

4.a Programming real-time systems (in Ada)

Preserving properties at run time

Workload model /1

- Static set of tasks
 - **Ada**: tasks declared at *library level* (i.e. the outermost scope), so that they have the longest lifetime
- Tasks issue jobs repeatedly
 - Task duty cycle: activation, execution, suspension
 - *Single* activation source per task
- Real-time attributes
 - Release time
 - Periodic: at every T time units
 - Sporadic: at least T between any two subsequent releases
 - Execution
 - Worst case execution time (WCET) assumed to be known
 - Deadline: D time units after release

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 216 of 538

Preserving properties at run time

Workload model /2

- Task communication
 - Shared variables with mutually exclusive access
 - **Ada**: *protected objects* with procedures and functions
 - No conditional synchronization
 - Other than for sporadic task activation
 - **Ada**: PO with a *single entry* ← For optimal determinism
- Scheduling model
 - Fixed-priority pre-emptive
 - **Ada**: FIFO within priorities
- Access protocol for shared objects
 - Ceiling priority protocol
 - **Ada**: Ceiling_Locking policy

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 217 of 538

Preserving properties at run time

Protected objects /1

```
protected type Shared_Integer (Initial_Value : Integer) is
  function Read return Integer;
  procedure Write (Value : Integer);
private
  The_Integer : Integer := Initial_Value;
end Shared_Integer;
```

Concurrent Read ———→

Mutually-exclusive Write ———→

```
protected body Shared_Integer is
  function Read return Integer is
  begin
    return The_Integer;
  end Read;
  procedure Write (Value : Integer) is
  begin
    The_Integer := Value;
  end Write;
end Shared_Integer;
```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 218 of 538

Preserving properties at run time

Protected objects /2

```

Buffer_Size : constant Positive := 5;
type Index is mod Buffer_Size; -- tipo modulare
subtype Count is Natural range 0 .. Buffer_Size;
type Buffer_T is array (Index) of Any_Type;

protected type Bounded_Buffer is
  entry Get (Item : out Any_Type);
  entry Put (Item : in Any_Type);
private
  First : Index := Index'First; -- 0
  Last : Index := Index'Last; -- 4
  In_Buffer : Count := 0;
  Buffer : Buffer_T;
end Bounded_Buffer;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 219 of 538

Preserving properties at run time

Protected objects /3

```

protected body Bounded_Buffer is
  entry Get (Item : out Any_Type)
  when In_Buffer > 0 is
  begin -- first read then move pointer
    Item := Buffer(First);
    First := First + 1; -- free from overflow
    In_Buffer := In_Buffer - 1;
  end Get;
  entry Put (Item : in Any_Type)
  when In_Buffer < Buffer_Size is
  begin -- first move pointer then write
    Last := Last + 1; -- free from overflow
    Buffer(Last) := Item;
    In_Buffer := In_Buffer + 1;
  end Put;
end Bounded_Buffer;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 220 of 538

Preserving properties at run time

Protected objects /4

```

Buffer_Size : constant Positive := 5;
type Index is mod Buffer_Size; -- tipo modulare
subtype Count is Natural range 0 .. Buffer_Size;
type Buffer_T is array (Index) of Any_Type;

protected type Bounded_Buffer is
  entry Get (Item : out Any_Type);
  entry Put (Item : in Any_Type);
private
  First : Index := Index'First; -- 0
  Last : Index := Index'Last; -- 4
  In_Buffer : Count := 0;
  Buffer : Buffer_T;
end Bounded_Buffer;

protected body Bounded_Buffer is
  entry Get (Item : out Any_Type)
  when In_Buffer > 0 is
  begin -- first read then move pointer
    Item := Buffer(First);
    First := First + 1; -- free from overflow
    In_Buffer := In_Buffer - 1;
  end Get;
  entry Put (Item : in Any_Type)
  when In_Buffer < Buffer_Size is
  begin -- first move pointer then write
    Last := Last + 1; -- free from overflow
    Buffer(Last) := Item;
    In_Buffer := In_Buffer + 1;
  end Put;
end Bounded_Buffer;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 221 of 538

Preserving properties at run time

Language profiling

- Enforced by means of a *configuration pragma*
pragma Profile (Ravenscar);
- Equivalent to a set of Ada restrictions, plus three additional configuration pragmas
pragma Task_Dispatching_Policy (FIFO_With_Priorities);
pragma Locking_Policy (Ceiling_Locking);
pragma Detect_Blocking;
- ISO/IEC TR 24718, *Guide for the use of the Ada Ravenscar Profile in High Integrity Systems*
 See «Per approfondire 8»

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 222 of 538

Preserving properties at run time

Ravenscar restrictions

