

SUAVE: Painless Extension for an Object-Oriented VHDL

Peter J. Ashenden

University of Adelaide

Philip A. Wilsey, Dale E. Martin

University of Cincinnati

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

SUAVE

SAVANT and
University of
Adelaide
VHDL
Extensions

Outline

- Design Objectives
- Extension to the type system
 - type derivation & inheritance
 - type extension
 - abstract types
 - class-wide types
- Extensions for encapsulation
 - private types and private parts
- Integrating encapsulation and inheritance

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, type genericity
- Preserve capability for synthesis & other analysis
- Support hw/sw codesign
 - improved integration with programming languages
- . . .

Design Objectives (cont)

- Refinement through elaboration of components
 - avoid repartitioning
- Preserve correctness of existing models
- Design principles from VHDL-93
 - preserve “conceptual integrity”

Overview of Extensions

- Borrow heavily from Ada-95
 - VHDL already has much in common with Ada
 - borrow encapsulation, information hiding, inheritance, genericity features
- Class-based *cf* programming by extension
 - class-based
 - replicates package features
 - choose one or the other, but not both!
 - programming by extension
 - integrates better with signal semantics

Type Derivation and Classes

- Adopt from Ada-95:
 - tagged records
 - type derivation
 - type derived from tagged record can add elements
 - inherits primitive operations from parent type
 - can override/augment operations
 - class-wide types, class-wide operations
 - T'Class is hierarchy of types derived from T
 - dynamic dispatching
 - abstract type and operations
- Signals and dynamic variables can be class-wide

Type Derivation Example

```
type instruction is
  tagged record
    opcode : opcode_type;
  end record instruction;

function privileged ( instr : instruction; mode : protection_mode )
  return boolean;

procedure disassemble ( instr : instruction; file output : text );

type ALU_instruction is new instruction with
  record
    destination, source_1, source_2 : register_number;
  end record ALU_instruction;

procedure disassemble ( instr : ALU_instruction; file output : text );
```

Type Derivation Example (cont)

```
type memory_instruction is abstract new instruction with record
  base : register_number;
  offset : integer;
end record memory_instruction;

function effective_address_of ( instr : memory_instruction ) return natural;
procedure perform_memory_transfer ( instr : memory_instruction ) is abstract;

type load_instruction is new memory_instruction with record
  destination : reg_number;
end record load_instruction;

procedure perform_memory_transfer ( instr : load_instruction );

type store_instruction is new memory_instruction with record
  source : reg_number;
end record store_instruction;

procedure perform_memory_transfer ( instr : store_instruction );
```

Type Derivation Example (cont)

```
procedure execute ( instr : instruction'class ) is
begin
  disassemble ( instr, trace_file );
  if privileged(instr) and execution_mode = user then
    handle_privilege_violation;
  else
    ...
  end if;
end procedure execute;
```

```
entity instruction_reg is
  port ( load_enable : in bit;
        instr_in : in instruction'class;
        instr_out : out instruction'class );
end entity instruction_reg;
```

Encapsulation

- Strengthen existing package feature
 - used to define secure ADTs
- Package can have *visible part* and *private part*
- *Private type*
 - declare *partial view* in visible part
 - includes some contractual details
 - declare *full view* in private part
- Allow packages in any declarative region
 - local ADTs

Encapsulation Example

```
package complex_numbers is
  type complex is private;
  constant i : complex;
  function cartesian_complex ( re, im : real ) return complex;
  function re ( C : complex ) return real;
  function im ( C : complex ) return real;
  function "abs" ( C : complex ) return real;
  function arg ( C : complex ) return real;
  function "+" ( L, R : complex ) return complex;
  function "-" ( L, R : complex ) return complex;
  ...
private
  type complex is record
    re, im : real;
  end record complex;
end package complex_numbers;
```

Encapsulation Example (cont)

```
use complex_numbers.all;  
signal a, b, sum : complex := cartesian_complex(0.0, 0.0);  
signal enable : bit;  
...  
sum <= a + b after 10 ns when enable = '1' else  
    0.0 + 0.0*i after 10 ns;
```

Interaction: Encapsulation and Derivation

- Adopt mechanisms from Ada-95
 - tagged private type
 - can be extended without revealing details of parent
 - private extension
 - concrete details of extension hidden
- See paper for example

Conclusion

- SUAVE improves VHDL's support for modeling
 - across the spectrum
 - system-level down to gate level
 - improves encapsulation, inheritance, genericity
 - integrates cleanly with existing language
- Full details in papers and TRs
 - <http://www.ececs.uc.edu/~petera/suave.html>
- Implementation in progress