Extending functionality of UVM components by using Visitor design pattern

Darko M. Tomušilović
Vtool LTD
Introduction

• Goal - add a new operation to each class in existing class hierarchy
• Most common solutions and their drawbacks
  – adding code that will perform each operation into each class in the environment
  – creating derived classes that will perform newly added operations
• The alternative to these approaches has been well established in the software development world
• Briefly introduce Visitor design pattern
  – benefits (decreased code complexity, facilitated maintenance, improved code stability)
  – drawbacks (adding a new type of object that a visitor should visit is costly, its usage has to be planned in advance)
Visitor design pattern infrastructure

**uvm_visitor**

- Abstract class defining a general *visit* operation on a node
- Concrete visitor giving implementation to *visit* operation according to the action the visitor needs to accomplish
- Pre-processing and post-processing hooks

```plaintext
class name_display_visitor extends uvm_visitor;

    virtual function void visit(uvm_component node);
        `uvm_info("visitor",node.get_full_name(),UVM_LOW)
    endfunction

    function new (string name = "");
        super.new(name);
    endfunction
endclass
```
Visitor design pattern infrastructure

**uvm_adapter**

- Abstract class defining a general `accept` operation that in turn applies the corresponding visitor on every element of the structure that the adapter wraps.

- The following adapter wraps a single component:

```cpp
class basic_adapter extends uvm_visitor_adapter;
  virtual function void accept(uvm_component s, uvm_visitor v, uvm_structure_proxy#(uvm_component) p, bit invoke_begin_end=1);
    if(invoke_begin_end)
      v.begin_v();
    v.visit(s);
    if(invoke_begin_end)
      v.end_v();
  endfunction
  function new (string name = "");
    super.new(name);
  endfunction
endclass
```
Visitor design pattern infrastructure

Context

- Context invokes method `accept` of an object of adapter class, providing the component to be visited and also the visitor object as arguments.
UVM library predefined adapters

- Traverse elements in a complex composite structure in a specific way and apply visitor operation upon each of them
  - `uvm_top_down_visitor_adapter`
  - `uvm_bottom_up_visitor_adapter`
  - `uvm_by_level_visitor_adapter`

- Abstract `uvm_structure_proxy` class provides all children subelements of a certain element in a structure, facilitating traversal

