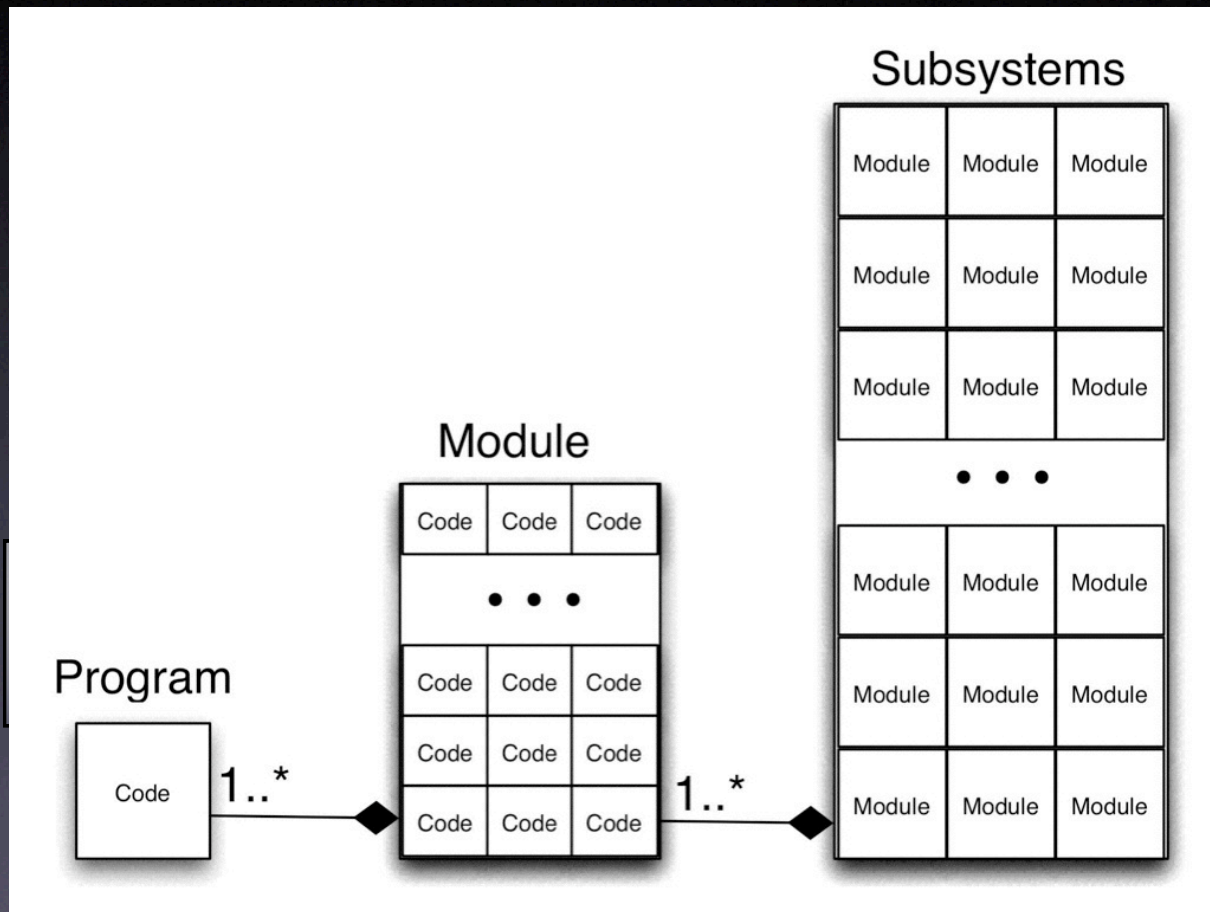
Modeling Dynamic Architectures

Linus Sherrill
CSC 299 - Spring 2005
Graduate College at Union University

Famous failures

- Baggage handling system in Denver
- FBI case management system
- FAA traffic control system

Levels of complexity



Software Architecture

- Large scale representation
- Help plan system construction
- Reason about system properties
- Clarify intent and assumptions
- Description languages

Dynamic Systems

- Change at run time
  - create, delete, reconnect components
  - change configuration
- Example: web server
  - Server creates new handlers for requests