```

No_Abort_Statements,
No_Dynamic_Attachment,
No_Dynamic_Priorities,
No_Implicit_Heap_Allocations,
No_Local_Protected_Objects,
No_Local_Timing_Events,
No_Protected_Type_Allocators,
No_Relative_Delay,
No_Requeue_Statements,
No_Select_Statements,
No_Specific_Termination_Handlers,
No_Task_Allocators,
No_Task_Hierarchy,
No_Task_Termination,
Simple_Barriers,
Max_Entry_Queue_Length => 1,
Max_Protected_Entries => 1,
Max_Task_Entries => 0,
No_Dependence => Ada.Asynchronous_Task_Control,
No_Dependence => Ada.Calendar,
No_Dependence => Ada.Execution_Time.Group_Budget,
No_Dependence => Ada.Execution_Time.Timers,
No_Dependence => Ada.Task_Attributes

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 223 of 538

Preserving properties at run time

Restriction checking

- Almost all of the Ravenscar profile restrictions are checked at compile time
- A few of them can be checked *only* at run time
 - Potentially blocking operations in PO bodies
 - Priority ceiling violation
 - More than one call queued on a protected entry or a suspension object
 - Task termination

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 224 of 538

Preserving properties at run time

Potentially blocking operations

- Protected entry call statement
- Delay until statement
- Call on a subprogram whose body contains a potentially blocking operation
- **Pragma Detect_Blocking** requires detection of potentially blocking operations
 - Exception **Program_Error** raised if detected
 - Signifying a grave violation of good conduct
 - Blocking need not be detected if it occurs in the domain of a call to a foreign language (e.g. into C embedded in an Ada procedure)

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 225 of 538

Preserving properties at run time

Other run-time checks

- Priority ceiling violation
- More than one call waiting on a protected entry or a suspension object
 - **Program_Error** must be raised in both cases
 - Signaling violation of analysis assumptions
- Task termination
 - Program behavior must be documented
 - Possible termination behaviors include
 - Silent termination
 - Holding the task in a pre-terminated state
 - Call of an application-defined termination handler defined with the **Ada.Task_Termination** package (C.7.3)

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 226 of 538

Preserving properties at run time

Other restrictions

- Some restrictions on the sequential part of the language may be useful in conjunction with the Ravenscar profile
 - **No_Dispatch**
 - **No_IO**
 - **No_Recursion**
 - **No_Unchecked_Access**
 - **No_Allocators**
 - **No_Local_Allocators**
- For details, see: ISO/IEC TR 15382, *Guide for the use of the Ada Programming Language in High Integrity Systems*

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 227 of 538

Preserving properties at run time

Outside of Ravenscar

- Real-time programming facilities of use when full static assurance is *not* possible
 - Execution-time measurement
 - Execution-time timers
 - Group budgets (for sporadic servers and other resource reservation policies)
 - Timing events
 - Additional dispatching policies

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 228 of 538

Preserving properties at run time

Execution-time measurement

- The CPU time consumed by tasks can be monitored
- *Per-task* CPU clocks can be defined
 - Set at $t = 0$ before task activation
 - The clock value increases (notionally) as the task executes
 - Actual increments occur solely at dispatching points (sound) or at synchronous queries (silly)

