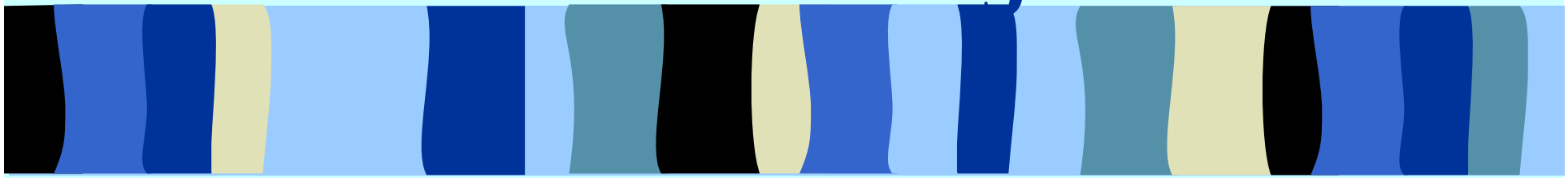
Ada Tasking: From the Ravenscar Profile to Dynamic Scheduling



Alan Burns
Department of Computer Science
University of York, UK

Tasking

- Ada provides a powerful concurrency model
- No other language has such a wide range of synchronous and asynchronous features

- In this talk we shall look at the extremes

Ravenscar

- Covered in other talks, but
- Simple, predictable, efficient, analyzable

- Static number of ‘flat’ tasks
- PO for mutual exclusion
- PO with single entry (and no queue) for condition synchronisation

- No rendezvous, select, abort, delay, calendar, dynamic priorities etc

- Simple but safe

Dynamic Scheduling

- Need to support a more flexible computational model
- A task consists of a mandatory **job** and an optional **job**

- Mandatory jobs must complete by the deadline
- Optional jobs have **value** and should complete by the deadline

- In an overload, value is used to determine which jobs to run and which to abandon

- Better to never start than to abort

Scheduling Results

- EDF (Earliest Deadline First) is optimal if no overload
- EDF is poor if overload
- Value density (V/C) is optimal if deadlines are ignored
- Basic Approach
  - Some form of admission control to prevent overload
  - EDF scheduling of all jobs
  - Mandatory jobs jump to a higher priority (than EDF queue) to ensure they meet their deadlines

Task Attributes

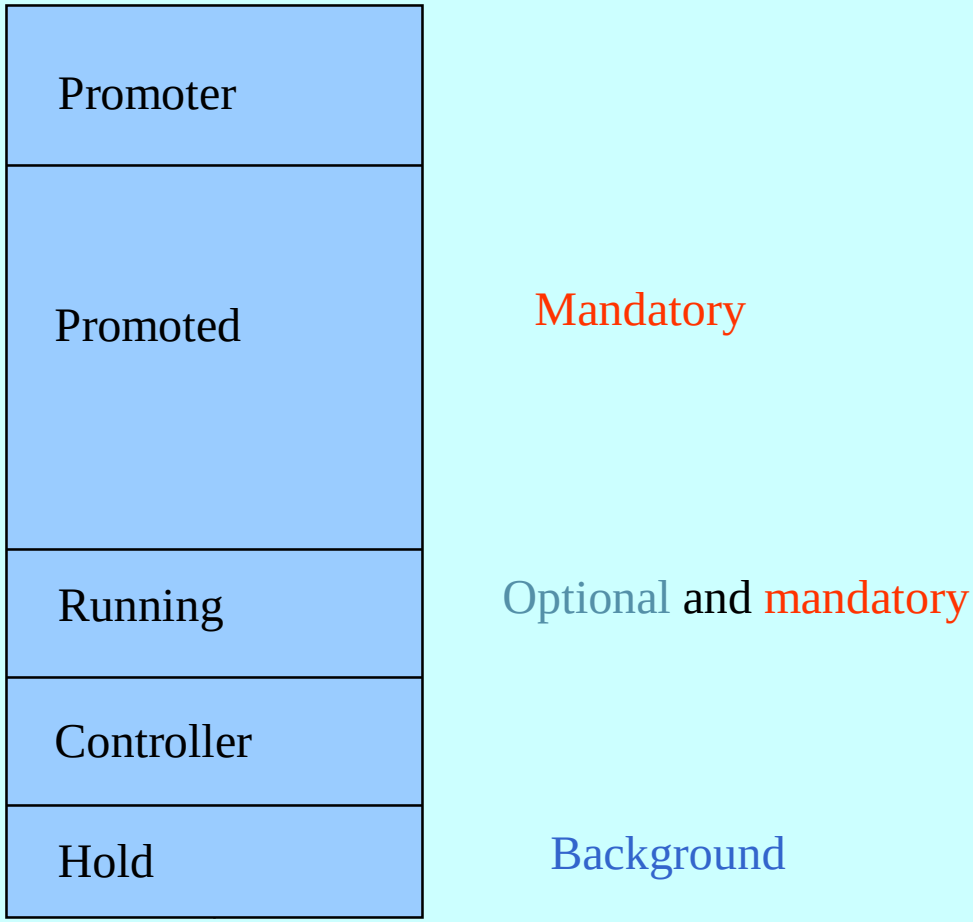
- Period (or arrival interval for sporadic activities)
- Deadline
- Priority promotion time - for mandatory work
- Computation time - mandatory
- Computation time - optional (perhaps minimum …)
- Priority for mandatory if priority must jump
- Value of optional job