Agenda

- Why Software Architecture
- Architecture Description Languages
- Example dynamic architecture
- How well do ADL represent it
- Conclusions / Recommendations

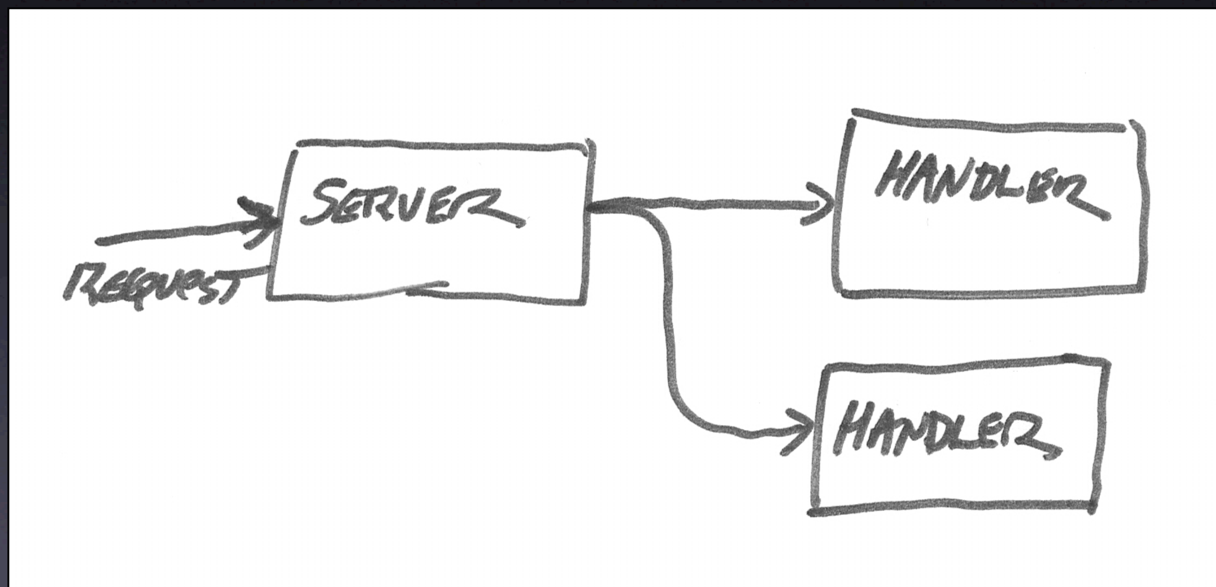
Architecture Description Languages

- Capture structure
- Model behavior
- Documentation

Methodology

- Select dynamic architecture
    - Server creates new handlers for requests
- Model in different ADLs
- Assess Results

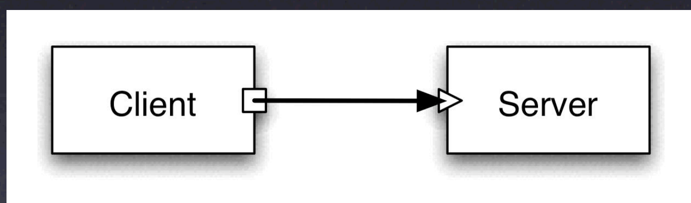
Typical Architecture Diagram



Acme

- Emphasizes structural aspects
- Good graphical support
- Commercial tools available (Eclipse)
- No native dynamic support
- Extensible through properties