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 229 of 538

Preserving properties at run time

Ada.Execution_Time

```

with Ada.Task_Indefinition;
with Ada.Real_Time; use Ada.Real_Time;
package Ada.Execution_Time is
  type CPU_Time is private;
  CPU_Time_First : constant CPU_Time;
  CPU_Time_Last  : constant CPU_Time;
  CPU_Time_Unit  : constant := implementation-defined-real-number;
  CPU_Tick      : constant Time_Span;
  function Clock
    (T : Ada.Task_Indefinition.Task_Id
     := Ada.Task_Indefinition.Current_Task)
    return CPU_Time;
  ...
end Ada.Execution_Time;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 230 of 538

Preserving properties at run time

Execution-time timers

- A user-defined event can fire when a CPU clock reaches a specified value
 - An *event handler* is automatically invoked by the runtime at that point
 - The handler is an (**access to**) a **protected procedure**
 - The educated equivalent of a callback
- Basic mechanism for execution-time monitoring

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 231 of 538

Preserving properties at run time

Ada.Execution_Time.Timers /1

```
with System;
package Ada.Execution_Time.Timers is
  type Timer (T : not null access constant
              Ada.Task_Identification.Task_Id) is
    tagged limited private;
  type Timer_Handler is
    access protected procedure (TM : in out Timer);
  Min_Handler_Ceiling : constant System.Any_Priority
    := implementation_defined;
  procedure Set_Handler (TM      : in out Timer;
                        In_Time : in Time_Span;
                        Handler  : in Timer_Handler);
  procedure Set_Handler (TM      : in out Timer;
                        At_Time  : in CPU_Time;
                        Handler  : in Timer_Handler);
  ...
end Ada.Execution_Time.Timers;
```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 232 of 538

Preserving properties at run time

Ada.Execution_Time.Timers /2

- Builds on execution-time clocks
- Needs an *interval timer*
 - To update at every dispatching point
 - To raise «zero events» that signify, for example, *budget overruns*
- Handling sensibly those zero events requires other sophisticated features

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 233 of 538

Preserving properties at run time

Group budgets

- Groups of tasks with a global execution-time budget can be defined
 - Basic mechanism for server-based scheduling
 - As needed to serve aperiodic arrivals
 - Can be used to provide temporal isolation among groups of tasks

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 234 of 538

Preserving properties at run time

Group budgets (spec)

```
with System;
package Ada.Execution_Time.Group_Budgets is
  type Group_Budget is tagged limited private;
  type Group_Budget_Handler is
    access protected procedure (GB : in out Group_Budget);
  ...
  Min_Handler_Ceiling : constant System.Any_Priority
    := implementation_defined;
  procedure Add_Task (GB : in out Group_Budget;
    T : in Ada.Task_Identification.Task_Id);
  ...
  procedure Replenish (GB : in out Group_Budget;
    To : in Time_Span);
  procedure Add (GB : in out Group_Budget;
    Interval : in Time_Span);
  ...
  procedure Set_Handler (GB : in out Group_Budget;
    Handler : in Group_Budget_Handler);
  ...
end Ada.Execution_Time.Group_Budgets;
```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 235 of 538

Preserving properties at run time

Timing events

- *Lightweight* mechanism for defining code to be executed at a specified time
 - Does *not* require an application-level task
 - Very similar to interrupt handling
- The code is defined as an event handler
 - An (**access to**) a **protected procedure**
- Directly invoked by the runtime

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 236 of 538

Preserving properties at run time

Ada.Real_Time.Timing events

