Preservation of Timing Properties with the Ada Ravenscar Profile

Enrico Mezzetti, Marco Panunzio, Tullio Vardanega

Department of Pure and Applied Mathematics
University of Padova, Italy
{emezzett, panunzio, tullio.vardanega}@math.unipd.it

15th International Conference on Reliable Software Technologies – Ada-Europe 2010

Valencia, Spain, June 15th, 2010

Outline

- Property preservation
- The Ravenscar Profile
- Ada 2005 monitoring constructs
- Property enforcement
- Property monitoring
- Fault handling
- Conclusion

Property preservation

- **A significant value fraction of new-generation systems arises from non-functional properties**
  - Verified at design time
- **Values assumed for static analysis should become constraints on system behavior**
  - Else the value of analysis is denied
  - Must be conveyed to implementation and preserved at run time
- **Key ingredients**
  1) Analysis framework
     - To statically analyze the system
  2) Programming model
     - To enforce analysis assumptions
     - To solely express the semantics assumed by the analysis
  3) Run-time enforcement of properties

Preservation of timing properties

- **Realized in three steps**
  - Enforcement of controllable properties
    - Period [periodic tasks]
    - Minimum inter-arrival time (MIAT) [sporadic tasks]
    - Phase
  - Monitoring of properties determined by system execution
    - Worst-case execution time (WCET)
    - Deadline
    - Worst-case blocking time (WCBT)
  - Treatment of detected violations

The Ravenscar Profile

- **Reduced tasking model**
  - Reject language constructs exposed to
    - Non-determinism
    - Unbounded execution time
  - Ravenscar systems are amenable to static analysis
    - In the time dimension
- **RP and property preservation**
  1) Analysis framework ✚
     - Based on (e.g.) Response Time Analysis
  2) Programming model ✚
     - Strict subset of Ada 2005
       - Enforced through *pragma Profile* / *pragma Restrictions*
       - Programs with forbidden constructs rejected by the compiler
  3) Run-time enforcement of properties ✖
     - Language mechanisms are insufficient

Monitoring of execution time

- **Worst-case execution time is one fundamental input to schedulability analysis**
  - Safe and tight bound needed
  - Achieving both qualities is difficult
    - Assumed bounds may prove unsafe in unanticipated scenarios
    - We need mechanisms to promptly detect violations (i.e., overruns)

$$R_i^{n+1} = B_i + \boxed{C_i} + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil \boxed{C_j}$$

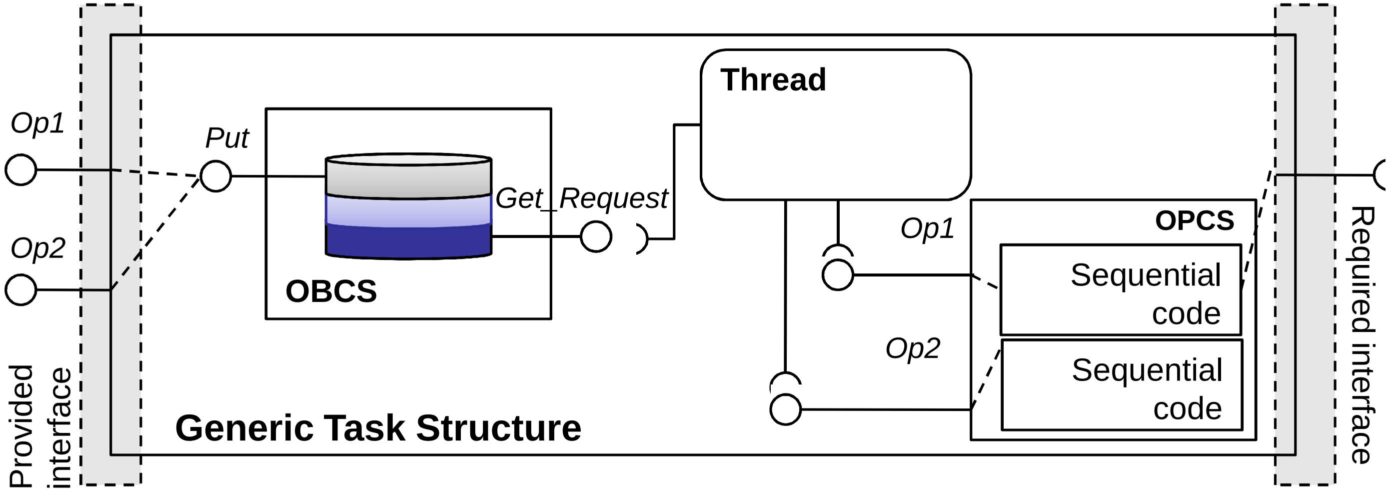
- **Efficient, practical and standard means to measure execution time are important to industrial practice**
  - Best achieved with language-level constructs
- **Ada 2005 provides execution-time timers**
  - Monitor the CPU time consumed by a single task
  - Handler (protected procedure) raised on timer expiration
  - IRTAW-14 proposed to include it in the Ravenscar Profile

