Gem #96: Архетипы кода программирования в реальном времени - Часть 4. Спорадическая Задача (Sporadic Task) - Функциональный код и полный OBCS

Автор: Marco Panunzio, University of Padua

Краткое содержание: В ходе программирования систем, работающих в реальном времени, код, обеспечивающий параллелизм и собственно работу в реальном времени часто следует явно или неявно определённым образцам. Такие образцы можно собрать в типовые архетипы и применять в случае необходимости. В этой серии публикаций будет описан набор таких архетипов, которые значительно облегчают разработку систем реального времени.

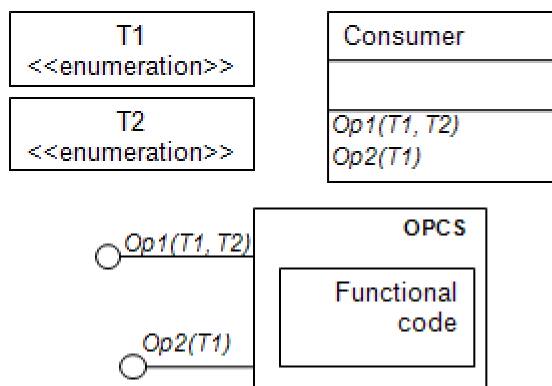
Введение.

В предыдущей Ada Gem мы начинали описывать полный архетип для спорадической задачи. Мы проиллюстрировали структуру задачи и реализацию сложной политики организации очередей для ее агента синхронизации (OBCS). В этой Ada Gem мы завершаем изображение с описанием OPCS, который содержит функциональный код, выполняемый спорадической задачей, и покажите, как мы завершаем объявление OBCS и обеспеченного интерфейса, представленного клиентам спорадической задачи.

Давайте начнём…

Sporadic Task - Функциональный код и полный OBCS

Пришло время создать функциональный код для процедур, выполняемых нашей спорадической задачей. Предположим, нам нужен потребитель, который предоставляет операции Op1 и Op2, как показано на рисунке ниже.



Операции Op1 и Op2 были бы включены в структуру OPCS, которая инкапсулирует их соответствующий функциональный код, разъединяя его от других нефункциональных проблем. OPCS тогда встроен в спорадическую структуру задачи.

Во-первых, мы объявляем, что два простых типа перечисления, T1 и T2, в отдельном пакете, используются нашим функциональным кодом:

```
package Types is
   type T1 is (F1, F2);
   T1_Default_Value : constant T1 := F1;
   type T2 is (X1, X2);
   T2_Default_Value : constant T2 := X1;
end Types;
```

Тогда в другом пакете мы объявляем новый тип скажем `Consumer_FC`, который расширяется `Controlled` (таким образом, это - теговый тип), и имеет две примитивных процедуры `Op1 (T1, T2)` и `Op2(T1)`:

```
with Types;
with Ada.Finalization; use Ada.Finalization;

package Consumer is

   type Consumer_FC is new Controlled with private;
   type Consumer_FC_Ref is access all Consumer_FC'Class;
   type Consumer_FC_Static_Ref is access all Consumer_FC;
   type Consumer_FC_Arr is array(Standard.Integer range <>) of Consumer_FC_Ref;
   type Consumer_FC_Arr_Ref is access Consumer_FC_Arr;

   overriding
   procedure Initialize(This : in out Consumer_FC);

   procedure Op1 (This : in out Consumer_FC; a : in Types.T1; b : in Types.T2);
   procedure Op2 (This : in out Consumer_FC; a : in Types.T1);

private

    type Consumer_FC is new Controlled with null record ...;

end Consumer;
```

`Consumer_FC` - это тип, который представляет то, что мы называем OPCS. Последовательный код указан в телах двух процедур Op1 и Op2:

```
package body Consumer is

   -- procedure Initialize omitted

   procedure Op1(This : in out Consumer_FC; a : in Types.T1; b : in Types.T2)
is
   begin
   -- User-defined sequential code here --
   end Op1;

   procedure Op2 (This : in out Consumer_FC; a : in Types.T1) is
   begin
   -- User-defined sequential code here --
   end Op2;

end Consumer;
```