```
package Ada.Real_Time.Timing_Events is
  type Timing_Event is tagged limited private;
  type Timing_Event_Handler is
    access protected procedure (Event : in out Timing_Event);
  procedure Set_Handler (Event : in out Timing_Event;
    At_Time : in Time;
    Handler : in Timing_Event_Handler);
  ...
  procedure Cancel_Handler (Event : in out Timing_Event;
    Cancelled : out Boolean);
  ...
end Ada.Real_Time.Timing_Events;
```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 237 of 538

Preserving properties at run time

Dispatching policies

- Additional dispatching policies
 - **Non preemptive** (explicit yield)
 - Run-to-completion semantics (per partition)
 - **Round robin**
 - Within specified priority band
 - Dispatch on quantum expiry deferred until end of protected action
 - **Earliest Deadline First**
 - Within specified priority band
 - Relative and absolute “deadline”
 - EDF ordered ready queues
 - Guaranteed form of resource locking (preemption level + deadline)

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 238 of 538

Preserving properties at run time

Priority-band dispatching

- Mixed policies can coexist within a single partition
 - Priority specific dispatching policy can be set by configuration
 - Protected objects can be used for tasks to communicate across different policies
 - Tasks do not move across bands

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 239 of 538

Preserving properties at run time

OOD for real-time systems

- Real-time components are objects
 - Instances of predefined classes
 - Internal state + interfaces
- Based on well-defined code patterns
 - Cyclic & sporadic tasks
 - Protected data
 - Passive data

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 240 of 538

Preserving properties at run time

Enforce intentions

- Static WCET analysis and response-time analysis can be used to assert correct temporal behavior at *design time*
- Platform mechanisms can be used at *run time* to ensure that temporal behavior stays within the asserted boundaries
 - Clocks, timers, timing events, ...
- Conveniently complementary approaches

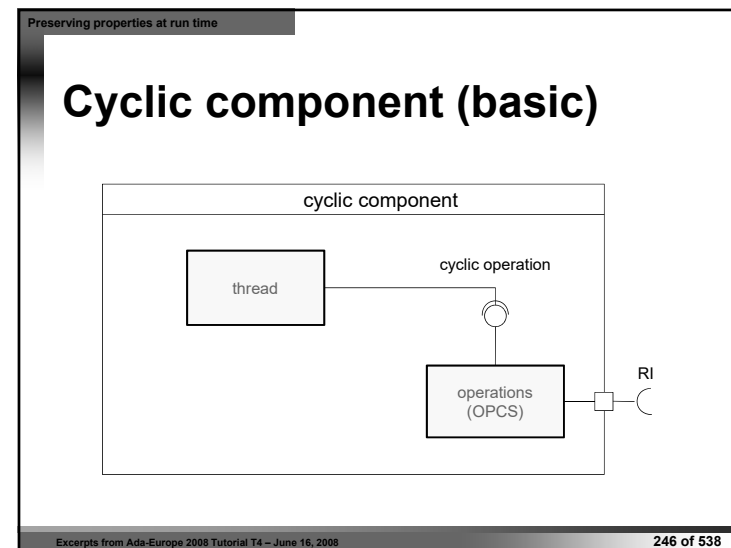
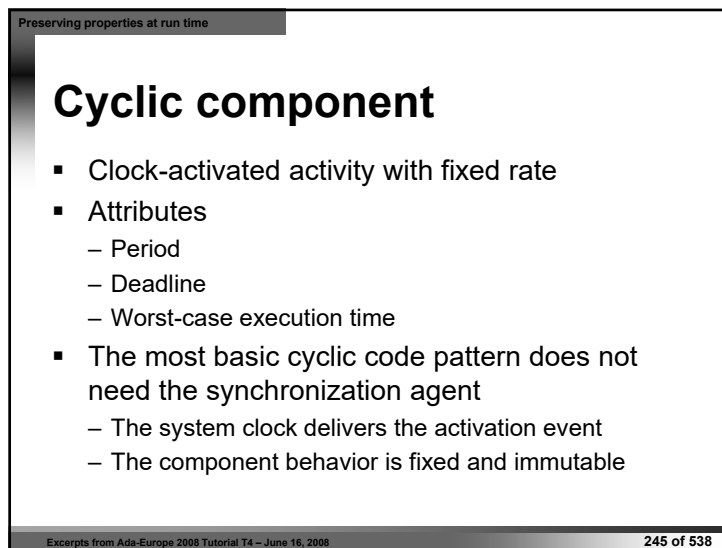
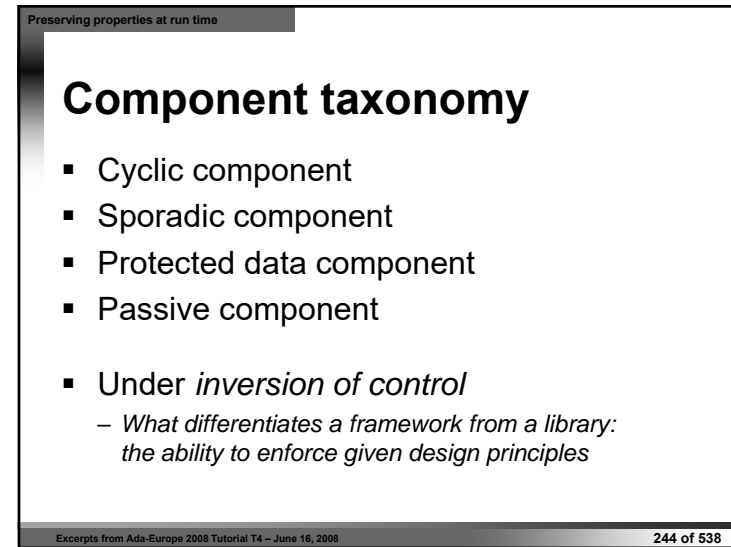
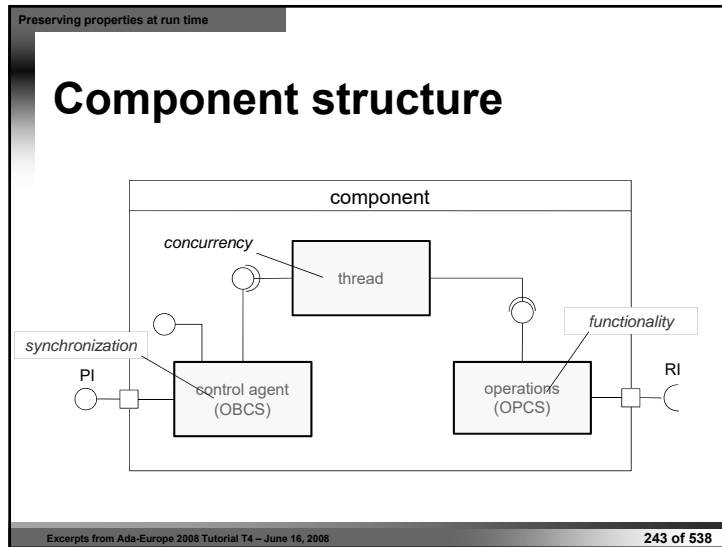
Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 241 of 538

Preserving properties at run time

Run-time services

- The execution environment must be capable of preserving properties asserted at model level
 - Real-time clocks & timers
 - Execution-time clocks & timers
 - Predictable scheduling
- We assume an execution environment implementing the Ravenscar model
 - Ada 2012 with the Ravenscar profile
 - Augmented with (restricted) execution-time timers

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 242 of 538



Preserving properties at run time

Cyclic thread (spec)