Ada 2005 monitoring constructs

Language constructs	Timing properties			Within the Ravenscar Profile
	Period or MIAT	Deadline	WCET	
delay until	*	N/A	N/A	yes
Timer	N/A	N/A	●	proposed for inclusion
Timing_Event	N/A	●	N/A	yes (*library level*)

* enforcement of the property
● monitoring of property and notification of violation

Task template



- **4 blocks which address distinct concerns**
  - Provided and Required Interface
  - OPCS (Operation Control Structure)
    - Sequential behaviour of each service
  - Thread
    - Task behaviour – executes the OPCS services as required
  - OBCS (Object Control Structure)
    - Synchronization agent – management of release events for the task
    - Reifies calls to the PI in a request descriptor, later fetched by the thread
    - Data-oriented asynchronous communication as per the Ravenscar profile

Add enforcement/ monitoring constructs to the **Thread**

Enforcement of period/MIAT

```
-- structure for a periodic / sporadic task with optional Phase
Next_Time := System_Activation_Time + Phase;
loop
    delay until Next_Time;
    <fetch a request descriptor from the OBCS and decode it>
    <invoke the required service in the OPCS>
    Next_Time := Next_Time + Milliseconds(Interval);
end loop;
```

- **Use of *delay until***
  - Absolute-time suspension as opposed to relative-time suspension
    - No drift in the period
    - Precision dependent on the hardware

Monitoring of deadlines

```
-- deadline-monitored loop for a periodic / sporadic task
loop
   Set_Handler (Deadline_Event, Next_Deadline, Deadline_Miss_Handler);
   <task operations>
   Next_Time := Next_Time + Milliseconds (Interval);
   Next_Deadline:= Next_Time + Milliseconds (Rel_Deadline);
   Cancel_Handler (Deadline_Event, isSetHandler);
   delay until Next_Time;
end loop;
```

- **Use of `Timing_Event`**
  - Sets the timer to expire at the absolute time of the next deadline
  - `Deadline_Miss_Handler` is triggered upon timer expiration
  - The alarm must be canceled at the end of each task activation
- **Inconclusive to determine the cause of the fault**
  - The task which misses the deadline or a higher-priority task

Monitoring of worst-case execution time

```
-- WCET-monitored loop for a periodic / sporadic task
loop
  Set_Handler (WCET_Timer, WCET_bound, WCET_Overrun_Handler);
  <task operations>
  Next_Time := Next_Time + Milliseconds (Interval);
  delay until Next_Time;
end loop;
```

- **Use of execution-time timers**
  - Sets the timer to expire when the task executes for more than *WCET_bound* CPU time
  - `WCET_Overrun_Handler` triggered upon timer expiration
- **Precise indication of the faulty task**