Теперь давайте завершим определение OBCS и обсудим инстанцирование спорадической задачи.

```
with System;
with Types;
with System_Types;
with Ada.Real_Time;

with Consumer;
with Ada.Real_Time; use Ada.Real_Time;

package Op1_Op2_Sporadic_Consumer is
```

```
   use System; use Types;

   use System_Types;

   -- Nested generic package for instantiating a sporadic task:

   generic
      Thread_Priority : Priority;
      Ceiling : Priority;
      MIAT : Integer;
      -- The OPCS instance
      OPCS_Instance : Consumer.Consumer_FC_Static_Ref;
   package My_Sporadic_Factory is

      procedure Op1(a : in T1; b : in T2);
      procedure Op2(a : in T1);

   private
      -- ...
   end My_Sporadic_Factory;

private

    Param_Queue_Size : constant Integer := 3;
    OBCS_Queue_Size : constant Integer := Param_Queue_Size * 2;

    -- Create data structures to reify invocations of Op1

    type Op1_Param_T is new Param_Type with record
       OPCS_Instance : Consumer.Consumer_FC_Static_Ref;
       a : T1;
       b : T2;
    end record;

    type Op1_Param_T_Ref is access all Op1_Param_T;

    type Op1_Param_Arr is array(Integer range <>) of aliased Op1_Param_T;

    overriding
    procedure My_OPCS(Self : in out Op1_Param_T);

    -- Create data structures to reify invocations of Op2

     type Op2_Param_T is new Param_Type with record
       OPCS_Instance : Consumer.Consumer_FC_Static_Ref;
       a : T1;
    end record;

    type Op2_Param_T_Ref is access all Op2_Param_T;

    type Op2_Param_Arr is array(Integer range <>) of aliased Op2_Param_T;

    overriding
    procedure My_OPCS(Self : in out Op2_Param_T);

    -- Create an OBCS that matches the interface of the OPCS (FC)
    protected type OBCS
     (Ceiling : Priority;
      Op1_Params_Arr_Ref_P : Param_Arr_Ref;
      Op2_Params_Arr_Ref_P : Param_Arr_Ref)
```

```
    is
      pragma Priority(Ceiling);

      entry Get_Request(Req : out Request_Descriptor_T; Release : out Time);
      procedure Op2(a : in T1);
      procedure Op1(a : in T1; b : in T2);

    private

       -- The queue system for the OBCS
       OBCS_Queue : Sporadic_OBCS(OBCS_Queue_Size);
       -- Arrays to store a set of reified invocations for Op1 and Op2
       Op1_Params : Param_Buffer_T(Param_Queue_Size) :=
          (Size => Param_Queue_Size, Index => 1, Buffer =>
Op1_Params_Arr_Ref_P.all);
       Op2_Params : Param_Buffer_T(Param_Queue_Size) :=
          (Size => Param_Queue_Size, Index => 1, Buffer =>
Op2_Params_Arr_Ref_P.all);
       Pending : Standard.Boolean := False;

    end OBCS;

end Op1_Op2_Sporadic_Consumer;
```

