Gem #113: Шаблон Visitor «Посетитель» в Ada

Автор: Emmanuel Briot, AdaCore

Краткое содержание: Шаблон «Посетитель» - это шаблон проектирования, с помощью которого можно выполнять определённые методы по отношению к объекту («посетителю»), основываясь на типе другого объекта. Этот шаблон также называют двойной диспетчеризацией, так как вид вызываемой подпрограммы зависит от типов обоих объектов.

Давайте начнём…

Представьте, что у вас есть модель UML и вы хотите сгенерировать из неё код. Удобный подход заключается в том, чтобы иметь объект "генератор кода", который имеет набор подпрограмм для обработки каждого типа элемента UML (тот, который генерирует код для класса, тот, который генерирует код для операции и т. д.).

Одним из способов реализации этого является использование большой серии операторов `if`, вида `if Obj in Cclass'Class then`, который довольно неэлегантен и неэффективен.

Другой подход заключается в использовании дискриминируемых типов. Затем утверждение `case` на дискриминанте эффективно, и Ada проверит, что все дискриминантные значения покрыты. Проблема в том, что в этом случае необходимо использовать операторы `case` для всех клиентов типов в приложении. Здесь мы предпочитаем использовать тегированные типы, чтобы воспользоваться возможностями ООП Ada, поэтому оператор `case` не может использоваться.

Рассмотрим конкретный пример. Опять же, взяв пример UML, предположим, что у нас есть следующие типы. Они лишь очень приблизительно похожи на фактическую метамодель UML, но будут достаточны для наших целей. На практике эти типы будут автоматически генерироваться из описания метамодели UML.

```
type NamedElement is tagged private;
type CClass is new NamedElement with private;
type PPackage is new NamedElement with private;
```

Кроме того, объявляется класс `visitor` – «Посетитель», который будет переопределён кодом пользователя, например, для предоставления генератора кода, средства проверки модели и т. д:

```
type Visitor is abstract tagged null record;

procedure Visit_NamedElement
   (Self : in out Visitor; Obj : NamedElement'Class) is null;
--  No parent type, do nothing
--  Нет родительского типа, ничего не делать

procedure Visit_CClass (Self : in out Visitor; Obj : CClass'Class) is
begin
   --  In UML, a "Class" inherits from a "NamedElement".
   --  Concrete implementations of the visitor might want to work at the
   --  "NamedElement" level (so that their code applies to both a Class
   --  and a Package, for instance), rather than duplicate the work for each
   --  child of NamedElement. The default implementation here is to call the
   --  parent type's operation.
   -- В UML "Class" наследуется от"NamedElement".
   -- Конкретные реализации visitor может захотеть работать на
   -- уровне "NamedElement" (чтобы, например, их код применялся к обоим и
   -- классам и пакету), а не дублировать работу для каждого
   -- потомка NamedElement. Реализация по умолчанию здесь должна вызываться
```

```
   -- работой родительского типа.

   Self.Visit_NamedElement (Obj);
end Visit_Class;

procedure Visit_PPackage (Self : in out Visitor; Obj : PPackage'Class) is
begin
   Self.Visit_NamedElement (Obj);
end Visit_PPackage;
```

Затем нам нужно добавить одну примитивную операцию `Visit` к каждому из типов, созданных из метамодели UML:

```
procedure Visit (Self : NamedElement; V : in out Visitor'Class) is
begin
   --  First dispatching was on "Self" (done by the compiler).
   --  Second dispatching is simulated here by calling the right
   --  primitive operation of V.

   V.Visit_NamedElement (Self);
end Visit;

overriding procedure Visit (Self : CClass; V : in out Visitor'Class) is
begin
   V.Visit_CClass (Self);
end Visit;

overriding procedure Visit (Self : PPackage; V : in out Visitor'Class) is
begin
   V.Visit_PPackage (Self);
end Visit;
```

Весь описанный выше код полностью систематизирован, и как таковой может и должен генерироваться в максимально возможной степени автоматически. Примитивные операции "`Visit`" никогда не должны переопределяться в пользовательском коде в обычном случае. С другой стороны, "`Visit_...`" примитивы самого посетителя должны быть переопределены, когда это имеет смысл. Реализация по умолчанию предоставляется только для того, чтобы у пользователя был выбор, на каком уровне выполнять переопределение.

Теперь давайте посмотрим, как будет выглядеть генератор кода. Будем считать, что изначально нас интересует только генерация кода для классов. Другие типы элементов (например, операции) будут вызывать реализацию по умолчанию для своего посетителя (`Visit_Operation`, например), который затем вызывает посетителя для его родителя (`Visit_NamedElement`) и так далее, пока мы в конечном итоге вызовет операцию `Visit` с нулевым телом. Так что для них ничего не происходит, и нам не нужно иметь с ними дело явно.