Components

- Client tasks
- Promoter tasks
- Scheduling PO
- EDF queue manager (all jobs)
- Value queue manager (all optional jobs)
- Controller task - runs when new job required

```
package Program_Priorities is
  Hold : constant System.Priority :=
                          System.Priority'first + 1;
    -- priority of all task with jobs on EDF queue
  Controller_Pri : constant System.Priority := Hold + 1;
  Running : constant System.Priority := Hold + 2;
    -- priority of running task,
    -- i.e. job just removed from EDF queue
  Pro_Pri : constant System.Priority :=
                          System.Priority'last;
    -- promoter task's priority
  subtype Promoted_Priorities is System.Priority range
     Hold + 3 .. System.Priority'last - 1;
    -- priority of tasks executing mandatory
    -- jobs (after promotion)
end Program_Priorities;
```

Priority Map



Parameters of Model

```
package Specific_Parameters is
  type value is range 0 .. 10; -- say
  type load is new float range 0.0 .. 10.0; -- say
    -- load is a measure of current EDF work load
  threshold : Load := 0.8; -- say
end Specific_Parameters;
```

Queue Managers

```
package EDF is
  procedure Add(ID : Task_ID);
  procedure Remove(ID : Task_ID; Found : out boolean);
  procedure Extract(ID : out Task_ID; L : out Load);
end EDF;


package Value_Manager is
  procedure Add(ID : Task_ID);
  procedure Remove(ID : Task_ID);
  procedure Cut(ID : Task_ID; LD : Load;
                             Admit : out boolean);
end Value_Manager;
```

Task Attributes

```
package Task_Info is
  type Task_Information is
  record
    Deadline : Time := Time_Last;
    Val : Value := Value'first;
    Comp_Min, Comp_Req : Time_Span := Time_Span_Zero;
    Man_Fin : boolean := false;
  end record;

  Default : Task_Information;
  -- default object is needed in
  -- definition of task attributed

  package Scheduling_Parameter is new
    Ada.Task_Attributes(Task_Information, Default);
end Task_Info;
```

