’ internal objects
  - does not include descriptions of requests/notifications
  - separates provided from required component services
  - separates interface from operations
C2SADEL Syntax

- An architecture consists of
  - component types
    - subtype specifications
    - state variables
    - invariant
    - interface
    - behavior (operations)
      - local variables
      - preconditions
      - postconditions
    - map from interface to behavior

C2SADEL Syntax (cont.)

- connector types
  - message filtering policy
    - no filtering
    - notification filtering
    - message filtering
    - prioritized
    - message sink
  - their configurations (architectural topology)
    - component instances
    - connector instances
    - their interconnections
Variables and Basic Types

- Variable declarations are similar to PLs
  - capacity : Integer;
- Variables can also be declared as functions
  - well_at : Integer -> Color;
- C2SADEL only supports declaration of basic types
  - no support for basic type semantics
- Subtyping relationships among basic types are allowed
  - useful for component evolution
  - Natural is basic_subtype Integer;

Component Evolution

- Evolution is supported via subtyping
  - subtyping relationships as regions in the space of types

component WellADT is subtype Matrix (beh)
component WellADT is subtype Matrix (beh \and \not int)
Preconditions, Postconditions, and Invariants

- First-order logic formulas
- Invariants apply to entire components
  - must be expressed in terms of component state variables only
    - invariant {
      (num_tiles $\geq$ 0) \and
      (num_tiles $\leq$ capacity);
    }
- Pre- and postconditions apply to individual operations
  - can be expressed in terms of component state or local operation variables
    - pre (pos $>$ 0) \and
      (pos $\leq$ num_tiles);
    - post \result = well_at(pos) \and
      ~num_tiles = num_tiles - 1;

Separate Interface and Behavior

component WellADT is subtype Matrix (beh) {
  state {
    capacity : Integer;
    num_tiles : Integer;
    well_at : Integer $\rightarrow$ GSColor;
  }
  invariant { (num_tiles $\geq$ 0) \and (num_tiles $\leq$ capacity); }
  interface {
    prov gt1: GetTile (loc : Integer) : Color;
    prov gt2: GetTile (i : Natural) : GSColor;
  }
  operations {
    prov tileget: {
      let pos : Integer;
      pre (pos $>$ 0) \and (pos $\leq$ num_tiles);
      post \result = well_at(pos) \and
      ~num_tiles = num_tiles - 1; }
  }
  map {
    gt1 $\rightarrow$ tileget (loc $\rightarrow$ pos);
    gt2 $\rightarrow$ tileget (i $\rightarrow$ pos); }
}
Merging Interface & Behavior

component WellADT is subtype Matrix (beh) {
state {
  capacity : Integer;
  num_tiles : Integer;
  well_at : Integer -> GSColor;
}

invariant {
  (num_tiles \geq 0) \and (num_tiles \leq capacity);
}

services {
  GetTile (loc : Integer) : Color {
    pre (loc \geq 0) \and (loc \leq num_tiles);
    post \result = well_at(loc) \and \neg num_tiles = num_tiles - 1;
  }
  GetTile (i : Natural) : GSColor {
    pre (i \geq 0) \and (i \leq num_tiles);
    post \result = well_at(i) \and \neg num_tiles = num_tiles - 1;
  }
}

C2SADEL Interface Elements → C2 Messages

- Provided interface elements → incoming requests
  outgoing notifications
- Required interface elements → outgoing requests
  incoming notifications
- This is a greatly simplified correspondence!