SUAVE: Extending VHDL to Improve Modeling Support

Peter J. Ashenden
University of Adelaide

Philip A. Wilsey
University of Cincinnati

June 1997

SUAVE

SAVANT and University of Adelaide
VHDL Extensions
Outline

- Design Objectives
- Overview of Extensions
  - encapsulation
  - type derivation & inheritance
  - genericity
- What we have not done (yet)
  - entity/architecture inheritance
  - concurrency and communication

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
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

```pascal
package complex_numbers is
  type complex is private;
  constant i : 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)

```vhdl
use complex_numbers.all;
signal a, b, sum : complex;
signal enable : bit;

sum <= a + b after 10 ns when enable = '1' else
  0.0 + 0.0*i after 10 ns;
```

---

Encapsulation: Contracts

- **Limited** private type
  - assignment not allowed, no predefined equality
  - use when deep copy/compare needed
- **Access** private type
  - needed if full view includes access types
  - can’t be used for signals
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

```ada
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;
```
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 papers for examples

Genericity

- Object-orientation is not a panacea
  - OO extension meets many, but not all, objectives
  - Doesn’t include type genericity needed for reuse
- VHDL has basic mechanism for genericity
  - generic constants
- Adopt generics from Ada-95
  - modified to integrate with VHDL generics
  - formal types, subprograms, packages
  - allow generic clause in subprograms and packages
- Instantiation done at elaboration-time
Formal Types in Entities

```vhdl
entity generic_multiplexer is
generic ( type data_type is private );
port ( control : in bit;  in0, in1 : in data_type;
       data_out : out data_type );
end entity generic_multiplexer;

architecture data_flow of generic_multiplexer is
begin
  with control select
    data_out <= in0 when '0',
                in1 when '1';
end architecture data_flow;

int_mux : entity work.generic_multiplexer(data_flow)
generic map ( data_type => integer );
port map ( . . . );
```

Formal Types in Packages

```vhdl
package sets is
  generic ( type element_type is private );
type set is access private;
constant empty_set;
procedure copy ( from : in set; to : out set );
function "+" ( R : element_type ) return set;
impure function "+" ( L : set;  R : element_type ) return set;
 . . .
private
  type element_node;
type element_ptr is access element_node;
type element_node is record
    next_element : element_ptr;
    value : element_type;
  end record element_node;
type set is new element_ptr;
end package sets;
```
Formal Types in Packages (cont)

```pascal
type test_vector is . . .
package test_sets is
    new sets
    generic map ( element_type => test_vector );
    use test_sets.all;
    variable tests_to_perform : test_sets.set := empty_set;
    . . .

test_to_perform := test_to_perform + new_test;
```

Interaction: Derivation and Genericity

- Formal derived type
  - provides mechanism for “mix-in” inheritance
  - obviates need for multiple inheritance in many cases
Derivation and Genericity Example

```vhdl
package indexed_addressing_mixin is
generic ( type parent_instruction is
  abstract new instruction with private );
type indexed_instruction is new instruction with record
  index_base, index_offset : register_number;
end record indexed_instruction;
function effective_address ( instr : indexed_instruction ) return address;
end package indexed_addressing_mixin;

type load_instruction is abstract new instruction with record
  destination : register_number;
end record load_instruction;

package indexed_loads is
  new indexed_addressing_mixin
  generic map ( parent_instruction => load_instruction );
alias indexed_load_instruction is indexed_loads.indexed_instruction;
```

Entity/Architecture Inheritance

- Other proposals
  - derived entities inherit ports/declarations
  - derived architectures inherit declarations/statements
- Not included in SUAVE
  - yet to be convinced that it’s worth it
  - compositional hierarchy more appropriate
  - counter-with-output-enable “contains-a” counter
  - counter-with-output-enable “contains-a” counter
Concurrency and Communication

- Other proposals
  - add procedural operations to entity interface
  - view entity as class of active objects
  - implies monitor-based semantics
    • concurrency-control approach
- Not included in SUAVE
  - true message-passing approach more appropriate
    • used by other system-level modeling languages
    • fits in better with VHDL concepts
    • future work

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