Promoter Task

```
task type Promoter is
  entry set_times(Start_T : Time; HP : Priority;
                  Period,R_Deadline,R_Pro : Time_Span);
  pragma priority(Pro_Pri);
end Promoter;

type Pro is access Promoter;

task body Promoter is
  My_Period : Time_Span;
  Pro_Time : Time_Span;
  Rel_Deadline : Time_Span;
  TP : Task_Information;
  High : Priority;
  Client : Task_ID;
  Epoch : Time;
begin
```

```
  accept Set_Times(Start_T : Time; HP : Priority;
                   Period,R_Deadline,R_Pro : Time_Span) do
    My_Period := Period;
    Epoch := Start_T;
    High := HP;
    Pro_Time := R_Pro;
    Rel_Deadline := R_Deadline;
    Client := Set_Times'Caller;
  end Set_Times;
  loop
    delay until Epoch + Pro_Time;
    TP := Scheduling_Parameter.Value(Client);
    if not TP.Man_Fin then
      set_priority(High,Client);
    end if;
    delay until Epoch + Rel_Deadline;
    set_priority(High,Client);
    Epoch := Epoch + My_period;
  end loop;
end Promoter;
```

```
task type Example_Client(My_Priority:
                         Promoted_Priorities) is
  pragma Priority(My_Priority);
end Example_Client;

task body Example_Client is
  Start : Time;
  Period : Time_Span := ...
  Rel_Deadline : Time_Span := ...
  Abs_Deadline : Time;
  Rel_Promotion_Time : Time_Span := ...
  Mandatory_Finished : boolean;
  Self : Task_ID;
  Quality : Value;
  Timing_Error : exception;
  Minimum_Comp_Man : Time_Span := ...
  Required_Comp_Man : Time_Span := ...
  Minimum_Comp_Opp : Time_Span := ...
  Required_Comp_Opp : Time_Span := ...
  P : Pro := new Promoter;
begin
```

```
begin
   Start := clock;
   Self := Current_Task;
   P.set_times(Start, My_Priority, Period, Rel_Deadline,
               Rel_Promotion_Time);
   loop
      Mandatory_Finished := false;
      Abs_Deadline := Start + Rel_Deadline;
      select
         delay until Abs_Deadline;
         if not Mandatory_Finished then
            raise Timing_Error;
         end if;
      then abort
         -- stuff
      end select;
      Start := Start + Period;
      delay until Start;
   end loop;
exception when ...
end Example_Client;
```

```
then abort
  Quality := Value'last;
  scheduling_parameter.set_value((Abs_Deadline,
    Quality, Minimum_Comp_Man,
    Required_Comp_Man, Mandatory_Finished));
  Sch.Register_Mandatory(Self);
  -- Mandatory Job
  Mandatory_Finished := true;
  Sch.Remove_Mandatory(Self);
  Quality := ... Not Value'last
  scheduling_parameter.set_value((Abs_Deadline,
     Quality, Minimum_Comp_Opp, Required_Comp_Opp,
     Mandatory_Finished));
  Sch.Register_Optional(Self);
  -- Optional Job
end select;
```

Scheduler PO

```
protected Sch is
   entry Release_New;
   procedure Register_Mandatory(Cl : Task_ID);
   procedure Remove_Mandatory(Cl : Task_ID);
   procedure Register_Optional(Cl : Task_ID);
   pragma Priority(Pro_Pri);
private
   Num_on_EDF : natural := 0;
end Sch;
```

Controller task

```
task Controller is
  pragma Priority(Controller_Pri);
end Controller;

task body Controller is
begin
  loop
    Sch.Release_New;
  end loop;
end Controller;
```

```
protected body Sch is

  procedure Register_Mandatory(Cl : Task_ID) is
  begin
    EDF.Add(Cl);
    Num_on_EDF := Num_on_EDF + 1;
    Set_Priority(Hold,Cl);
  end;

  procedure Remove_Mandatory(Cl : Task_ID) is
    Found : boolean;
  begin
    EDF.Remove(Cl,Found);
    if Found then
      Num_on_EDF := Num_on_EDF - 1;
    end if;
  end;
```

```
procedure Register_Optional(Cl : Task_ID) is
begin
   EDF.Add(Cl);
   Num_on_EDF := Num_on_EDF + 1;
   Value_Manager.Add(Cl);
   Set_Priority(Hold,Cl);
end;
```

```
entry Release_New when Num_on_EDF > 0 is
  ID : Task_ID;
  LD : Load;
  TP : Task_Information;
  Run_it : boolean;
begin
  EDF.extract(ID,LD);
  Num_on_EDF := Num_on_EDF - 1;
  TP := Scheduling_Parameter.Value(ID);
  if TP.Val = Value'last then -- mandatory part
    Set_Priority(Running,ID);
    return;
  end if;
  if clock + TP.Comp_Min > TP.Deadline then
    Value_Manager.Remove(ID); -- not worth starting
    requeue Release_New;
  end if;
  ...
```

```
    if LD <= Threshold then  -- no overload
      Value_Manager.Remove(ID);
      Set_Priority(Running,ID);
      return;
    end if;
    Value_Manager.Cut(ID,LD,Run_it); -- overload
    if Run_it then
      Set_Priority(Running,ID);
    else
      requeue Release_New; -- find another task
    end if;
  end Release_New;
```

Features Used

- Tasks and Protected Objects
- Rendezvous
- Dynamic Priorities
- Task Attributes
- Select-then-Abort
- Requeue

Feature Missing from Ada

- Computation time management

```
select
   delay 7.0; -- execution time
then abort
   code
end select;
```

Conclusions

- Ada has nearly all the features needed to program flexible schedulers
- The tasking model has high expressive power
- It is a vehicle by which scheduling results can be moved into industrial practice
- Lack of control over execution time is a drawback - to be rectified in the next version of Ada?