Gem #94: Архетипы кода программирования в реальном времени - Часть 3. Спорадическая Задача

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.

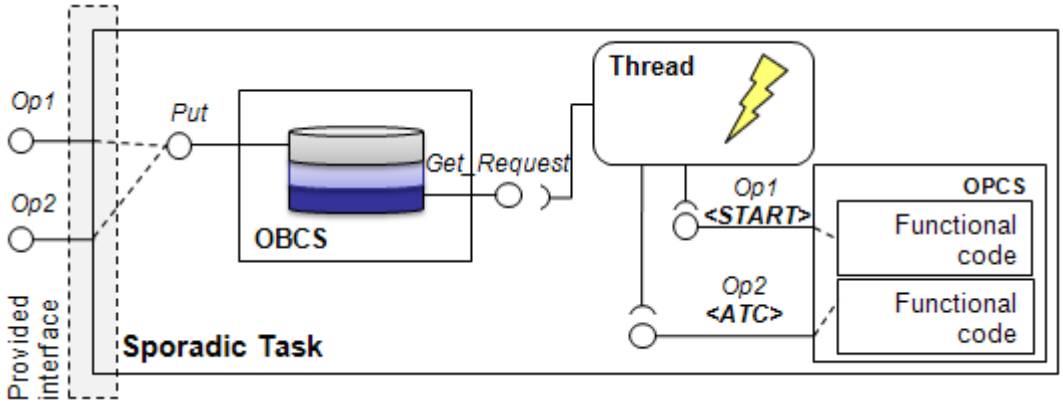
В предыдущем Ada Gem мы описали архетип кода для простой спорадической задачи. Тем не менее, мы признали, что архетип не является полностью удовлетворительным, по крайней мере, по двум причинам: (i) невозможно передать параметры спорадической операции; (ii) агент синхронизации (OBCS) является простым счётчиком ожидающих запросов.

В этом Gem Ada мы проиллюстрируем более сложный архетип, который поддерживает вызов спорадической операции с параметрами. Кроме того, мы хотим исследовать, как возможно расширить функционал OBCS для поддержки сложных политик организации очереди для входящих запросов.

Давайте начнём…

Спорадическая Задача

Архетип спорадической задачи, которую мы хотим проиллюстрировать, изображён на рисунке ниже.

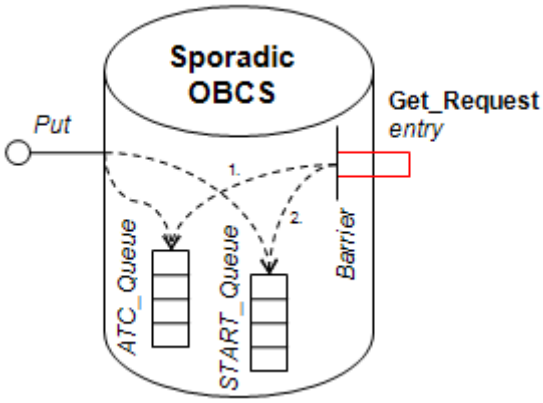


Предположим, что мы хотим создать спорадическую задачу, которая в каждом выпуске может выполнять либо операцию Op1, либо операцию Op2 в соответствии с входящими запросами клиентов. Выполнение различных операций с одной и той же задачей не является необычной потребностью в системах реального времени, особенно когда Платформа выполнения имеет ограниченные вычислительные мощности и ресурсы памяти, а чрезмерное распространение задач может слишком сильно увеличить накладные расходы, снижая производительность системы.

Кроме того, в этом архетипе мы также можем установить некоторый относительный порядок важности между Op1 и Op2. Мы рассматриваем Op1 как номинальную операцию спорадической задачи (операция, обычно вызываемая клиентами – операция по умолчанию), и мы называем её операцией START. Напротив, мы рассматриваем Op2 как операцию модификатора, называемую ATC. Затем мы оговариваем, что отложенные запросы на выполнение Op1 обслуживаются спорадической задачей в порядке FIFO, но запросы на Op2 имеют приоритет над ожидающими

запросами Op1. Этот выбор подразумевает, что операции модификатора могут вызывать одноразовое (в отличие от постоянной START) модификацию поведения по умолчанию выполнения задачи.

OBCS для спорадической задачи



Мы хотим инкапсулировать реализацию этой политики и просто предоставить клиентам этой спорадической задачи набор процедур с подписями Op1 и Op2; роль этих процедур заключается в повторном определении соответствующих запросов на выполнение. Вызов (тип и фактические параметры) записывается в языковую структуру и хранится в OBCS. Когда спорадическая задача получает запрос, она декодирует исходный запрос и вызывает соответствующую операцию с правильными параметрами.

