Introduction

- Architecture is key to reducing development costs
  - development focus shifts to coarse-grained elements
- Formal architectural models are needed
- ADLs have been proposed as a possible answer
- Several prototype ADLs have been developed
  - ACME
  - Aesop
  - ArTek
  - C2
  - Darwin
  - LILEANNA
  - MetaH
  - Rapide
  - SADL
  - UniCon
  - Weaves
  - Wright

→ What an ADL is and its role are still open questions

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
What Does and ADL Description Look Like? (1)

- A Rapide Component

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

- A Wright connector

  ```
  connector Pipe =
  role W = write → W R close → /
  role R =
  let Exit = close → /
  in let DoR = (read R read-eof Exit)
  in DoR R Exit
  ```

  ```
  glue = let ROnly = R.read → ROnly
  in let WOnly = W.write → WOnly
  R.read-eof R.close → /
  R.close → /
  W.close → /
  W.write → glue
  R.read → glue
  W.close → ROnly
  Reader.close → WriteOnly
  ```

What Does and ADL Description Look Like? (2)

- An ACME architecture

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

```
Attempts at Understanding and Classifying ADLs

- Previous ADL surveys
  - Kogut and Clements
  - Vestal

- Insights from individual systems
  - Luckham and Vera
  - Shaw et al.

- Identifying underlying ADL characteristics
  - Tracz
  - Shaw and Garlan
  - Medvidovic, Taylor, and Whitehead
  - Medvidovic and Rosenblum

- Architecture interchange
  - ACME

Example Attempts at Understanding ADLs

- Shaw and Garlan
  - composition
  - abstraction
  - reusability
  - (re)configuration
  - heterogeneity
  - analysis

- Tracz
  - components
  - connectors
  - configurations
  - constraints
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

ADL Components

- Definition
  - A *component* is a unit of computation or a data store. Components are loci of computation and state.

- All ADLs support component modeling

- Differing terminology
  - *component*
  - *interface*
  - *process*
## Component Classification Categories

- **Interfaces**
  - model both required and provided services

- **Types**
  - enable reuse and multiple instances of the same functionality

- **Semantics**
  - facilitate analyses, constraint enforcement, and mapping of architectures across levels of refinement

- **Constraints**
  - ensure adherence to intended component uses, usage boundaries, and intra-component dependencies

- **Evolution**
  - components as design elements evolve
  - supported through subtyping and refinement

- **Non-Functional Properties**
  - enable simulation of runtime behavior, analysis, constraints, processor specification, and project management

### Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Interface</th>
<th>Types</th>
<th>Semantics</th>
<th>Constraints</th>
<th>Evolution</th>
<th>Non-Funct. Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aesop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Darwin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MetaH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rapide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UniCon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weaves</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wright</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### ADL Connectors

- **Definition**
  - A *connector* is an architectural building block used to model interactions among components and rules that govern those interactions.

- **All ADLs support connector modeling**
  - several ADLs do not model connectors as first-class entities
  - all ADLs support at least syntactic interconnection

- **Differing terminology**
  - *connector*
  - *connection*
  - *binding*

### Connector Classification Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interfaces</strong></td>
<td>ensure proper connectivity and communication of components</td>
</tr>
<tr>
<td><strong>Types</strong></td>
<td>abstract away and reuse complex interaction protocols</td>
</tr>
<tr>
<td><strong>Semantics</strong></td>
<td>analyze component interactions, enforce constraints, and ensure consistent refinements</td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td>ensure adherence to intended interaction protocols, usage boundaries, and intra-connector dependencies</td>
</tr>
<tr>
<td><strong>Evolution</strong></td>
<td>maximize reuse by modifying or refining existing connectors</td>
</tr>
<tr>
<td><strong>Non-Functional Properties</strong></td>
<td>enable simulation of runtime behavior, analysis, constraint enforcement, and selection of OTS connectors</td>
</tr>
</tbody>
</table>
Architecture Description Languages (ADLs)

<table>
<thead>
<tr>
<th>Connectors</th>
<th>Interface</th>
<th>Types</th>
<th>Semantics</th>
<th>Constraints</th>
<th>Evolution</th>
<th>Non-Func. Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aesop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Darwin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MetaH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rapide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UniCon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weaves</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wright</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ADL Configurations**

- **Definition**
  - An *architectural configuration* or *topology* is a connected graph of components and connectors which describes architectural structure.
- **ADLs must model configurations explicitly by definition**
- **Configurations help ensure architectural properties**
  - proper connectivity
  - concurrent and distributed properties
  - adherence to design heuristics and style rules
Configuration Classification Categories (1)

- **Understandability**
  - enables communication among stakeholders
  - system structure should be clear from configuration alone

- **Compositionality**
  - system modeling and representation at different levels of detail

- **Heterogeneity**
  - development of large systems with pre-existing elements of varying characteristics

- **Constraints**
  - depict dependencies among components and connectors

Configuration Classification Categories (2)

- **Refinement and Traceability**
  - bridge the gap between high-level models and code

- **Scalability**
  - supports modeling of systems that may grow in size

- **Evolution**
  - evolution of a single system or a system family

- **Dynamism**
  - enables runtime modification of long-running systems

- **Non-Functional Properties**
  - enable simulation, analysis, constraints, processor specification, and project management
### ADL Tool Support

- Formality of ADLs enables their manipulation by tools
  - toolset is not part of an ADL
  - usefulness of an ADL depends on its support for architecture-based development
- Every ADL provides some tool support
- Focus typically on a particular area and/or technique
- Limited overall support motivated the need for architectural interchange
  - ACME
Tool Support Classification Categories

- **Active Specification**
  - support architect by reducing cognitive load
  - proactive vs. reactive

- **Multiple Views**
  - support for different stakeholders

- **Analysis**
  - upstream evaluation of large, distributed, concurrent systems

- **Refinement**
  - increase confidence in correctness and consistency of refinement

- **Code Generation**
  - ultimate goal of architecture modeling activity
  - manual approaches result in inconsistencies and lack of traceability

- **Dynamism**
  - enable changes to architectures during execution

---

<table>
<thead>
<tr>
<th>Tool Support</th>
<th>Active Specification</th>
<th>Multiple Views</th>
<th>Analysis</th>
<th>Refinement</th>
<th>Code Generation</th>
<th>Dynamism</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aesop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Darwin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MetaH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rapide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SADL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UniCon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weaves</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wright</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Discussion

- Goal: distinguish different kinds of ADLs
- ADL definition is a simple litmus test
- Several ADLs straddle the boundary
  - implementation constraining languages
  - in-line configuration languages
- Support extensive in certain areas, lacking in others
  - implementation of complex connectors
  - non-functional properties
  - refinement
  - dynamism
- Determine relative “value” of an ADL
- Aid development of ADLs
- Aid architecture interchange
  - identifying complementary ADLs