```

task type Cyclic_Thread
  (Thread_Priority : Priority;
   Period         : Positive) is
  pragma Priority(Thread_Priority);
end Cyclic_Thread;

```

ms

cannot be Time_Span!

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 247 of 538

Preserving properties at run time

Cyclic thread (body)

```

task body Cyclic_Thread is
  Next_Time : Time := <Start_Time>; -- taken at elaboration time
                                           --+ higher in the system
                                           --+ hierarchy

begin
  loop
    delay until Next_Time; -- so that all tasks start at T0
    OPCS.Cyclic_Operation; -- fixed and parameterless
    Next_Time := Next_Time + Milliseconds(Period);
  end loop;
end Cyclic_Thread;

```

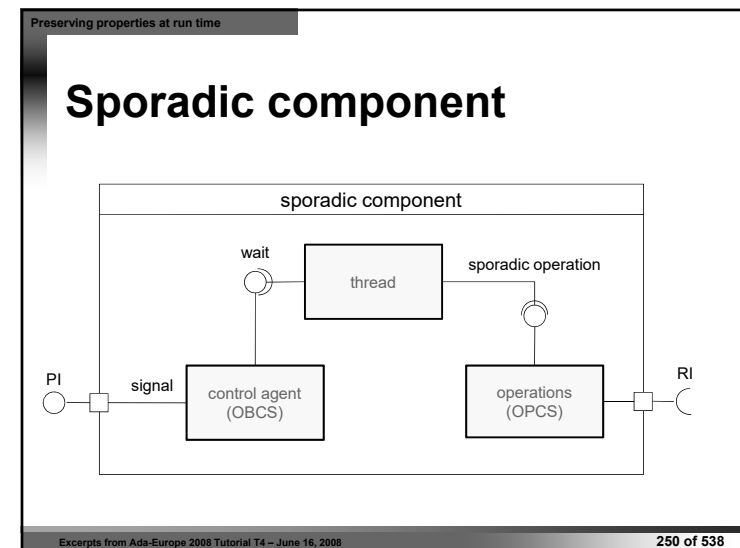
Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 248 of 538

Preserving properties at run time

Sporadic component

- Activated by a software-mediated event
 - Signaled by software or hardware interrupts
- Attributes
 - Minimum inter-arrival time
 - Deadline
 - Worst-case execution time
- The synchronization agent of the target component is used to signal the activation event
 - And to store-and-forward signal-related data (if any)

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 249 of 538



Preserving properties at run time

Sporadic component (spec)

```

task type Sporadic_Thread(Thread_Priority : Priority) is
  pragma Priority(Thread_Priority);
end Sporadic_Thread;

protected type OBCS(Ceiling : Priority) is
  pragma Priority(Ceiling);
  procedure Signal;
  entry Wait;
private
  Occurred : Boolean := False;
end OBCS;

```

A sporadic thread is activated by calling the Signal operation

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 251 of 538

Preserving properties at run time

Sporadic thread (body)

```

task body Sporadic_Thread is
  Next_Time : Time := <Start_Time>;
begin
  delay until Next_Time; -- so that all tasks start at T0
  loop
    OBCS.Wait;
    OPCS.Sporadic_Operation;
    -- may take parameters if they were delivered by Signal
    --+ and retrieved by Wait
  end loop;
end Sporadic_Thread;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 252 of 538

Preserving properties at run time

Sporadic control agent (body)

```

protected body OBCS is
  procedure Signal is
  begin
    Occurred := True;
  end Signal;
  entry Wait when Occurred is
  begin
    Occurred := False;
  end Wait;
end OBCS;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 253 of 538

Preserving properties at run time

Other components

- **Protected component**
 - No thread, only synchronization and operations
 - Straightforward direct implementation with protected object
- **Passive component**
 - Purely functional behavior, neither thread nor synchronization
 - Straightforward direct implementation with functional package

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 254 of 538

Preserving properties at run time

Temporal properties

- Basic patterns only guarantee periodic or sporadic activation
- They can be augmented to guarantee additional temporal properties at run time
 - Minimum inter-arrival time for sporadic events
 - Deadline for all types of thread
 - WCET budgets for all types of thread

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 255 of 538

Preserving properties at run time

Minimum inter-arrival time /1

- Violations of the specified separation interval may cause increased interference on lower priority tasks
- Approach: prevent sporadic thread from being activated earlier than stipulated
 - Compute earliest (absolute) allowable activation time
 - Withhold activation (if triggered) until that time

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 256 of 538

Preserving properties at run time

Sporadic thread with minimum separation (spec)

```

task type Sporadic_Thread
  (Thread_Priority : Priority;
   Separation      : Positive) is
  pragma Priority(Thread_Priority);
end Sporadic_Thread;
  
```

(ms)

Minimum inter-arrival time
expressed in ms

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 257 of 538

Preserving properties at run time

Sporadic thread (body)

```

task body Sporadic_Thread is
  Release_Time : Time;
  Next_Release : Time := <Start_Time>;
begin
  loop
    delay until Next_Release;
    OBCS.Wait;
    Release_Time := Clock;
    OPCS.Sporadic_Operation;
    Next_Release := Release_Time + Milliseconds(Separation);
  end loop;
end Sporadic_Thread;
  
```

Still a single point of activation

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 258 of 538

Preserving properties at run time

Critique

- May incur some temporal drift as the clock is read *after* task release
 - Preemption may hit just after the release but before reading the clock
 - Separation may become *larger* than required
- Better to read the clock at the place and time the task is released
 - Within the synchronization agent
 - Which is protected and thus less exposed to general interference

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 259 of 538

Preserving properties at run time

Minimum inter-arrival time /2

