Ada Issue 00354 Group Execution-Time Budgets

!standard D.14.2 (01) 05-03-24 AI95-00354/08
!class amendment 03-09-27
!status Amendment 200Y 04-06-25
!status ARG Approved 7-0-1 04-06-13
!status work item 03-09-27
!status received 03-09-27
!priority Low
!difficulty Medium
!subject Group execution-time budgets

!summary

This AI proposes a child package of Ada.Execution_Time to allow more than one task to share an execution-time budget.

!problem

Currently Ada has no mechanisms to allow the implementation of aperiodic servers such as sporadic servers and deferrable servers. This severely limits the language's ability to handle aperiodic activities at anything other than a background priority. The fundamental problem that prohibits the implementation of a periodic server algorithms is that tasks cannot share CPU budgets.

!proposal

(See wording.)

!wording

Add new section:

D.14.2 Group Execution Time Budgets

This clause describes a language-defined package to assign execution time budgets to groups of tasks.

Static Semantics

The following language-defined library package exists:

```
with System;
package Ada.Execution_Time.Group_Budgets is

  type Group_Budget is limited private;

  type Group_Budget_Handler is access
       protected procedure (GB : in out Group_Budget);
```

```
  type Task_Array is array (Positive range <>) of
                          Ada.Task_Identification.Task_Id;

  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 Remove_Task (GB : in out Group_Budget;
                         T  : in Ada.Task_Identification.Task_Id);
  function Is_Member (GB : Group_Budget;
                      T  : Ada.Task_Identification.Task_Id) return Boolean;
  function Is_A_Group_Member (
     T : Ada.Task_Identification.Task_Id) return Boolean;
  function Members (GB : Group_Budget) return Task_Array;

  procedure Replenish (GB : in out Group_Budget; To : in Time_Span);
  procedure Add (GB : in out Group_Budget; Interval : in Time_Span);
  function Budget_Has_Expired (GB : Group_Budget) return Boolean;
  function Budget_Remaining (GB : Group_Budget) return Time_Span;

  procedure Set_Handler (GB      : in out Group_Budget;
                         Handler : in Group_Budget_Handler);
  function Current_Handler (GB : Group_Budget) return Group_Budget_Handler;
  procedure Cancel_Handler (GB        : in out Group_Budget;
                            Cancelled : out Boolean);

  Group_Budget_Error : exception;
private
   --  not specified by the language
end Ada.Execution_Time.Group_Budgets;
```

The type Group_Budget represents an execution time budget to be used by a group of tasks. The type Group_Budget needs finalization (see 7.6). A task can belong to at most one group. Tasks of any priority can be added to a group.

An object of type Group_Budget has an associated non-negative value of type Time_Span known as its *budget*, which is initially Time_Span_Zero.
The type Group_Budget_Handler identifies a protected procedure to be executed by the implementation when the budget is *exhausted*, that is, reaches zero. Such a protected procedure is called a *handler*.

An object of type Group_Budget also includes a handler, which is a value of type Group_Budget_Handler. The handler of the object is said to be *set* if it is not null and *cleared* otherwise. The handler of all Group_Budget objects is initially cleared.

Dynamic Semantics

The procedure Add_Task adds the task identified by T to the group GB; if that task is already a member of some other group, Group_Budget_Error is raised.

The procedure Remove_Task removes the task identified by T from the group GB; if that task is not a member of the group GB, Group_Budget_Error is raised.
After successful execution of this procedure, the task is no longer a member of any group.

The function Is_Member returns True if the task identified by T is a member of the group GB; otherwise it return False.

The function Is_A_Group_Member returns True if the task identified by T is a member of some group; otherwise it returns False.

The function Members returns an array of values of type Task_Identification.Task_Id identifying the members of the group GB. The order of the components of the array is unspecified.

The procedure Replenish loads the group budget GB with the Time_Span value To. Group_Budget_Error is raised if the Time_Span value To is non-positive. Any execution of any member of the group of tasks results in the budget counting down. When the budget becomes exhausted (reaches Time_Span_Zero), the associated handler is executed if the handler of group budget GB is set; the tasks continue to execute.

The procedure Add modifies the budget of the group GB. A positive value for Interval increases the budget. A negative value for Interval reduces the budget, but never below Time_Span_Zero. A zero value for Interval has no effect. A call of procedure Add that results in the value of the budget going to Time_Span_Zero causes the associated handler to be executed if the handler of the group budget GB is set.

The function Budget_Has_Expired returns True if the budget of group GB is exhausted (equal to Time_Span_Zero); otherwise it returns False.

