Wright ADL

Wright Model

- Components - computation elements with multiple ports
- Connectors - interaction - first class - consists of roles and glue
- Configuration - components and connectors related into a system

Example

System SimpleExample

component Server =

port provide = [*provide protocol*]

computation =

component Client =

port request = [*request protocol*]

computation =

connector C-S-connector =

role client = [*client protocol*]

role server = [*server protocol*]

glue = [*glue protocol*]

Instances

s: Server

c: Client

cs: C-S-connector

Attachments

s.provide **as** cs.server

c.request **as** cs.client

Describing connection

- Should be able to express common cases of architectural interaction (pipes, events, procedure call)
- Should allow description of complex dynamic interactions between components (for example, that a connection must be initialized before use)
- Should allow distinctions between connector variations
- Should be based on analyzable formal model

Communicating Sequential Processes - CSP

- Processes and events - events may have input (e?x) and output (e!x) data
- Prefixing e --> P
- Internal/external choice (| and □)
- Parallel Composition -- P || Q - joint interaction over events in intersection of alphabets of P and Q
- Special symbol for successful termination ⌘
- Scoped process names (let Q = … in R)

Why CSP?

- Other options: Petri Nets, SDL, I/O Automata, StateCharts
- In CSP, we can capture the distinction between internal and external choice
- CSP has parallel composition
- Tool support (theorem proving)
- Disadvantages: timing, fairness not addressed in CSP

Wright Connectors

- Connectors describe the behavior of connection
- Roles - local behavior of the interaction parties - the obligations of each participant in the interaction
- Glue - describes how the activities of the roles are coordinated
- Glue || role-1 || role-2 || . . . || role-n

Simple Pipe

```
connector pipe =
  port source = in!x --> source
  port sink = out?y --> sink
  glue = source.in?x --> sink.out!x --> glue
```

Simple Client/Server

connector C-S-connector =

role Client = (request!x --> result?y --> Client) | ⌘

role Server = (invoke?v --> return!w --> Server) □ ⌘

glue = (Client.request?x --> Server.invoke!x -->
Server.return?y --> Client.result!y --> glue)

Another Pipe

```
connector Pipe =
  role Writer = write --> Writer  |  close -->⌘

  role Reader = let ExitOnly = close -> ⌘
                in let DoRead = (read --> Reader
                                 □ read-eof --> ExitOnly)
                   in DoRead  |   ExitOnly
```

```
glue = let ReadOnly = Reader.read --> ReadOnly
                    □ Reader.read-eof --> Reader.close --> ⌘
                    □ Reader.close --> ⌘
       in let WriteOnly = Writer.write --> WriteOnly
                                Writer.close --> ⌘
          in Writer.write--> glue □
             Reader.read --> glue □
             Writer.close --> ReadOnly □
             Reader.close --> WriteOnly
```

connector shared_memory =

role User1 = set --> User1 | get --> User1 |

role User2 = set --> User2 | get --> User2 |

glue = User1.set --> glue □ User2.set --> glue □

User1.get --> glue □ User2.get --> glue □ ⌘

Wright Components

- Port - logical point of interaction between a component and its environment (I.e. the rest of the system) - Defines the expectations of the component.
- A component may have multiple ports
- Computation - describes the relationship between the ports

Example

```
system System
  component Split =
     port input = getchar?x --> input
     port output1 = putchar!x --> output1
     port output2 = putchar!x --> output2
     computation = input.getchar?y --> output1.putchar!y
                  --> input.getchar?y --> output2.putchar!y -->
                 computation
component filter =
    port inport = get?x --> inport
    port outport = put!x --> outport
    computation = inport.get?x --> outport!x --> computation
```

```
component Merger
  port input1 = get?c --> input1
  port input2 = get?c --> input2
  port output = put!c --> output
  computation = input1.get?c --> output.put!c -->
            input2.get?d --> output.put!d --> computation
```

Wright Configuration

- Describe the system structure
- Instances - instantiate some components and connectors of given types
- Attachments - bind the port of a component to the role of a connector
- Hierarchical

Instances

s : Split

l, u : Filter

m: Merger

p1, p2, p3, p4: Pipe;

Attachments

s.output1 **as** p1.source

s.output2 **as** p2.source

l.inport **as** p1.sink

u.inport **as** p2.sink

l.outport **as** p3.source

u.output **as** p4.source

m.input1 **as** p3.sink

m.input2 **as** p4.sink