Monitoring of worst-case blocking time

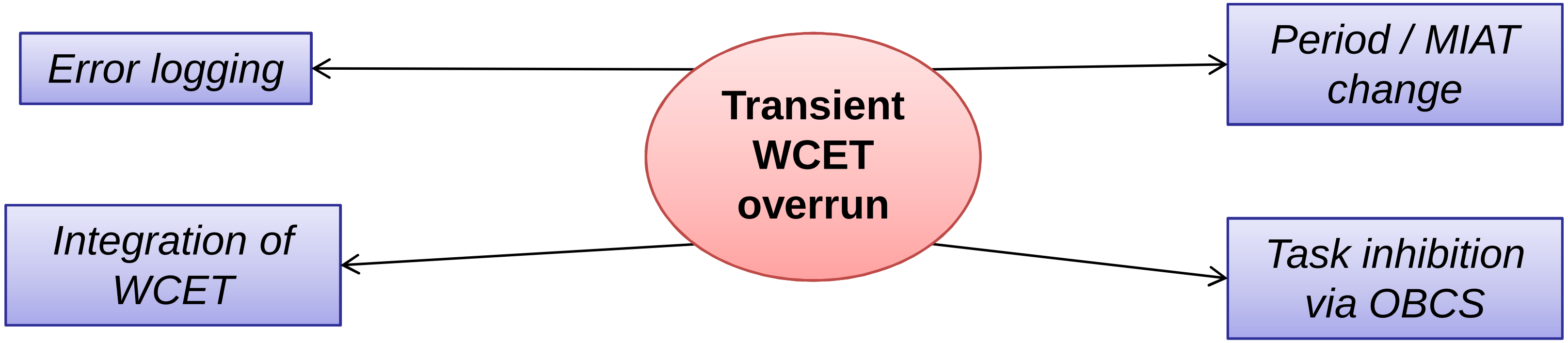
- **WCBT bounded by the resource access protocol**
  - `Ceiling_Locking` policy (ICP) under the Ravenscar Profile
- **Overrun in WCBT may cause subtle timing faults**
  - An overrun in a critical section may cause a deadline miss in *higher-priority* tasks (with *p* < ceiling(CS))
  - Unrelated to WCET overrun
  - Solution: direct monitoring of blocking time
    - Requires specialized kernel support
  - `Group_Budgets` and `Execution_Time.Timers` not useful

$$R_i^{n+1} = \boxed{B_i} + C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j$$

```
-- WCBT-monitored critical section
Time_In := Execution_Time.Clock;
<beginning of critical section CS>
<end of critical section CS>
Time_Out := Execution_Time.Clock;
if (Time_Out − Time_In > CS_WCET) then
  <overrun handling>
end if;
```

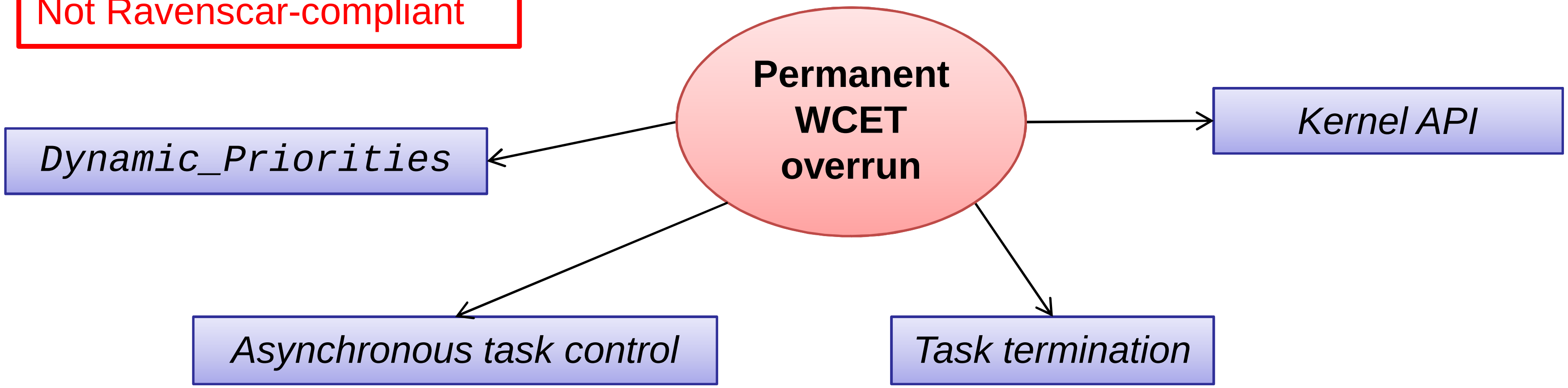
- **Use of the execution-time `Timers` unsatisfactory**
  - **WCBT overrun detected after exiting from critical section**
  - **Overrun handled at task priority level** (occurs when the blocked task has already missed its deadline)