The function Budget_Remaining returns the remaining budget for the group GB. If the budget is exhausted it returns Time_Span_Zero. This is the minimum value for a budget.

The procedure Set_Handler associates the handler Handler with the Group_Budget GB; if Handler is null, the handler of Group_Budget is cleared, otherwise it is set.

A call of Set_Handler for a Group_Budget that already has a handler set replaces the handler; if Handler is not null, the handler for Group_Budget remains set.

The function Current_Handler returns the handler associated with the group budget GB if the handler for that group budget is set; otherwise it returns null.

The procedure Cancel_Handler clears the handler for the group budget if it is set. Cancelled is assigned True if the handler for the group budget was set prior to it being cleared; otherwise it is assigned False.

The constant Min_Handler_Ceiling is the priority value that ensures that no ceiling violation would occur, were a handler to be executed.

The precision of the accounting of task execution time to a Group_Budget is the same as that defined for execution-time clocks from the parent package.

As part of the finalization of an object of type Group_Budget all member tasks are removed from the group identified by that object.

If a task is a member of a Group_Budget when it terminates then as part of the finalization of the task it is removed from the group.

For all the operations defined in this package, Tasking_Error is raised if the task identified by T has terminated, and Program_Error is raised if the value of T is Task_Identification.Null_Task_Id.

An exception propagated from a handler invoked when the budget of a group of tasks becomes exhausted has no effect.

Erroneous Execution

For a call of any of the subprograms defined in this package, if the task identified by T no longer exists, the execution of the program is erroneous.

Implementation Requirements

For a given Group_Budget object, the implementation shall perform the operations declared in this package atomically with respect to any of these operations on the same Group_Budget object. The replacement of a handler, by a call of Set_Handler, shall be performed atomically with respect to the execution of the handler.

AARM note:
This prevents various race conditions. In particular it ensures that if the budget is exhausted when Set_Handler is changing the handler then either the new or old handler is executed and the exhausting event is not lost.
End AARM note

Notes

Clearing or setting of the handler of a group budget does not change the current value of the budget. Exhaustion or loading of a budget does not change whether the handler of the group budget is set or cleared.

A Group_Budget_Handler can be associated with several Group_Budget objects.

!example

One common bandwidth preserving technique is the deferrable server.
The code for a simple deferrable server is given below:

```
with Ada.Timing_Events; use Ada.Timing_Events;
with Ada.Execution_Time.Group_Budgets;
use Ada.Execution_Time.Group_Budgets;
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Real_Time; use Ada.Real_Time;
with System; use System;
package Deferrable_Servers is
```

```
  type Deferrable_Server is limited private;

  procedure Create(DS : in out Deferrable_Server; First : Time;
                   Budget : Time_Span; Period : Time_Span);
  procedure Add(DS : in out Deferrable_Server; T : Task_Id);

private
  protected type Controller(DS : access Deferrable_Server) is
    procedure Budget_Has_Expired(GB: in out Group_Budget);
    procedure Replenish_Due(TE : in out Timing_Event);
    pragma Priority(Priority'Last);
  end Controller;

  type Deferrable_Server is record
    Budget : Time_Span;
    Period : Time_Span;
    Next_Replenishment_Time : Time;
    Replenish_Control : Timing_Event;
    Budget_Control : Group_Budget;
    Server_Control : Controller(Deferrable_Server'Access);
  end record;
end Deferrable_Servers;
```

A deferrable server can be represented by a type which encapsulated the Budget, the replenishment period, the next replenishment time, a Timing_Event to signal the next replenishment time, a Group_Budget to monitor the execution time consumed by the controlled tasks, and a controller to perform the required suspension and resumption of the tasks.

