Data Modeling in VHDL

Peter J. Ashenden
The University of Adelaide

Philip A. Wilsey
University of Cincinnati

This work was partially supported by Wright Laboratory
under USAF contract F33615-95-C-1638.

Complexity Management

- For system-level design of behavior
  - abstraction of data
  - abstraction of communication & timing
  - abstraction of concurrency
- SUAVE:
  - SAVANT & University of Adelaide VHDL Extensions
  - object-oriented data modeling
  - genericity
  - communication & concurrency

Design Objectives

- Support high-level modeling
  - improve encapsulation and information hiding
  - provide for hierarchies of abstraction
- Support re-use and incremental development
  - polymorphism, dynamic binding
- Support hw/sw codesign
  - improved integration with programming languages
- Preserve correctness of existing models
- Design principles from VHDL-93
  - preserve “conceptual integrity”

Class-Based Approach

- Example: Objective VHDL (Radetzki *et al*)
- *Class* is a specific language construct
  - encapsulates data and operations
  - may inherit from a superclass
  - an object is an instance of a class
- *cf.* Simula, Smalltalk, C++, Java

Programming by Extension

- Example: SUAVE (Ashenden & Wilsey)
- *Class* is a type-derivation hierarchy
  - package encapsulates a tagged type and operations (ADT)
  - a new type may extend a parent type
    - inherits operations
  - T'Class denotes type hierarchy rooted at T
  - an object is of a tagged type or class-wide type
- *cf.* Oberon-2, Ada-95

Integration with VHDL Signals

- Variables denote machine locations
- Signals denote trajectories of values
  - assignment: editing scheduled trajectory
  - update: resolution; type conversions on net
  - event: change of value triggers processes

Classes and Signals

- Problem: data in hidden in the class
  - behavior of operation depends on kind of object
- Objective VHDL solution:

```
type complex is class
    class attribute re, im : real;
    impure function real_part ( x : complex ) return real;
    impure function imag_part ( x : complex ) return real;
    procedure add ( y : complex );
end class complex;
```


Classes and Signals

```
type complex is class body
    . . .
    for signal
        procedure add ( y : complex ) is
        begin
            re <= re + y.real_part;
            im <= im + y.imag_part;
        end procedure add;
    end for;
    for variable
        procedure add ( y : complex ) is
        begin
            re := re + y.real_part;
            im := re + y.imag_part;
        end procedure add;
    end for;
end class complex;
```

ADTs and Signals

- Package defines a type
  - signals declared to be of the type
  - assignment/update" use predefined ":=", "="
  - signal/variable operations distinguished by parameter kind if necessary
- SUAVE solution: ...

ADTs and Signals

```
package complex_numbers is
    type complex is private;
    . . .
    function "+" ( x, y : complex ) return complex;
private
    type complex is record
        re, im : real;
    end record complex;
end package complex_numbers;

package body complex_numbers is
    . . .
    function "+" ( x, y : complex ) return complex is
    begin
        return ( x.re + y.re, x.im + y.im );
    end function "+";
end package body complex_numbers;
```

ADTs and Signals

```
use complex_numbers.all;
variable c1, c2 : complex;
signal s1, s2 : complex;
. . .
c1 := c1 + c2;
s1 <= s2 + c1;
```


Encapsulation

- ADTs require strong encapsulation
- VHDL packages provide encapsulation
  - weak: concrete type is visible
  - strengthen by adding private types (à la Ada)
- Classes provide encapsulation
  - but replicate aspects of packages
  - complicate the language

Summary

- Both approaches can be made to work
- Prefer programming by extension
  - integrates with existing language better
    - existing features
    - existing style
  - avoids duplication of features
  - integrates with generics to provide mixin inheritance
- SUAVE
  - http://www.ececs.uc.edu/~petera/suave.html