По сути, в приведённой выше спецификации: (i) мы объявляем вложенный универсальный пакет (`My_Sporadic_Factory`), который мы используем для создания экземпляра спорадической задачи. Таким образом, мы можем создать несколько отдельных задач, которые отличаются только сроки их атрибуты и свойства (MIAT, приоритет и потолок приоритетом для OBCS); универсальный пакет предоставляет интерфейс к остальной части системы, которая соответствует его OPCS (она обеспечивает Op1 и Op2 в нашем случае); (II) в собственной части родительской пакет (`Op1_Op2_Sporadic_Consumer`), то создание структуры данных для хранения овеществлённых вызовов Op1 и Op2. Это делается путём расширения `Param_Type` (определённого в `System_Types`, см. предыдущий Ada Gem в этой серии) новыми типами `Op1_Param_T` и `Op2_Param_T`, которые являются записями, содержащими параметры вызова и ссылку на OPCS (доступ к FC в нашем случае). Кроме того, мы переопределяем процедуру `My_OPCS`. Поэтому, когда `My_OPCS` вызывается на `Op1_Param_T` или `Op2_Param_T`, он будет отправлен в соответствующую процедуру, которую мы позже определим в теле этого пакета. Читатель может проверить снова, что это то, что действительно происходит, когда спорадический Тип задачи (определённый в предыдущем Gem в этой серии) вызывает процедуру `My_OPCS` после выборки дескриптора запроса от OBCS.

```
with Ada.Real_Time; use Ada.Real_Time;

with Sporadic_Task;
with Types; use Types;

package body Op1_Op2_Sporadic_Consumer is

   use System_Types;

-- Redefinition of My_OPCS. Call Consumer_FC.Op1 and set In_Use to False.

   procedure My_OPCS(Self : in out Op1_Param_T) is
   begin
      Self.OPCS_Instance.Op1(Self.a, Self.b);
      Self.In_Use := False;
   end My_OPCS;
```

```
   -- Redefinition of My_OPCS. Call Consumer_FC.Op2 and set In_Use to False.
   procedure My_OPCS(Self : in out Op2_Param_T) is
   begin
      Self.OPCS_Instance.Op2(Self.a);
      Self.In_Use := False;
   end My_OPCS;

   protected body OBCS is

      procedure Update_Barrier is
      begin
         Pending := (OBCS_Queue.Pending) > 0;
      end Update_Barrier;

      -- Get_Request stores the time of the release of the task,
      -- gets the next request (according to the OBCS queuing policy),
      -- and updates the guard.

      entry Get_Request (Req : out Request_Descriptor_T; Release : out Time)
               when Pending is
      begin
         Release := Clock;
         Get(OBCS_Queue, Req);
         Update_Barrier;
      end Get_Request;

-- When a client calls Op1, the request is reified and put in the OBCS queue.

      procedure Op1(a : in T1; b : in T2) is
      begin
        if Op1_Params.Buffer(Op1_Params.Index).In_Use then
           Increase_Index(Op1_Params);
        end if;

        Op1_Param_T_Ref(Op1_Params.Buffer(Op1_Params.Index)).a := a;
        Op1_Param_T_Ref(Op1_Params.Buffer(Op1_Params.Index)).b := b;
        Put(OBCS_Queue, START_REQ, Op1_Params.Buffer(Op1_Params.Index));
        Increase_Index(Op1_Params);
        Update_Barrier;
      end Op1;

-- When a client calls Op2, the request is reified and put in the OBCS queue.

      procedure Op2(a : in T1) is
      begin
         if Op2_Params.Buffer(Op2_Params.Index).In_Use then
             Increase_Index(Op2_Params);
         end if;

         Op2_Param_T_Ref(Op2_Params.Buffer(Op2_Params.Index)).a := a;
         Put(OBCS_Queue, ATC_REQ, Op2_Params.Buffer(Op2_Params.Index));
         Increase_Index(Op2_Params);
         Update_Barrier;
      end Op2;

   end OBCS;

   package body My_Sporadic_Factory is

       Op1_Par_Arr : Op1_Param_Arr(1..Param_Queue_Size) := (others =>
```

```
                  (False,
                   OPCS_Instance,
                   T1_Default_Value,
                   T2_Default_Value));

      Op1_Ref_Par_Arr : aliased Param_Arr := (Op1_Par_Arr(1)'access,
          Op1_Par_Arr(2)'access, Op1_Par_Arr(3)'access);

      Op2_Par_Arr : Op2_Param_Arr(1..Param_Queue_Size) := (others =>
                 (false,
                  OPCS_Instance,
                  T1_Default_Value));

      Op2_Ref_Par_Arr : aliased Param_Arr := (Op2_Par_Arr(1)'access,
           Op2_Par_Arr(2)'access, Op2_Par_Arr(3)'access);

      -- Creation of the OBCS
      Protocol : aliased OBCS(Ceiling, Op1_Ref_Par_Arr'access,
                                       Op2_Ref_Par_Arr'access);

      -- Indirection to Get_Request of the OBCS

      procedure Getter(Req : out Request_Descriptor_T; Release : out Time) is
      begin
         Protocol.Get_Request(Req, Release);
      end Getter;

      -- Instantiate the generic package using the procedure above

      package My_Sporadic_Task is new Sporadic_Task(Getter);

      Thread : My_Sporadic_Task.Thread_T(Thread_Priority, MIAT);

      -- When a client calls Op1, redirect the call to the OBCS
      procedure Op1(a : in T1; b : in T2) is
      begin
         Protocol.Op1(a, b);
      end Op1;

      -- When a client calls Op2, redirect the call to the OBCS
      procedure Op2(a : in T1) is
      begin
         Protocol.Op2(a);
      end Op2;

   end My_Sporadic_Factory;

end Op1_Op2_Sporadic_Consumer;
```

Тело пакета выше переопределяет `My_OPCS` для каждой операции, предоставляемой внешним клиентам (Op1 и Op2). Переопределение просто гарантирует, что `My_OPCS` вызывает правильную операцию с сохранённым параметром исходного запроса, а затем сигнализирует, что параметры больше не используются (что обеспечивает правильное управление циклическими буферами в OBCS).

Тело OBCS следует той же логике, что и более простой OBCS, описанный в Gem # 92. Процедуры Op1 и Op2 просто расширены для подтверждения запросов на вызовы и правильного сохранения параметров вызовов в дескрипторе запроса.

Наконец, в теле общего пакета `My_Sporadic_Factory` мы создаём OBCS с определённым потолочным приоритетом и очередями для хранения параметров reified вызовов на Op1 и Op2. Спорадический поток создаётся в одном пакете, и мы завершаем изображение, перенаправляя вызовы с Op1 и Op2 в предоставленном интерфейсе структуры задачи на операции с теми же именами в OBCS.

Связанный со статьёй текст программы

Attached Files отсутствуют

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Сведения об авторе отсутствуют

Last Updated: 10/13/2017
Posted on: 12/6/2010

Обсуждение…