The body of the package is given below.

```
with Ada.Asynchronous_Task_Control; use Ada.Asynchronous_Task_Control;
package body Deferrable_Servers is

  procedure Create(DS : in out Deferrable_Server; First : Time;
                   Budget : Time_Span; Period : Time_Span) is
  begin
    DS.Budget := Budget;
    DS.Period := Period;
    DS.Next_Replenishment_Time := First;
    Group_Budgets.Set_Handler(DS.Budget_Control,
                              DS.Server_Control.Budget_Has_Expired'Access);
    Timing_Events.Set_Handler(DS.Replenish_Control,
                              DS.Next_Replenishment_Time,
                              DS.Server_Control.Replenish_Due'Access);
  end Create;

  procedure Add(DS : in out Deferrable_Server; T : Task_Id) is
  begin
    Add_Task(DS.Budget_Control,T);
  end Add;
```

```
  protected body Controller is
    procedure Budget_Has_Expired(GB : in out Group_Budget) is
      TA : Task_Array := Members(GB);
    begin
      for ID in TA'Range loop
        Ada.Asynchronous_Control.Hold(TA(ID));
      end loop;
    end Budget_Has_Expired;

    procedure Replenish_Due(TE : in out Timing_Event) is
      TA : Task_Array;
    begin
      Replenish(DS.Budget_Control, DS.Budget);
      DS.Next_Replenishment_Time := DS.Next_Replenishment_Time + DS.Period;
      Timing_Events.Set_Handler(DS.Replenish_Control,
                                DS.Next_Replenishment_Time,
                                DS.Server_Control.Replenish_Due'Access);
      TA := Members(DS.Budget_Control);
      for ID in TA'Range loop
        Ada.Asynchronous_Control.Continue(TA(ID));
      end loop;
    end Replenish_Due;

  end Controller;

end Deferrable_Servers;
```

!discussion

Various alternative models were considered including:

a) Passing the protected procedure as an access discriminant to the Group_Budget type.

This was rejected in favor of explicit get and set methods mainly for ease of use when combining an object of the Group_Budget type and the required protected object into a single record type.

b) Passing an unconstrained array of task identifiers as a parameter to the Handler protected procedure.

The argument for such a facility is that the user of the package is probably going to want to know the group of tasks whose Timer has expired. This can now be done with the Members function.

c) Having the Group_Budget type as a tagged type.

This was rejected by the IRTAW as unclear on whether the benefit was worth the added complexity and overhead.

d) Having the package automatically suspend the group of tasks when the associated Group_Budget expired.

This was rejected because not all Aperiodic Server approaches suspend the tasks, some set the tasks' priorities to a background priority.

e) Making the handler a non null access type. This was eventually rejected in favor of unifying the approach used with that in AI-297 on single timers.

!corrigendum D.14.2(01)

@dinsc

This clause describes a language-defined package to assign execution time budgets to groups of tasks.

@i<@s8<Static Semantics>>

The following language-defined library package exists:

@xcode<@b<with> System;
@b<package> Ada.Execution_Time.Group_Budgets @b<is>

  @b<type> Group_Budget @b<is limited private>;

  @b<type> Group_Budget_Handler @b<is access>
      @b<protected procedure> (GB : @b<in out> Group_Budget);

  @b<type> Task_Array @b<is array> (Positive @b<range> <@>) @b<of>
                            Ada.Task_Identification.Task_Id;

  Min_Handler_Ceiling : @b<constant> System.Any_Priority :=
    @ft<@i<implementation-defined>>;

  @b<procedure> Add_Task (GB : @b<in out> Group_Budget;
                  T : @b<in> Ada.Task_Identification.Task_Id);
  @b<procedure> Remove_Task (GB: @b<in out> Group_Budget;
                    T : @b<in> Ada.Task_Identification.Task_Id);
  @b<function> Is_Member (GB : Group_Budget;
                  T : Ada.Task_Identification.Task_Id) @b<return> Boolean;
  @b<function> Is_A_Group_Member
    (T : Ada.Task_Identification.Task_Id) @b<return> Boolean;
  @b<function> Members (GB : Group_Budget) @b<return> Task_Array;

  @b<procedure> Replenish (GB : @b<in out> Group_Budget; To : @b<in> Time_Span);
  @b<procedure> Add (GB : @b<in out> Group_Budget; Interval : @b<in> Time_Span);
  @b<function> Budget_Has_Expired (GB : Group_Budget) @b<return> Boolean;
  @b<function> Budget_Remaining (GB : Group_Budget) @b<return> Time_Span;

  @b<procedure> Set_Handler (GB   : @b<in out> Group_Budget;

```
                    Handler : @b<in> Group_Budget_Handler);
  @b<function> Current_Handler (GB : Group_Budget)
     @b<return> Group_Budget_Handler;
  @b<procedure> Cancel_Handler (GB        : @b<in out> Group_Budget;
                        Cancelled : @b<out> Boolean);

  Group_Budget_Error : @b<exception>;
@b<private>
   --  not specified by the language
@b<end> Ada.Execution_Time.Group_Budgets;>
```

The type Group_Budget represents an execution time budget to be used by a group of tasks. The type Group_Budget needs finalization (see 7.6). A task can belong to at most one group. Tasks of any priority can be added to a group.

The type Group_Budget_Handler identifies a protected procedure to be executed by the implementation when the budget is @i<exhausted>, that is, reaches zero. Such a protected procedure is called a @i<handler>.

