C2SADEL Tutorial

Example ADL - C2SADEL

- Developed at UC Irvine and USC
- 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.)

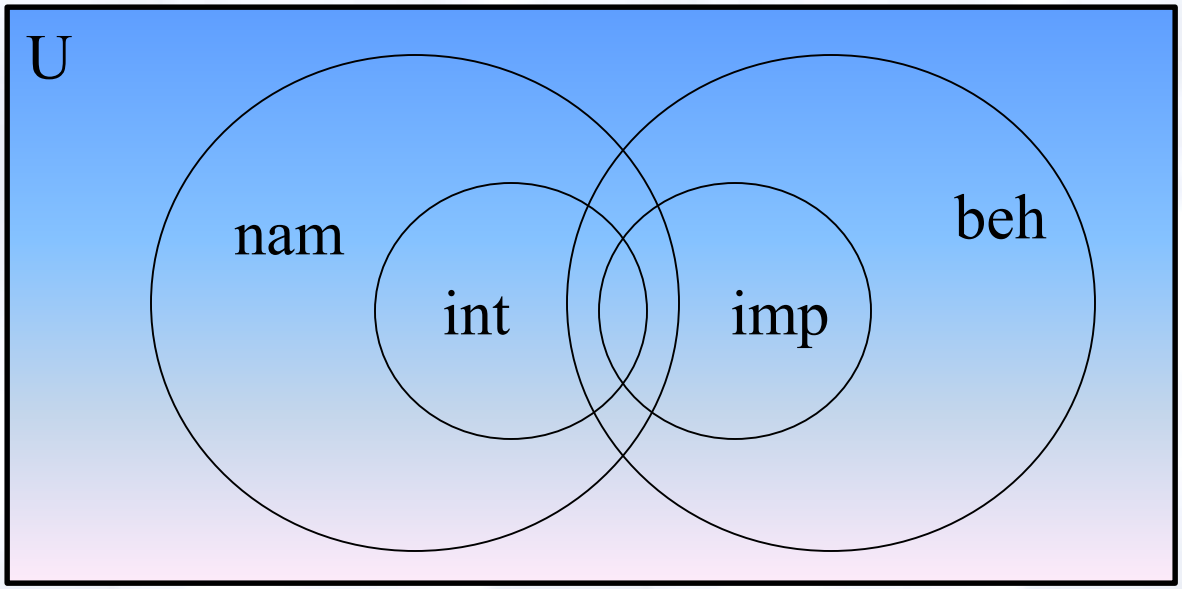
- *connector types*
  - *message filtering policy*
  - *no filtering*
  - *notification filtering*
  - *message filtering*
  - *prioritized*
  - *message sink*
- *their configurations (**architectural topology**)*
  - *component instances*
  - *connector instances*
  - *their interconnections*

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*
  - ***invariant { (num_tiles |eqgreater 0) |and***
    (num_tiles |eqless capacity);}
- Pre- and postconditions apply to individual operations
  - *can be expressed in terms of component state or local operation variables*
  - ***pre (pos |greater 0) |and (pos |eqless num_tiles);***
  - ***post |result = well_at(pos) |and ~num_tiles = num_tiles - 1;***
- Generic way to express *required operations' semantics*
  - ***STATE_VARIABLE basic type***

Separate Interface and Behavior

```
component WellADT is subtype Matrix (beh) {
        state {
                capacity : Integer;
                num_tiles : Integer;
                well_at : Integer -> GSColor; }
        invariant {
                (num_tiles \eqgreater 0) \and (num_tiles \eqless capacity);}
        interface {
                prov gt1: GetTile (location : Integer) : Color;
                prov gt2: GetTile (i : Natural) : GSColor;}
        operations {
                prov tileget: {
                let pos : Integer;
                pre (pos \greater 0) \and (pos \eqless num_tiles);
                post \result = well_at(pos) \and ~num_tiles = num_tiles – 1;}
         }
        map {
                gt1 -> tileget (location -> pos);
                gt2 -> tileget (i -> pos); }
}
```

- *WellADTUser component cannot refer to WellADT's state variables*
  - *capacity, num_tiles, and well_at are referenced as STATE_VARIABLEs*

Merging Interface & Behavior

```
component WellADT is subtype Matrix (beh) {
        state {
            capacity : Integer;
            num_tiles : Integer;
            well_at : Integer -> GSColor; }
        invariant {
            (num_tiles \eqgreater 0) \and (num_tiles \eqless capacity); }
        services {
            GetTile (loc : Integer) : Color
                pre (loc \greater 0) \and (loc \eqless num_tiles);
                post \result = well_at(loc) \and ~num_tiles = num_tiles - 1;
            GetTile (i : Natural) : GSColor
                pre (i \greater 0) \and (i \eqless num_tiles);
                post \result = well_at(i) \and ~num_tiles = num_tiles - 1;
        }
}
```