Спорадическая задача - Системные типы и Тип задачи

Давайте теперь посмотрим на набор типов, которые нам нужны для реализации этого архетипа. Они объявлены в модифицированной версии пакета `System_Types`, который мы также использовали в предыдущем Ada Gem.

```
with System;
with Ada.Real_Time; use Ada.Real_Time;
with System_Time;
with Ada.Finalization; use Ada.Finalization;

package System_Types is

   -- Abstract parameter type --
   type Param_Type is abstract tagged record
      In_Use : Boolean := False;
   end record;

   -- Abstract functional procedure --
   procedure My_OPCS (Self : in out Param_Type) is abstract;

   type Param_Type_Ref is access all Param_Type'Class;
   type Param_Arr is array(Integer range <>) of Param_Type_Ref;
   type Param_Arr_Ref is access all Param_Arr;

   -- Request type --
   type Request_T is (NO_REQ, START_REQ, ATC_REQ);

   -- Request descriptor to reify an execution request
   type Request_Descriptor_T is
      record
         Request : Request_T;
         Params : Param_Type_Ref;
```

```
      end record;

   -- Parameter buffer
   type Param_Buffer_T(Size : Integer) is
      record
         Buffer : aliased Param_Arr(1..Size);
         Index : Integer := 1;
      end record;

   type Param_Buffer_Ref is access all Param_Buffer_T;

   procedure Increase_Index(Self : in out Param_Buffer_T);
```

Мы объявили набор типов для представления параметров, Тип, описывающий виды запросов (START_REQ, ATC_REQ и дополнительный вид NO_REQ только ради объяснения), и тип дескриптора запроса для инкапсуляции информации о вызовах Op1 и Op2. Мы также объявляем процедуру My_TOPICS(..), которая представляет собой абстрактную процедуру, представляющую все возможные операции, которые могут быть вызваны спорадической задачей.

Теперь мы продолжим с оставшейся частью спецификации пакета System_Types:

```
   -- Abstract OBCS --
   type OBCS_T is abstract new Controlled with null record;
   type OBCS_T_Ref is access all OBCS_T'Class;

   procedure Put(Self : in out OBCS_T; Req : Request_T; P : Param_Type_Ref)
      is abstract;

   procedure Get(Self : in out OBCS_T; R : out Request_Descriptor_T)
      is abstract;

   -- Sporadic OBCS --
   type Sporadic_OBCS(Size : Integer) is new OBCS_T with
      record
         START_Param_Buffer : Param_Arr(1..Size);
         START_Insert_Index : Integer;
         START_Extract_Index : Integer;
         START_Pending : Integer;
         ATC_Param_Buffer : Param_Arr(1..Size);
         ATC_Insert_Index : Integer;
         ATC_Extract_Index : Integer;
         ATC_Pending : Integer;
         Pending : Integer;
      end record;

   overriding
   procedure Initialize(Self : in out Sporadic_OBCS);

   overriding
   procedure Put(Self : in out Sporadic_OBCS; Req : Request_T; P :
Param_Type_Ref);

   overriding
   procedure Get(Self : in out Sporadic_OBCS; R : out Request_Descriptor_T);

end System_Types;
```

Выше мы объявляем корневой Тип для представления абстрактного OBCS (OBCS_T) и Sporadic_OBCS тип, который реализует политику очередей, которую мы ранее описали.

START_Param_Buffer и ATC_Param_Buffer – это два различных циклично замкнутых буфера, которые используются для хранения вызовов соответствующих типов операций. Кроме того, мы создаём буфер для параметров.

Тело пакета :

```
package body System_Types is

   -- Sporadic OBCS --
   procedure Initialize (Self : in out Sporadic_OBCS) is
   begin
      Self.START_Pending       := 0;
      Self.START_Insert_Index  := Self.START_Param_Buffer'First;
      Self.START_Extract_Index := Self.START_Param_Buffer'First;
      Self.ATC_Pending         := 0;
      Self.ATC_Insert_Index    := Self.ATC_Param_Buffer'First;
      Self.ATC_Extract_Index   := Self.ATC_Param_Buffer'First;
   end Initialize;

   procedure Put(Self : in out Sporadic_OBCS; Req : Request_T; P : Param_Type_Ref) is
   begin
      case Req is
         when START_REQ =>
            Self.START_Param_Buffer (Self.START_Insert_Index) := P;
            Self.START_Insert_Index := Self.START_Insert_Index + 1;
            if Self.START_Insert_Index > Self.START_Param_Buffer'Last then
               Self.START_Insert_Index := Self.START_Param_Buffer'First;
            end if;
            -- Increase the number of pending requests, but do not overcome
            -- the number of buffered ones
            if Self.START_Pending < Self.START_Param_Buffer'Last then
               Self.START_Pending := Self.START_Pending + 1;
            end if;
         when ATC_REQ =>
            Self.ATC_Param_Buffer (Self.ATC_Insert_Index) := P;
            Self.ATC_Insert_Index := Self.ATC_Insert_Index + 1;
            if Self.ATC_Insert_Index > Self.ATC_Param_Buffer'Last then
               Self.ATC_Insert_Index := Self.ATC_Param_Buffer'First;
            end if;

            if Self.ATC_Pending < Self.ATC_Param_Buffer'Last then
               -- Increase the number of pending requests, but do not overcome
               -- the number of buffered ones
               Self.ATC_Pending := Self.ATC_Pending + 1;
            end if;

         when others => null;
      end case;
      Self.Pending := Self.START_Pending + Self.ATC_Pending;
   end Put;

   procedure Get(Self : in out Sporadic_OBCS; R : out Request_Descriptor_T) is
   begin
      if Self.ATC_Pending > 0 then
         R := (ATC_REQ, Self.ATC_Param_Buffer(Self.ATC_Extract_Index));
         Self.ATC_Extract_Index := Self.ATC_Extract_Index + 1;
         if Self.ATC_Extract_Index > Self.ATC_Param_Buffer'Last then
            Self.ATC_Extract_Index := Self.ATC_Param_Buffer'First;
         end if;
         Self.ATC_Pending := Self.ATC_Pending - 1;
         else
            if Self.START_Pending > 0 then
               R := (START_REQ, Self.START_Param_Buffer(Self.START_Extract_Index));
               Self.START_Extract_Index := Self.START_Extract_Index + 1;
               if Self.START_Extract_Index > Self.START_Param_Buffer'Last then
                  Self.START_Extract_Index := Self.START_Param_Buffer'First;
```

```
               end if;
               Self.START_Pending := Self.START_Pending - 1;
            end if;
         end if;
         R.Params.In_Use := True;
         Self.Pending := Self.START_Pending + Self.ATC_Pending;
      end Get;

      procedure Increase_Index(Self : in out Param_Buffer_T) is
      begin
         Self.Index := Self.Index + 1;
         if Self.Index > Self.Buffer'Last then
            Self.Index := Self.Buffer'First;
         end if;
      end Increase_Index;
   end System_Types;
```