Acme Description

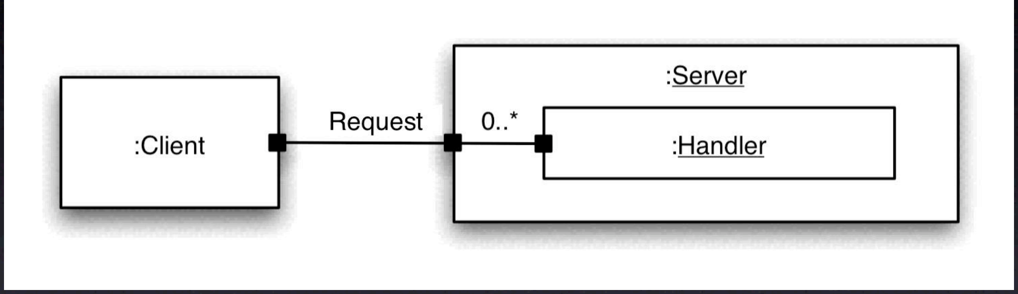


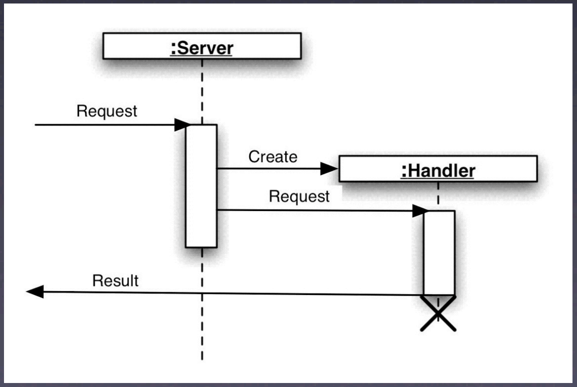

```
system simple_cs = {
  component client = {
    port send-request;
    properties { unicon-style:style-id = cs;
      source-code:external = “CODELIB/client.c”
    }
  }

  component server = {
    port recv-request;
    properties {
      max-concurrent-clients:integer=1;
      source-code:external = “CODELIB/server.c” }
  }

  connector rpc = {
    roles {caller, callee}
    properties { synchronous:Boolean=true;
      max-roles:integer=2 }
}

  attachments : {
    client.send-request to rpc.caller;
    server.recv-request to rpc.callee }
}
```

UML

- De-facto standard for modeling
- Generally understood
- Good tools support
- Needs to be adapted to architectural modeling

UML Description

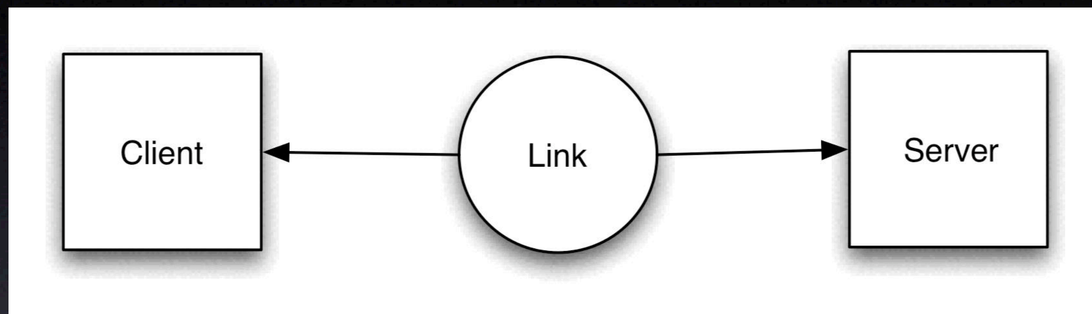




Wright

- Graph based
- Graphical representation
- Based on a variant of CSP

Wright



Style Client-Server

Component Client

Port p = $\overline{\text{request}} \rightarrow \text{reply} \rightarrow \text{p} \sqcap \S$

Computation = $\text{internalCompute} \rightarrow \overline{\text{p.request}} \rightarrow \text{p.reply} \rightarrow \textbf{Computation} \sqcap \S$

Component Server

Port p = $\text{request} \rightarrow \overline{\text{reply}} \rightarrow \text{p} \mathbin{[]} \S$

Computation = $\text{p.request} \rightarrow \text{internalCompute} \rightarrow \overline{\text{p.reply}} \rightarrow \textbf{Computation} \mathbin{[]} \S$

Connector Link

Role c = $\overline{\text{request}} \rightarrow \text{reply} \rightarrow \text{c} \sqcap \S$

Role s = $\text{request} \rightarrow \overline{\text{reply}} \rightarrow \text{s} \mathbin{[]} \S$

Glue = $\text{c.request} \rightarrow \overline{\text{s.request}} \rightarrow \textbf{Glue}$

$[]$ $\text{s.reply} \rightarrow \overline{\text{c.reply}} \rightarrow \textbf{Glue}$

