Considerations on Object-Oriented Extensions to VHDL

Peter J. Ashenden
University of Adelaide
Visiting Scholar at U. Cincinnati

Philip A. Wilsey
University of Cincinnati

OO or High-Level Modeling?

- Need to better support high-level modeling
  - specify data and behavior in a more abstract manner
- OO is part of that, not a panacea
- VHDL is already “object-based”
- Need to improve facilities
  - abstraction, encapsulation, concurrency and communication

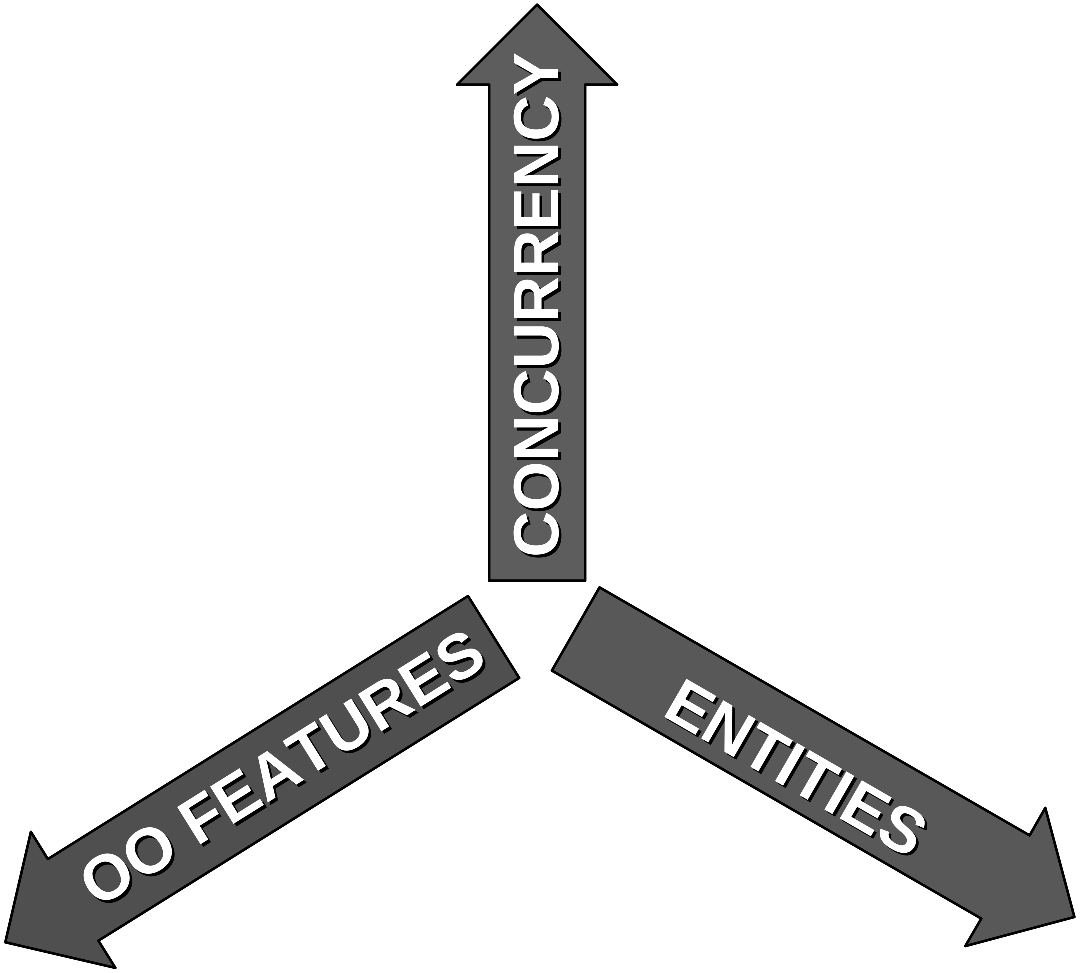
Extension Principles

- Focus on semantics
  - syntax follows
- Aim for simplicity and orthogonality
  - clear interactions between features
- *Integrate*: maintain conceptual integrity
  - build on existing language features and philosophy

A Rough Taxonomy

- Data modeling
  - programming language ideas
- Structure modeling
  - inheritance of generics/ports in entities, concurrent statements in architectures
- System-level modeling
  - e.g., before hardware/software partitioning

Separation of Concerns



Concurrency

- Extend existing concurrency and communication features
  - – e.g., dynamic creation of processes
  - – e.g., abstract communication
    - message passing, RPC/rendezvous
- Monitors are insufficient
  - – they are just *concurrency control* for encapsulated data

Concurrency Example

```
type elevator_class is class

        channel elevator_call : in floor_number;
        channel elevator_location : out floor_number;

        elevator : process is
            . . .
        begin
            . . .
            receive calling_floor from elevator_call;
            send current_floor to elevator_location;
            . . .

        end process;

    end class;
```

Data Modeling

- “Programming by extension” à la Ada-95
- Class-based à la C++
- What about signal objects?
  - use class-provided variable assignment and equality for signal assignment and update

Data Modeling Example

```
type complex is class
        private variable re, im : real;
        public procedure ":=" ( c : complex );
        public function "=" ( right : complex )
                                return boolean;
        . . .
    end class;
signal s1, s2 : complex;

s1 <= complex(0.0, 1.0);
wait on s2;
```

Encapsulation: Private Parts

Genericity

- *c.f.* template functions and classes in C++
- *c.f.* generics in Ada
- Example:

```
entity shift_reg is
    generic ( type item is private;
              type index is (<>);
              type vector is array (index) of item );
    port ( shift_clk : in bit;  data_in : in item;
           data_out : out vector );
end entity;
```

Synthesis

- Don’t forget it!
- Behavioral synthesis
- Hardware/software co-synthesis
- Use of new features across the modeling spectrum

Conclusions

- Simple, regular extensions in keeping with existing language
- Carefully analyze alternatives and consider interactions
- Need to take a holistic view
- OO is part of the picture, not all of it