An object of type Group_Budget has an associated non-negative value of type Time_Span known as its @i<budget>, which is initially Time_Span_Zero.
It also has a value of type Group_Budget_Handler, known as its handler.
The handler is said to be @i<set> if it is not null and @i<cleared> otherwise.
The handler of all Group_Budget objects is initially cleared.

An object of type Group_Budget has an associated non-negative value of type Time_Span known as its @i<budget>, which is initially Time_Span_Zero.
The type Group_Budget_Handler identifies a protected procedure to be executed by the implementation when the budget is @i<exhausted>, that is, reaches zero. Such a protected procedure is called a @i<handler>.

An object of type Group_Budget also includes a handler, which is a value of type Group_Budget_Handler. The handler of the object is said to be @i<set> if it is not null and @i<cleared> otherwise. The handler of all Group_Budget objects is initially cleared.

@i<@s8<Dynamic Semantics>>

The procedure Add_Task adds the task identified by T to the group GB; if that task is already a member of some other group, Group_Budget_Error is raised.

The procedure Remove_Task removes the task identified by T from the group GB; if that task is not a member of the group GB, Group_Budget_Error is raised.
After successful execution of this procedure, the task is no longer a member of any group.

The function Is_Member returns True if the task identified by T is a member of the group GB; otherwise it return False.

The function Is_A_Group_Member returns True if the task identified by T is a member of some group; otherwise it returns False.

The function Members returns an array of values of type Task_Identification.Task_Id identifying the members of the group GB. The order of the components of the array is unspecified.

The procedure Replenish loads the group budget GB with the Time_Span value To. Group_Budget_Error is raised if the Time_Span value To is non-positive. Any execution of any member of the group of tasks results in the budget counting down. When the budget becomes exhausted (reaches Time_Span_Zero), the associated handler is executed if the handler of group budget GB is set; the tasks continue to execute.

The procedure Add modifies the budget of the group GB. A positive value for Interval increases the budget. A negative value for Interval reduces the budget, but never below Time_Span_Zero. A zero value for Interval has no effect. A call of procedure Add that results in the value of the budget going to Time_Span_Zero causes the associated handler to be executed if the handler of the group budget GB is set.

The function Budget_Has_Expired returns True if the budget of group GB is exhausted (equal to Time_Span_Zero); otherwise it returns False.

The function Budget_Remaining returns the remaining budget for the group GB. If the budget is exhausted it returns Time_Span_Zero. This is the minimum value for a budget.

The procedure Set_Handler associates the handler Handler with the Group_Budget GB; if Handler is @b<null>, the handler of Group_Budget is cleared, otherwise it is set.

A call of Set_Handler for a Group_Budget that already has a handler set replaces the handler; if Handler is not @b<null>, the handler for Group_Budget remains set.

The function Current_Handler returns the handler associated with the group budget GB if the handler for that group budget is set; otherwise it returns @b<null>.

The procedure Cancel_Handler clears the handler for the group budget if it is set. Cancelled is assigned True if the handler for the group budget was set prior to it being cleared; otherwise it is assigned False.

The constant Min_Handler_Ceiling is the priority value that ensures that no ceiling violation would occur, were a handler to be executed.

The precision of the accounting of task execution time to a Group_Budget is the same as that defined for execution-time clocks from the parent package.

As part of the finalization of an object of type Group_Budget all member tasks are removed from the group identified by that object.

If a task is a member of a Group_Budget when it terminates then as part of the finalization of the task it is removed from the group.

For all the operations defined in this package, Tasking_Error is raised if the task identified by T has terminated, and Program_Error is raised if the value of T is Task_Identification.Null_Task_Id.

An exception propagated from a handler invoked when the budget of a group of tasks becomes exhausted has no effect.

@i<@s8<Erroneous Execution>>

For a call of any of the subprograms defined in this package, if the task identified by T no longer exists, the execution of the program is erroneous.

@i<@s8<Implementation Requirements>>

For a given Group_Budget object, the implementation shall perform the operations declared in this package atomically with respect to any of these operations on the same Group_Budget object. The replacement of a handler, by a call of Set_Handler, shall be performed atomically with respect to the execution of the handler.

@xindent<@s9<NOTES@hr
Clearing or setting of the handler of a group budget does not change the current value of the budget. Exhaustion or loading of a budget does not change whether the handler of the group budget is set or cleared.>>

@xindent<@s9<A Group_Budget_Handler can be associated with several Group_Budget objects.>>

!ACATS test

Tests should be created to check on the implementation of this feature.