$[]$ $\S$

Constraints

$\exists!\, s \in \text{Component}, \forall\, c \in \text{Component} : \text{TypeServer}(s) \wedge \text{TypeClient}(c) \Rightarrow \text{connected}(c,s)$

EndStyle

Configuration Simple

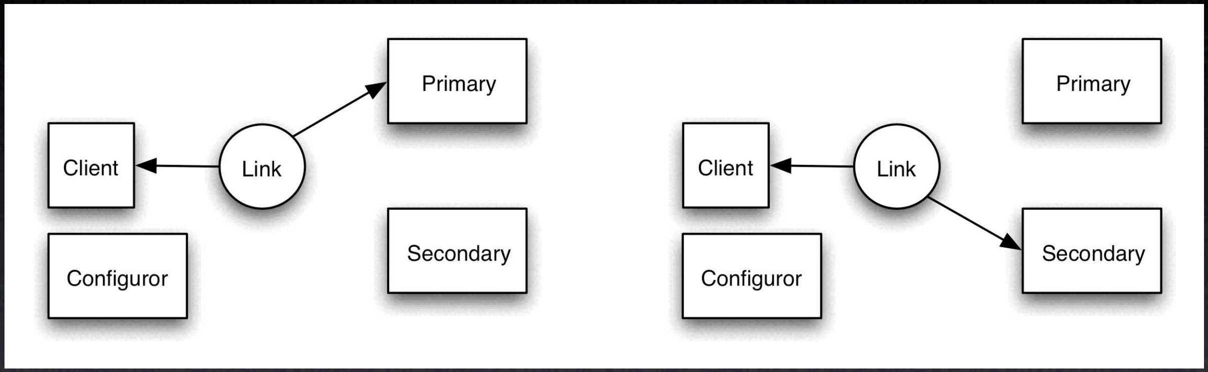
Style Client-Server

Instances C : Client ; L : Link ; S : Server

Attachments C.p **as** L.c ; S.p **as** L.s

EndConfiguration

Dynamic Wright



- Modeling dynamic architectures
- Dynamism encapsulated in configuror
- Finite set of configurations

Example Architecture

Style Client-Server
  Component Client
    Port p = $\overline{\text{request}} \rightarrow \text{reply} \rightarrow \text{p} \sqcap \S$
    Computation = $\text{internalCompute} \rightarrow \overline{\text{p.request}} \rightarrow \text{p.reply} \rightarrow \textbf{Computation} \sqcap \S$

  Component Server
    Port p = $\text{request} \rightarrow \overline{\text{reply}} \rightarrow \text{p} \;[]\; \S$
    Computation = $\text{p.request} \rightarrow \text{internalCompute} \rightarrow \overline{\text{p.reply}} \rightarrow \textbf{Computation} \;[]\; \S$

  Connector Link
    Role c = $\overline{\text{request}} \rightarrow \text{reply} \rightarrow \text{c} \sqcap \S$
    Role s = $\text{request} \rightarrow \overline{\text{reply}} \rightarrow \text{s} \;[]\; \S$
    Glue = $\text{c.request} \rightarrow \overline{\text{s.request}} \rightarrow \textbf{Glue}$
      $[]\; \text{s.reply} \rightarrow \overline{\text{c.reply}} \rightarrow \textbf{Glue}$
      $[]\; \S$

  Constraints
    $\exists!\, s \in \text{Component}, \forall\, c \in \text{Component} : \text{TypeServer}(s) \wedge \text{TypeClient}(c) \Rightarrow \text{connected}(c,s)$
EndStyle

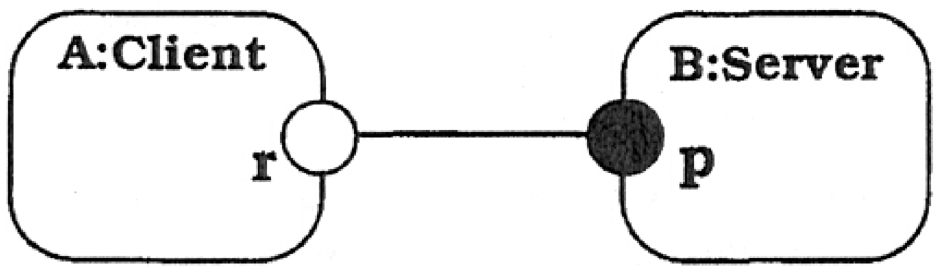
Configuration Simple
  Style Client-Server
  Instances C : Client ; L : Link ; S : Server
  Attachments C.p **as** L.c ; S.p **as** L.s
EndConfiguration