- Its specialization class `uvm_components_proxy` provides all subcomponents for a given UVM component
Visitor traversal

```verilog
task visitor_env::run_phase(uvm_phase phase);
    name_display_visitor name_display_v;
    uvm_top_down_visitor adapter adapter;
    uvm_component_proxy proxy;

    name_display_v = new("name_display_v");
    adapter = new("adapter");
    proxy = new("proxy");

    adapter.accept(this, name_display_v, proxy);
endtask
```

```
UVM_INFO ../src/visitor_env.sv(7) @ 0: reporter [NAME DISPLAY VISITOR] uvm_test_top.env
UVM_INFO ../src/visitor_env.sv(7) @ 0: reporter [NAME DISPLAY VISITOR] uvm_test_top.env.master_agent
UVM_INFO ../src/visitor_env.sv(7) @ 0: reporter [NAME DISPLAY VISITOR] uvm_test_top.env.master_agent.driv
UVM_INFO ../src/visitor_env.sv(7) @ 0: reporter [NAME DISPLAY VISITOR] uvm_test_top.env.master_agent.driv.rsp_port
UVM_INFO ../src/visitor_env.sv(7) @ 0: reporter [NAME DISPLAY VISITOR] uvm_test_top.env.master_agent.driv.seq_item_port
UVM_INFO ../src/visitor_env.sv(7) @ 0: reporter [NAME DISPLAY VISITOR] uvm_test_top.env.master_agent.mon
UVM_INFO ../src/visitor_env.sv(7) @ 0: reporter [NAME DISPLAY VISITOR] uvm_test_top.env.master_agent.mon.analysis_port
UVM_INFO ../src/visitor_env.sv(7) @ 0: reporter [NAME DISPLAY VISITOR] uvm_test_top.env.master_agent.mon.peek_imp
UVM_INFO ../src/visitor_env.sv(7) @ 0: reporter [NAME DISPLAY VISITOR] uvm_test_top.env.master_agent.seqr

...
UML sequence diagram
Verification use-case examples

- Component configuration check visitor
  - check that every component in the environment is properly configured
- Reset and clock generation check visitor
- Adding messages and improving reporting system
Verification use-case examples
Component configuration check visitor

class component_check_visitor extends uvm_visitor;

virtual function void visit(uvm_component node);
    if (node.get_object_type() == visitor_master_driver::type_id::get()) begin
        visit_driver(node);
    end
endfunction

virtual function void visit_driver(uvm_component node);
    visitor_master_driver drv;
    $cast(drv, node);

    if (drv.visitor_if == null)
        `uvm_error("COMPONENT CHECK VISITOR", $sformatf("%s: Interface not set", drv.get_full_name()))
    else
        `uvm_info("COMPONENT CHECK VISITOR",$sformatf("%s: Interface set", drv.get_full_name()),UVM_LOW)
    if (drv.cfg == null)
        `uvm_error("COMPONENT CHECK VISITOR",$sformatf("%s: CFG not set", drv.get_full_name()))
    else
        `uvm_info("COMPONENT CHECK VISITOR",$sformatf("%s: CFG set", drv.get_full_name()),UVM_LOW)
endfunction

function new (string name = "");
    super.new(name);
endfunction
endclass
Verification use-case examples

- Component configuration check visitor
- Reset and clock generation check visitor
  - check that the components in the environment are provided with a proper clock and reset
- Adding messages and improving reporting system
Verification use-case examples

Reset check visitor

class reset_check_visitor extends uvm_visitor;

virtual function void visit(uvm_component node);
    if (node.get_object_type() == visitor_master_driver::type_id::get()) begin
        visit_driver(node);
    end
endfunction

virtual function void visit_driver(uvm_component node);
    visitor_master_driver drv;
    $cast(drv, node);

    if (drv.visitor_if.reset_n == 1'b1)
        `uvm_info("RESET CHECK VISITOR",$sformatf("%s: reset deasserted", drv.get_full_name()),UVM_LOW)
    else
        `uvm_info("RESET CHECK VISITOR",$sformatf("%s: reset asserted", drv.get_full_name()),UVM_LOW)
endfunction

function new (string name = "");
    super.new(name);
endfunction
endclass
Verification use-case examples

• Component configuration check visitor
• Reset and clock generation check visitor
• Adding messages and improving reporting system
  – attach a visitor to certain events in the environment and upon their triggering, perform proper reporting (for example, attach a visitor to a queue within a scoreboard)
Verification use-case examples
Queue display visitor

class queue_display_visitor extends uvm_visitor;

    virtual function void visit(uvm_component node);
        if (node.get_object_type() == visitor_sb::type_id::get()) begin
            fork
                visit_sb_tcb(node);
            join_none
            end
    endfunction

    virtual task visit_sb_tcb(uvm_component node);
        visitor_sb sb;
        $cast(sb, node);

        `uvm_info("QUEUE DISPLAY VISITOR", $sformatf("Start monitoring scoreboard queue"), UVM_LOW)
        forever begin
            @(sb.data_q.size());
            `uvm_info("QUEUE DISPLAY VISITOR", $sformatf("Queue size changed. New size: %d", sb.data_q.size()), UVM_LOW)
            `uvm_info("QUEUE DISPLAY VISITOR", $sformatf("Queue content: %p", sb.data_q), UVM_LOW)
        end
    endtask

    function new (string name = "");
        super.new(name);
    endfunction

class queue_display_visitor extends uvm_visitor;
Summary
Questions?

Thanks!