CS560
Lecture: Architecture Description Languages

A. O’Riordan, 2009
What is an Architecture Description Language?

- modelling notation to support architecture-based development

- used to define and model system architecture prior to detailed design and implementation

- little consensus as to what is an ADL, what aspects of an architecture should be modelled in an ADL, and which of several possible ADLs is best

- Related to OO Modelling Languages and Module interconnection languages (MILs)
Many ADLs

Architecture Description Languages (ADLs) can be used to specify an architecture

- UML (OMG) - general-purpose
- SADL (SRI)
- Aesop (Carnegie Mellon University)
- Acme (CMU) – and interchange format
- Rapide (Stanford University)
- Wright (CMU)
- Darwin (Imperial College London)
Parts of an ADL

Architecture Style = \{Component/Connector Vocabulary, Topology, Semantic Constraints\}

Components (locus of computation), e.g. filter, data store, object, process, server
- ADLs typically provide support for specifying both functional and non functional characteristics of components.

Connectors (interactions between components), e.g. procedure call, RPC, pipe, TCP/IP

Key issue - tool support
Acme: a generic ADL

- “Acme is a simple, generic software architecture description language (ADL) that can be used as a common interchange format for architecture design tools and/or as a foundation for developing new architectural design and analysis tools.”
  [http://www.cs.cmu.edu/~acme/](http://www.cs.cmu.edu/~acme/)

- Developed by David Garlan and others (CMU) and David Wile (USC)
- Simple, generic language for describing software architectures and families of architectures
- Intended to be standard representation for tools
- Acme is an extendable language - extensions of Acme include Armani
Tools – ACME Studio

ACME Studio - a viewer and editor for ACME descriptions. Text and graphical views supported. Additional analysis tools can be incorporated using ACMELib.
ACME Overview

- Provides constructs for describing systems as graphs of components interacting via connectors, a representation mechanism for hierarchical decomposition of components and connectors into subsystems
- Does not provide a specific model for describing system behaviour, instead elements may be annotated with properties that represent this information
- Rather than providing a fixed set of models for formulating solutions, Acme provides general, domain-neutral foundation for developing new tools and notations
- Structures provided by Acme are based on the informal box and line diagrams traditionally used to depict the architecture of the system
- Acme helps a designer document design decisions and reason about the implications of those decisions
Language Overview

- The Acme language provides the following key features:
  1. an *architectural ontology* consisting of seven basic architectural design elements;
  2. a flexible *annotation mechanism* supporting association of non-structural information using externally defined sublanguages;
  3. a *type mechanism* for abstracting common, reusable architectural idioms and styles; and
  4. an *open semantic framework* for reasoning about architectural descriptions.

- Acme is built on a core ontology of seven types of entities for architectural representation: *components, connectors, systems, ports, roles, representations, and rep-maps*.

Language Overview (2)

- Acme supports the hierarchical description of architectures - any component or connector can be represented by one or more detailed, lower-level descriptions.

- Each ADL typically has its own set of auxiliary information that determines such things as the run-time semantics of the system, typing information, protocols of interaction, scheduling constraints, and information about resource consumption.

- Acme supports annotation of architectural structure with lists of properties. Each property has a name, an optional type, and a value. Properties become useful only when a tool makes use of them.
Components

- Components are the basic building blocks in an Acme description. They represent centres of computation: the elements responsible for doing the work.
- Part of architectural design is properly separating concerns about communication from computational concerns.
- A component includes properties which can be used to describe aspects of its computational behaviour or structure. Properties provide a way of encoding information.
- Typical use of a property is to describe the actual computation performed by a component using a language that provides a more abstract depiction of the computation than a programming language.
Component Examples

- Example:

```java
Component TheFilter = {
  Port in;
  Port out;
  Property implementation : String =
    "while (!in.eof) {
      in.read; in.read; compute; out.write)";
  }
}
```

- A property may also be used to describe a non-functional property of a component, e.g. fault-tolerance, memory requirements or performance

```java
Component Server = {
  Port requests;
  Property responsetime : Float = 15.00 << units='ms'>;
  }
```
Connectors

- Represent communication glue that captures the nature of an interaction between components.
- Typical connector might define a synchronization model for communication, a communication protocol, a locking model, or characteristics of a communication channel like bandwidth.
  - A connector might embody a particular protocol (like HTTP).
  - A connector may also be used to represent data flow channel (e.g., a Pipe) or an asynchronous communication medium like an event bus.
- A connector includes a set of interfaces in the form of role - examples of simple roles include the reading and writing roles of a pipe or the sender and receiver roles of a message passing connector.
Example:

Connector AnEventBus = {
// All the events broadcast by this connector
Property eventList = <
    "WorldWar3Begins",
    "RainToday" >;
// Two different listener roles
Role curiousListener;
Role selfInvolvedListener;
}

Systems

- components and connectors assembled into systems
- define properties which describe "system-level" attributes - property may be used to represent properties of the environment in which the system is operating, or "global" properties

```plaintext
System ClientServerSystem = {
  Component server = {
    Port requests; 
  }
  Component client1 = {
    Port makeRequest; 
  }
  Connector req = {
    Role requestor;
    Role requestee; 
  }
  Attachments {
    server.requests to req.requestor;
    client.makeRequest to req.requestee; 
  }
}
```
Simple Client-Server System in ACME

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;
    }
}
Schematic of Structure
Properties store auxiliary information

System simple_cs = {
    Component client = {
        Port send-request;
        Property Aesop-style : style-id = client-server;
        Property source-code : external = "CODE-LIB/client.c";
    }
    Component server = {
        Port receive-request;
        Property idempotence : boolean = true;
        Property max-concurrent-clients : integer = 1;
        source-code : external = "CODE-LIB/server.c";
    }
    Connector rpc = {
        Role caller;
        Role callee;
        Property asynchronous : boolean = true;
        max-roles : integer = 2;
        protocol : Wright = "...";
    }
    Attachment client.send-request to rpc.caller;
    Attachment server.receive-request to rpc.callee;
}