Код будет выглядеть примерно так:

```
type CodeGen is new Visitor with private;

overriding procedure Visit_CClass
   (Self : in out Codegen; Obj : CClass'Class) is
begin
    ...;  --  Do some code generation
end Visit_CClass;

procedure Main is
   Gen : CodeGen;
begin
```

```
      for Element in All_Model_Elements loop  --  Pseudo code
          Element.Visit (Gen);   --  Double dispatching
      end loop;
   end Main;
```

Если бы мы хотели сделать проверку модели, мы бы создали `type Model_Checker`, производный от `Visitor`, который переопределяет некоторые операции `Visit_*`. Тело процедуры `Main` не изменится, за исключением типа `Gen`.

При использовании этого на практике, есть несколько вопросов для решения. Например, типы UML должны иметь доступ к типу `Visitor` (поскольку он отображается в качестве параметра в их операциях). Но `visitor` также должен видеть типы UML по той же причине. Одна из возможностей заключается в том, чтобы поместить все типы в одном пакете. Другой способ – использовать "`limited with`", чтобы обеспечить видимость `access types`, а затем передать `access` к `Visitor'Class` в качестве параметра для `Visit`.

Вот полный пример. Этот пример должен быть скомпилирован с параметром "`- gnat05`", так как он использует функции Ada 2005, такие как предложение `limited with` и нотация вызова с префиксом.

```
with UML;          use UML;
with Visitors;     use Visitors;
with Ada.Text_IO; use Ada.Text_IO;

procedure Main is
   type Code_Generator is new Visitor with null record;

   overriding procedure Visit_CClass
      (Self : in out Code_Generator; Obj : in out CClass'Class) is
   begin
      Put_Line ("Visiting CClass");
   end Visit_CClass;

   Tmp1 : NamedElement;
   Tmp2 : CClass;
   Tmp3 : PPackage;

   Gen  : aliased Code_Generator;

begin
   Tmp1.Visit (Gen'Access);  --  No output
   Tmp2.Visit (Gen'Access);  --  Outputs "Visiting CClass"
   Tmp3.Visit (Gen'Access);  --  No output
end Main;


limited with Visitors;
package UML is
   type NamedElement is tagged null record;
   procedure Visit
      (Self        : in out NamedElement;
       The_Visitor : access Visitors.Visitor'Class);

   type CClass is new NamedElement with null record;
   overriding procedure Visit
      (Self        : in out CClass;
       The_Visitor : access Visitors.Visitor'Class);

   type PPackage is new NamedElement with null record;
```

```
   overriding procedure Visit
      (Self        : in out PPackage;
       The_Visitor : access Visitors.Visitor'Class);
end UML;


with Visitors;  use Visitors;
package body UML is

   procedure Visit
      (Self        : in out NamedElement;
       The_Visitor : access Visitors.Visitor'Class) is
   begin
      The_Visitor.Visit_NamedElement (Self);
   end Visit;

   overriding procedure Visit
      (Self        : in out CClass;
       The_Visitor : access Visitors.Visitor'Class) is
   begin
      The_Visitor.Visit_CClass (Self);
   end Visit;

   overriding procedure Visit
      (Self        : in out PPackage;
       The_Visitor : access Visitors.Visitor'Class) is
   begin
      The_Visitor.Visit_PPackage (Self);
   end Visit;

end UML;


with UML;  use UML;

package Visitors is
   type Visitor is abstract tagged null record;

   procedure Visit_NamedElement
      (Self : in out Visitor; Obj : in out NamedElement'Class);
   procedure Visit_CClass
      (Self : in out Visitor; Obj : in out CClass'Class);
   procedure Visit_PPackage
      (Self : in out Visitor; Obj : in out PPackage'Class);

end Visitors;


package body Visitors is

   procedure Visit_NamedElement
      (Self : in out Visitor; Obj : in out NamedElement'Class) is
   begin
      null;
   end Visit_NamedElement;

   procedure Visit_CClass
      (Self : in out Visitor; Obj : in out CClass'Class) is
   begin
      Self.Visit_NamedElement (Obj);
```

```
   end Visit_CClass;

   procedure Visit_PPackage
      (Self : in out Visitor; Obj : in out PPackage'Class) is
   begin
      Self.Visit_NamedElement (Obj);
   end Visit_PPackage;

end Visitors;
```

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

Сведений об авторе нет.

Last Updated: 10/13/2017
Posted on: 11/7/2011

Обсуждение…