```

task body Sporadic_Thread is
  Release_Time : Time;
  Next_Release : Time := <Start_Time>;
begin
  loop
    delay until Next_Release;
    OBCS.Wait(Release_Time);
    OPCS.Sporadic_Operation;
    Next_Release := Release_Time + MilliSeconds(Separation);
  end loop;
end Sporadic_Thread;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 260 of 538

Preserving properties at run time

Recording release time /1

```

protected type OBCS(Ceiling : Priority) is
  pragma Priority(Ceiling);
  procedure Signal;
  entry Wait(Release_Time : out Time);
private
  Occurred : Boolean := False;
end OBCS;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 261 of 538

Preserving properties at run time

Recording release time /2

```

protected body OBCS is
  procedure Signal is
  begin
    Occurred := True;
  end Signal;

  entry Wait(Release_Time : out Time) when Occurred is
  begin
    Release_Time := Clock;
    Occurred := False;
  end Wait;
end OBCS;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 262 of 538

Preserving properties at run time

Deadline miss

- May result from
 - Higher priority tasks executing more often than expected
 - Can be prevented with inter-arrival time enforcement
 - Overruns in the same or higher priority tasks
 - Programming error in the functional code
 - Inaccurate WCET analysis

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 263 of 538

Preserving properties at run time

Deadline miss detection

- Can be done with the help of **timing events**
 - A mechanism for requiring some application-level action to be executed at a given time
 - Under the Ravenscar Profile timing events can only exist at library level
- Timing events are statically allocated
- Minor optimization possible for periodic tasks
 - Which however breaks the symmetry of code patterns

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 264 of 538

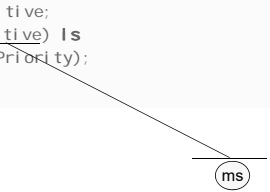
Preserving properties at run time

Cyclic thread with deadline miss detection (spec)

```

task type Cyclic_Thread
  (Thread_Priority : Priority;
   Period         : Positive;
   Deadline       : Positive) is
  pragma Priority(Thread_Priority);
end Cyclic_Thread;

```



Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 265 of 538

Preserving properties at run time

Thread body

```

Deadline_Ovrrun : Timing_Event; -- static, local per component
task body Cyclic_Thread is
  Next_Time : Time := <Start_Time>;
  Cancelled : Boolean := False;
begin
  loop
    delay until Next_Time;
    Set_Handler(Deadline_Ovrrun,
               Next_Time + Milliseconds(Deadline),
               Deadline_Ovrrun_Handler); -- application-specific
    OPCS.Cyclic_Operation;
    Cancel_Handler(Deadline_Ovrrun, Cancelled);
    Next_Time := Next_Time + Milliseconds(Period);
  end loop;
end Cyclic_Thread;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 266 of 538

Preserving properties at run time

Thread body (streamlined)

```

Deadline_Ovrrun : Timing_Event; -- static, local per component
task body Cyclic_Thread Is
  Next_Time : Time := <Start_Time>;
  Canceled : Boolean := False;
begin
  loop
    -- setting again cancels any previous event
    Set_Handler(Deadline_Ovrrun,
      Next_Time + Milli seconds(Deadline),
      Deadline_Ovrrun_Handler); -- application-specific
    delay until Next_Time;
    OPCS.Cyclic_Operation;
    Next_Time := Next_Time + Milli seconds(Period);
  end loop;
end Cyclic_Thread;

```

Watch out!
What about the critical instant?

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 267 of 538

Preserving properties at run time

Sporadic thread with deadline miss detection (spec)

```

task type Sporadic_Thread
  (Thread_Priority : Priority;
   Separation : Positive;
   Deadline : Positive) Is
  pragma Priority(Thread_Priority);
end Sporadic_Thread;

```

ms

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 268 of 538

Preserving properties at run time

Thread body

