Component based software

Building concurrent systems from reusable parts

David Knight Apr05
Component construction

• A Component:
  – Encapsulates some interesting behaviour
  – Has inputs and/or outputs

• An input or output can be:
  – A *Signal*: A value of any type that can be changed at any time without warning
  – An *Event*: A notification that something has happened, accompanied by optional parameters of any type
A component, in pictures

ClassName

InstanceName

inSignal inEvent outSignal outEvent
Interfaces

- An *interface* is a group of logically-related signals and/or events that are treated as a unit
  - An interface can contain a mixture of in and out signals or events
- A component can *implement* an interface, in which case it must provide inputs and outputs as described by the interface
- A component can implement an interface in *reverse*, in which case the in/out direction of all signals/events is reversed
- A component can implement multiple interfaces, including the *same* interface multiple times
- An interface can also contain other interfaces, in normal or reversed direction, to arbitrary depth
Interfaces in pictures

ComponentName

Normal direction interface

Reverse direction interface
Connection rules

• An output of a component can be connected to the input of one or more other components (including itself)
• Interconnection is only permitted between signals and events of the same type
• An interconnected network of components can itself be viewed as a component. This can be carried to any depth.
Connection in pictures

C1

\[ \text{ClkIn} \quad \text{Carry} \quad \text{Enable} \quad \text{ClkOut} \]

C2

\[ \text{A} \quad \text{Error} \quad \text{B} \quad \text{Run} \]

U1

U2

U3

\[ \text{ClkIn} \quad \text{Carry} \quad \text{Enable} \quad \text{ClkOut} \]
Component interfaces

• The external view of a component is specified by an interface.
• Several different components can implement the same interface, and thus would be plug-compatible.
Component interface in pictures

- ByteStream
- inStream
- clock
- enable
- modem
- error
- ready
- speed
Interface as text

```java
interface Example;
    ByteStream inStream;
    in event clock();
    in boolean enable;
    reverse Dte modem;
    out event error(int reason);
    out boolean ready;
    out int speed;
endinterface;
```
Component as text...

class cName implements iName {
    interface {
        in int ival;
        in event clock();
        out boolean ready;
    }
    static {
        Doofer d = new Doofer(...);
        Scronge s = new Scronge(...);
        Gadget g = new Gadget(...);
        connect clock d.tick g.clk s.clock;
        connect s.ready g.enable;
        connect s.out ready;
        connect ival d.speed;
    }
    dynamic {
        (see next slide)
    }
}
...component as text

dynamic
    machine mName;
    initial
        initialisationActions;
        next s1;
    state s1;
        entryActions;
        when event1(int b)
            doThis;
            next s2;
        when event2();
            doThat;
            next s1;
        endwhen;
    state s2;
        etc...
endmachine;
endcomponent;
Example 1
Simple producer/consumer

See the pages:
- ByteStream.ifc
- Producer.ifc
- Consumer.ifc
- Example.ifc
- Producer.cmp
- Consumer.cmp
- Pc.cmp
Example 1 - interfaces

ByteStream
  data

Example
  clock

Producer
  Clock
  stream

ByteStream
  stream

Consumer
  ByteStream
  stream
Pc.cmp
Example 2
Bounded queue

See files
  Queue.ifc
  Queue.cmp
  Pqc.cmp
  Pqqc.cmp
Queue interface
Pqc.cmp
Pqqc.cmp
Example 3
Async data transmission

See files
AsyncTransmitter.ifc
AsyncReceiver.ifc

AsyncTransmitter.cmp
AsyncReceiver.cmp
Ptrc.cmp
Pqtrqc.cmp
Example 3 interfaces

AsyncTransmitter
- stream
- clock
- txd

AsyncReceiver
- rxd
- stream
- Clock

ByteStream
StatMux/StatDemux

• A StatMux interleaves two data independent streams onto a single channel, to make better use of the channel data capacity

• A StatDemux reverses the process, and separates the two streams again