Towards modelling and Verification of Concurrent Ada programs using Petri Nets

A. Burns
A.J. Wellings
F.P. Burns
A.M. Koelmans
M. Koutny
A. Romanovski
A.V. Yakovlev

Atomic Actions

- Structuring of complex concurrent systems
- Several action participants
- Synchronous or asynchronous entry
- Synchronous exit
- No flow across border during action
- System layering
- Complexity hiding
Error Recovery

- Indivisible units of execution
- All-or-nothing semantics
- Tolerance of software faults
- Backward error recovery
  - Restore previous state
- Forward error recovery
  - Exception handling

ADA features

- Packages
- Types
- Asynchronous transfer of control
- Protected types
- Requeue
- Exceptions
- Controlled types
Problems with ADA

- Model is very complex
- Even small examples have many states
- Need confidence in correctness
- Use Petri Nets to model concurrency
- Analyse using PN tools
  - PEP
  - INA
  - Design/CPN

ADA example

package simple_action is
  procedure T1(params : param); -- from Task 1
  procedure T2(params : param); -- from Task 2
  procedure T3(params : param); -- from Task 3
end simple_action;
with Ada.Exceptions; use Ada.Exceptions;
package body action is

    type Vote_T is (Commit, Aborted);
    protected controller is
        entry Wait_Abort(E: out Exception_Id);
        entry Done;
        entry Cleanup (Vote : Vote_t; Result : out Vote_t);
        procedure Signal_Abort(E: Exception_Id);
    private
        entry Wait_Cleanup(Vote : Vote_t; Result : out Vote_t);
        Killed : boolean := False;
        Releasing_cleanup : Boolean := False;
        Releasing_Done : Boolean := False;
        Reason : Exception_Id;
        Final_Result : Vote_t := Commit;
        informed : integer := 0;
    end controller;

protected body controller is
    entry Wait_Abort(E: out Exception_id) when killed is
        begin
            E := Reason;
            informed := informed + 1;
            if informed = 3 then
                Killed := False;
                informed := 0;
            end if;
        end Wait_Abort;
entry Done when Done'Count = 3 or Releasing_Done is 
begin 
    if Done'Count > 0 then
        Releasing_Done := True;
    else
        Releasing_Done := False;
    end if;
end done;

entry Cleanup (Vote: Vote_t;
  Result: out Vote_t) when True is 
begin 
    if Vote = aborted then
        Final_result := aborted;
    end if;
    requeue Wait_Cleanup with abort;
end Cleanup;

procedure Signal_Abort(E: Exception_id) is 
begin 
    killed := TRue;
    reason := E;
end Signal_Abort;

entry Wait_Cleanup (Vote : Vote_t; Result: out Vote_t) 
    when Wait_Cleanup'Count = 3 or Releasing_Cleanup is
begin 
    Result := Final_Result;
    if Wait_Cleanup'Count > 0 then
        Releasing_Cleanup := True;
    else
        Releasing_Cleanup := False;
        Final_Result := Commit;
    end if;
end Wait_Cleanup;
end controller;
procedure T1(params: param) is
    X : Exception_ID;
    Decision : Vote_t;
begin
    select
        Controller.Wait_Abort(X);
        raise_exception(X);
    then abort
        begin
            -- code to implement atomic action
            Controller.Done; --signal completion
        exception
            when E: others =>
                Controller.Signal_Abort (Exception_Identity(E));
        end;
    end select;

    exception
        -- if any exception is raised during the action
        -- all tasks must participate in the recovery
        when E: others =>
            -- Exception_Identity(E) has been raised in all tasks
            -- handle exception
            if handled_ok then
                Controller.Cleanup(Commit, Decision);
            else
                Controller.Cleanup(Aborted, Decision);
            end if;
            if decision = aborted then
                raise atomic_action_failure;
            end if;
    end T1;

    -- similar for T2 and T3
end action;
State diagram

Task Model
Controller model

Verification with PEP

success1,fail2
-- results in <NO>

Success1,success2
-- results in <YES>

Fail1,fail2
-- results in <YES>
Verification with INA

Safe - No
Bounded - Yes
Dead State Reachable - No
Covered by Transition-Invariants - Yes
Resettable, reversible (to home state) - Yes
Dead transitions exist - No
Live - Yes
Live and Safe - No

Verification with Design/CPN
Design/CPN: Statistics

<table>
<thead>
<tr>
<th>Occurrence Graph</th>
<th>Scc Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes: 63</td>
<td>Nodes: 13</td>
</tr>
<tr>
<td>Arcs: 114</td>
<td>Arcs: 14</td>
</tr>
<tr>
<td>Secs: 0</td>
<td>Secs: 0</td>
</tr>
<tr>
<td>Status: Full</td>
<td></td>
</tr>
</tbody>
</table>

Design/CPN: Boundedness

<table>
<thead>
<tr>
<th>Best Integers Bounds</th>
<th>Upper</th>
<th>Lower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control'Fail_n 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Control'Killed 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Control'LocDone_n 1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
## Design/CPN: liveness

<table>
<thead>
<tr>
<th>Dead Markings: None</th>
<th>Control'AbortAll 1</th>
<th>Fair</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live Transitions Instances:</td>
<td>Control'CommitAll 1</td>
<td>Fair</td>
</tr>
<tr>
<td></td>
<td>Control'DoneAll 1</td>
<td>Fair</td>
</tr>
<tr>
<td>Control'AbortAll 1</td>
<td>Control'Restart_nf 1</td>
<td>Fair</td>
</tr>
<tr>
<td>Control'CommitAll 1</td>
<td>Control'Restart_ns 1</td>
<td>Fair</td>
</tr>
<tr>
<td>Control'Restart_nf 1</td>
<td>Control'SigAbort 1</td>
<td>Fair</td>
</tr>
<tr>
<td>Control'Restart_ns 1</td>
<td>Control'Sync 1</td>
<td>Fair</td>
</tr>
<tr>
<td>Control'SigAbort 1</td>
<td>Tasks'Arr_n 1</td>
<td>Impartial</td>
</tr>
<tr>
<td>Control'Sync 1</td>
<td>Tasks'Done_n 1</td>
<td>Just</td>
</tr>
</tbody>
</table>

## Conclusions

- Modelling and verification possible
- Improve confidence in correctness
- Hand translation only
- Need automated extraction
- No real-time issues yet