```

Deadline_Ovrrun : Timing_Event; -- static, local per component
task body Sporadic_Thread Is
  Release_Time : Time;
  Next_Release : Time := <Start_Time>;
  Canceled : Boolean := False;
begin
  loop
    delay until Next_Release;
    OPCS.Wait(Release_Time);
    Set_Handler(Deadline_Ovrrun,
      Release_Time + Milli seconds(Deadline),
      Deadline_Ovrrun_Handler); -- application-specific
    OPCS.Sporadic_Operation;
    Cancel_Handler(Deadline_Ovrrun, Canceled);
    Next_Release := Release_Time + Milli seconds(Separation);
  end loop;
end Sporadic_Thread;

```

Cannot streamline as the deadline cannot be computed until returning from Wait

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 269 of 538

Preserving properties at run time

Execution-time overruns


- Tasks may execute for longer than stipulated owing to
 - Programming errors in the functional code
 - Inaccurate WCET values used in feasibility analysis
 - Optimistic vs. pessimistic
- WCET overruns can be detected at run time with the help of **execution-time timers**
 - Not included in Ravenscar
 - Extended profile

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 270 of 538

Preserving properties at run time

Cyclic thread with WCET overrun detection (spec)

```
task type Cyclic_Thread
  (Thread_Priority : Priority;
   Period         : Positive;
   WCET_Budget    : Positive) is
  pragma Priority(Thread_Priority);
end Cyclic_Thread;
```



Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 271 of 538

Preserving properties at run time

Thread body

```
task body Cyclic_Thread is
  Next_Time : Time := <Start_Time>;
  Id : aliased constant Task_ID := Current_Task;
  WCET_Timer : Timer(Id'access);
begin
  loop
    delay until Next_Time;
    Set_Handler(WCET_Timer,
               MilliSeconds(WCET_Budget),
               WCET_Overrun_Handler); -- application-specific
    OPCS.Cyclic_Operation;
    Next_Time := Next_Time + MilliSeconds(Period);
  end loop;
end Cyclic_Thread;
```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 272 of 538

Preserving properties at run time

Observation

- WCET overruns in sporadic tasks can be detected similarly
- The timer should be set after the activation
- No need for timer cancellation

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 273 of 538

Preserving properties at run time

Multiple job types per task

- To support mode changes, threaded objects may have **modifier** PI operations
 - So that tasks issue other jobs than default
- Asynchronous Transfer of Control (ARM § 9.7.4) are not allowed in Ravenscar
 - Hence mode change must be synchronous
 - Modifier requests are queued in the OBCS

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 274 of 538

Preserving properties at run time

Cyclic thread with modifier

```

task body Cyclic_Thread Is
  Next_Release_Time : Time := <Start_Time>;
  Request : Request_Type;
begin
  loop
    delay until Next_Release_Time;
    OBCS.Get_Request(Request); -- may include operation parameters
    case Request is
      when NO_REQ => OPCS.Periodic_Activity;
      when ATC_REQ => -- may take parameters
                     OPCS.Modifier_Operation;
    end case;
    Next_Release_Time := Next_Release_Time + Period;
  end loop;
end Cyclic_Thread;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 275 of 538

Preserving properties at run time

Synchronization agent /1

```

-- for cyclic thread
protected type OBCS (Ceiling: Priority) is
  pragma Priority(Ceiling);
  procedure Put_Request(Request : Request_Type);
  procedure Get_Request(out Request : Request_Type);
private
  Buffer : Request_Buffer; -- bounded queue
end OBCS;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 276 of 538

Preserving properties at run time

Synchronization agent /2

```

-- for cyclic thread
protected body OBCS(Ceiling : Priority) is
  procedure Put_Request(Request : Request_Type) is
  begin
    Buffer.Put(Request);
  end Put_Request;
  procedure Get_Request(out Request : Request_Type) is
  begin
    if Buffer.Empty then
      Request := NO_REQ;
    else
      Buffer.Get(Request);
    end if;
  end Get_Request;
end OBCS;

```

Excerpts from Ada-Europe 2008 Tutorial T4 – June 16, 2008 277 of 538