Why Are Informal Architectural Descriptions Useful?

- Easy to understand
  - e.g., diagrammatic descriptions
- Easy to communicate
  - e.g., to non-technical stakeholders
- Guide system development
  - e.g., via use case scenarios, examples, and rules-of-thumb
- Provide structure to the design process
  - they put us into the proper frame of mind
- Based on architectural intuitions
  - “this should be so because it makes sense”

Example — Informal Architectural Description

>> Is this architecture C2, pipe and filter, client-server, GenVoca, or some hybrid thereof?

Why Are Formal Architectural Descriptions Necessary?

- Imprecision of diagrams makes it difficult to attach unambiguous meanings to the descriptions
  - what does $A \rightarrow B$ mean?
- It is difficult to determine when an implementation is (in)consistent with the more abstract description
- It is virtually impossible to reason rigorously about the descriptions
- It is difficult to compare two different descriptions, even with the same interpretation
  - how do we compare two English paragraphs objectively?

Pros and Cons of Formal Architectural Descriptions

- Advantages
  - precise with provable properties
  - enable formal analysis
  - based on principles, not just intuitions
- Disadvantages
  - difficult to transfer formally proven properties to the implementation
  - if the formal framework is too difficult to understand, is it truly useful?
  - practitioners are likely to avoid it
  - “how many times do I have to study it before I am allowed to give up?”
Formalism in Software Architectures

- Architectural models can be formally described at several levels
  - styles
  - configurations
  - component interactions (i.e., connectors)
  - component interfaces
  - component behaviors
  - component, connector, and system properties
- All of these levels are supportable by ADLs
  - configurations and component interfaces — Darwin
  - and component behaviors — C2SADEL
  - and component interactions — Rapide
  - and styles — Wright
  - properties — UniCon

What Is Formalism Used for in Architectures?

- Rigorous specification
- Constraining the design space
- Analysis
- Understanding and communication
- Refinement
- Evolution
- Implementation generation
- Dynamic change
- ...

Formalizing Style — Z

- Constrains the design space
- Aids communication and understanding
- Enables analysis and evolution
- Possibly enables code generation

Formalizing Configurations — Darwin

```haskell
component Composite {
  provide provserv;
  require reqserv;
  inst
    C1 : CompType1;
    C2 : CompType2;
  bind
    provserv -- C1.pserv;
    C2.rserv -- reqserv;
}
```

Compositional scenario at the architectural level:

```
Composite

C1
provserv
pserv

C2
rserv
reqserv
```

Formalizing Component Behaviors — C2SADEL

```plaintext
component DeliveryPort is
    subtype CargoRouteEntity (int \and beh) {
        state {
            cargo: \set Shipment;
            capacity, selected : Integer;
        }
        invariant {
            (cap >= 0) \and (cap <= max_cap);
        }
        interface {
            prov ip_selshp: Select(sel : Integer);
        }
        operations {
            prov op_selshp: {
                let num : Integer; pre num <= #cargo; post ~selected = num;
            }
        }
        map {
            ip_selshp -> op_selshp (sel -> num);
        }
    }
```

Formalizing Component Interaction — Rapide

```plaintext
type Resource is interface
    public action Receive(Msg : String);
    extern action Results(Msg : String);
    constraint
        match
            ((?S in String)(Receive(?S) -> Results(?S)))^(*~);
end Resource;
```
Formalizing Architectural Refinement — SADL

arch_map MAPPING FROM arch_L1 TO arch_L2
BEGIN
  comp --> (new_comp)
  conn --> (new_comp!subcomp)
  port --> ()
  ...

Formalizing Architectural Refinement — C2 to UML

--1-- C2 attachments are binary associations.
    self.associationEnd->size = 2
--2-- One end of the attachment must be a single C2Component.
    let ends = self.associationEnd in
    ends[1].multiplicity.min = 1 and
    ends[1].multiplicity.max = 1 and
    ends[1].class.stereotype = C2Component
--3-- The other end of the attachment must be a single C2Connector.
    let ends = self.associationEnd in
    ends[2].multiplicity.min = 1 and
    ends[2].multiplicity.max = 1 and
    ends[2].class.stereotype = C2Connector
--4-- Each C2Component has at most one C2AttachOverComp.
    let comps = self.modelElement->select(me | me.stereotype = C2Component) in
    comps->forall(c | c.associationEnd.association->select(a | a.stereotype = C2AttachOverComp)->size <= 1)

Challenges in Formalizing Architectures

- Models must be scalable
- Multiple formal methods must be supported
  - using multiple ADLs to model a single system
  - formalizing different aspects of a system in a single ADL
- Incremental formalization must be supported
  >> how do you formalize in the face of incompleteness?
- Analysis results must be transferable to design and implementation
  >> what good is deadlock detection at architecture alone?
- Key challenges
  - formalize only and exactly as much as necessary
  - combine formal, semi-formal, and informal specs