- Put dynamism in *internalCompute*

Darwin

- Configuration language
- π-Calculus based
- Directly supports dynamism
  - Lazy instantiation
  - Dynamic instantiation

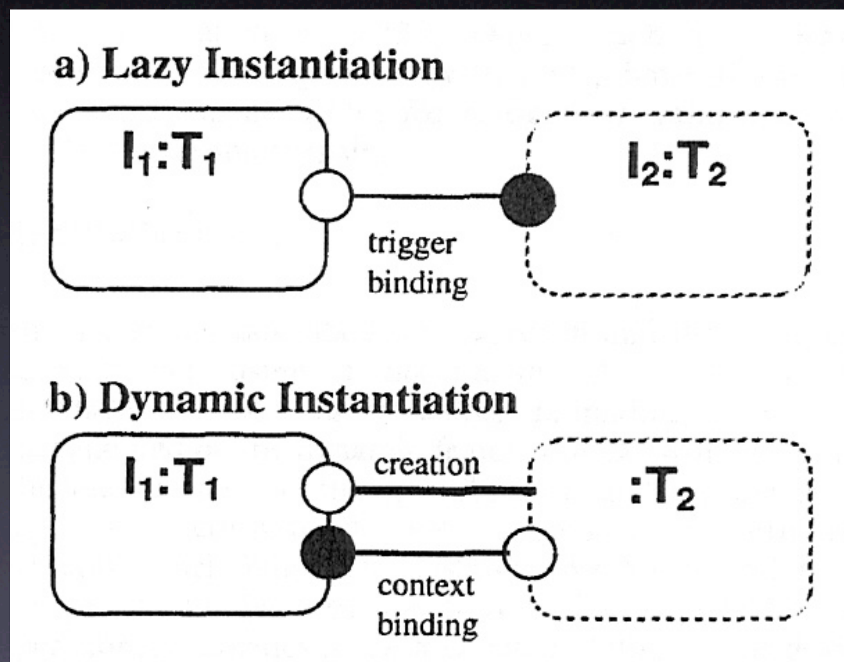
Darwin Constructs



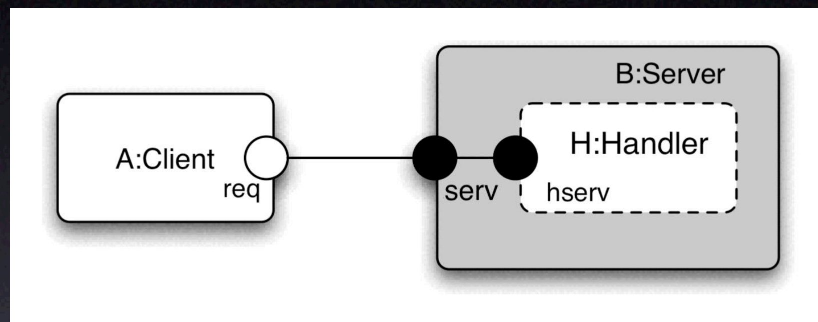

```
component Server {
  provide p;
}

component Client {
  require r;
}
```

```
component System {
inst
   A:Client;
   B:Server;
bind
   A.r -- B.p;
}
```

Dynamic Modes



Darwin Representation




```
 component Client {
   require req;
}

 component Handler {
   provide hserv;
}

 component Server {
   provide serv;

   inst
     H:dyn Handler;

   bind
     B.serv -- H:hserv;
 }
```

```
 component System {
   inst
     A:Client;
     B:Server;

   bind
     A.req -- B.serv;
}
```

Summary

- Darwin – good dynamism support
- Wright – some dynamism
- UML – well known
- Acme – flexible

Conclusion

- Individual ADLs have narrow focus
- Fragmented support
- Darwin is dynamic
- Combine strengths

Use Acme

- Structural representation
- Extensible through properties
- Transfer model to other representations

- Comments?
- Questions?
- Bitter invective?