В теле пакета мы реализуем желаемую политику очередей. `Procedure Put(..)` просто вставляет представление входящего запроса в очередь запрошенного вида операции (`START_REQ` или `ATC_REQ`). Упорядочивание среди запросов одного и того же вида операций-FIFO.

`Procedure Get(..)` используется для извлечения дескриптора запроса. Мы можем видеть, что пока есть ожидающие запросы ATC, они выбираются на основе их заказа на прибытие. Когда очередь ATC пуста, извлекаются запросы для операций START.

Задача, которая использует этот спорадический OBCS, имеет спецификацию, почти идентичную "простой спорадической задаче", которую мы представили в предыдущем Ada Gem. Единственное отличие состоит в том, что `Get_Request` теперь также получает дескриптор запроса.

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

generic
   with procedure Get_Request(Req : out Request_Descriptor_T;
                              Release : out Time);
package Sporadic_Task is

   task type Thread_T (Thread_Priority : Any_Priority;
                       MIAT : Integer) is
      pragma Priority (Thread_Priority);
   end Thread_T;

end Sporadic_Task;
```

Тело типа `task` выглядит следующим образом:

```
with System_Time; use System_Time;
package body Sporadic_Task is

   task body Thread_T is
      Req_Desc : Request_Descriptor_T;
      Release : Time;
      Next_Time : Time := System_Start_Time;
   begin
      loop
         delay until Next_Time;
         Get_Request (Req_Desc, Release);
         Next_Time := Release + Milliseconds (MIAT);
         case Req_Desc.Request is
            when NO_REQ =>
                    null;
```

```
            when START_REQ | ATC_REQ =>
                    My_OPCS (Req_Desc.Params.all);
            when others =>
                    null;
         end case;
      end loop;
   end Thread_T;

end Sporadic_Task;
```

Обратите внимание, что дескриптор полученного запроса может использоваться для различения действия, выполняемого в соответствии с типом операции (это делается с помощью оператора `case`). В нашем случае, если мы выберем запрос типа `START_REQ` или `ATC_REQ`, мы просто выполним `My_TOPICS`, который динамически отправит запрошенную операцию. Этот механизм будет ясен, когда в следующей статье этого цикла Gem мы завершим картину декларацией Op1 и Op2, как видят их клиенты.

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: 11/8/2010

Обсуждение…

4 ответов на "Gem # 94: Code Archetypes для программирования в реальном времени - часть 3"

1. 8 ноября 2010 года в 23:52

Ань Во (Anh Vo) сказал:

Мне очень нравится этот Code Archetype для серии Real-Time System. Было бы даже лучше, если бы был включён короткий тестовый код.

Кстати, два типа и одна процедура повторяются в пакете `System_Types`, как показано ниже.

```
type Param_Buffer_T(Size : Integer) is
record
Buffer : aliased Param_Arr(1..Size);
Index : Integer := 1;
end record;

type Param_Buffer_Ref is access all Param_Buffer_T;

procedure Increase_Index(Self : in out Param_Buffer_T);
```

2. 9 ноября 2010 года в 21:08

Марко Панунцио (Marco Panunzio) сказал:

Дорогой Ань Во,

Благодарю. Ты прав.

Разделение кода для объяснения привело к дублированию.

Мы исправим Gem. Ещё раз спасибо.

Марко.

3. 9 ноября 2010 года в 21:28

Гэри Дисукес (Gary Dismukes) сказал:

Ань,

Дублированные объявления теперь удалены с конца спецификации пакета. Спасибо что подметил это.

- Гэри

4. 10 ноября 2010 года в 19:57

Ань Во сказал:

Гэри и Марко,

Действительно, дублирование было удалено. Спасибо, что сделали это.

A. Vo