Fault handling



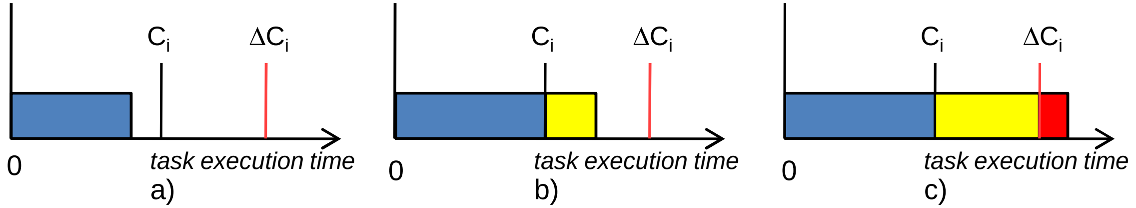
Can be realized with Ravenscar

Not Ravenscar-compliant



Integration of WCET

- **Sensitivity analysis can calculate the largest WCET overrun which does not impair overall system schedulability**
  - Can be used to safely increase the WCET bound for a task in case of transient overruns
    - Requires recalculation of all ΔC_x



- **Scheduling analysis as formulated for "weakly hard real-time systems"**
  - The task set meets "any *n* in *m* deadlines"
  - The task set meets "any row *n* in *m* deadlines"
  - Possible to calculate the ΔC_x even under the above requirements
- **Both require extension to account for shared resources**
  - For the calculation of task ΔC_x and ΔC_{CSi}

Handling of permanent overruns

- **`Dynamic_Priorities`**
  - Only to decrease base priority
  - Preserves task schedulability
  - Not satisfactory for data integrity
  - Large time and space overhead
- **Kernel API**
  - To flag a task as non-executable
    - Immediate task dispatching point
  - Reversible flag
  - Little time and space overhead
- **Asynchronous Task Control**
  - Unable to cope with task stuck in critical section

- How to deal with overruns inside shared resources?
- Who should use those mechanisms?
- How fast are we able to react to a fault detection?
- What is the maximum latency of the fault handling mechanism?

Architectural issues

Recap

Techniques	Transient WCET overrun	Permanent WCET overrun	Ravenscar compliance
Error logging	*	○	yes
Integration of WCET	●	○	yes
Period/MIAT change	●	○	yes
Inhibition via OBCS	●	○	yes
Task termination	○	●	no
Dynamic priorities	○	●	no
Asynchronous Task Control	○	●	no
Kernel API	●	●	no

* = applicable to the temporal fault
● = possible to remedy to the temporal fault
○ = unable or inappropriate to cope with such temporal fault

Conclusion

- **Property preservation is essential**
  - To assure that the system at run time corresponds with the analysis stipulations
- **The Ravenscar Profile (with execution-time timers) offers good property preservation value**
  - Enforces controllable properties
  - Monitors timing properties at run-time
  - Reacts to timing faults
  - Still some areas with no satisfactory solution yet
    - Monitoring of *blocking time*
    - Permanent overruns
      - However we may consider those situations as exceptional in high-integrity real-time systems