Gem #84: Приложение к стандарту языка Ada «Распределенные системы» (The Distributed Systems Annex)

1 - Простой клиент/сервер

Автор: Thomas Quinot, AdaCore

Краткое содержание: Gem Ada #84 - Этот Gem - первый в серии публикаций, показывающих возможности вспомогательного приложения по распределённым системам, описанного в Справочном Руководстве языка Ada (Приложение E). В Gem Ada #84 будет показано, как архитектура простого клиент/сервера может быть реализована с помощью приложения «Распределённые системы» (ПРС) (the Distributed Systems Annex (DSA)).

Давайте начнём…

Многие аспекты разработки программного обеспечения требуют или могут использовать распределенные технологии. Среди прочих это:

Балансировка нагрузки

Отказоустойчивость

Взаимосвязь между несколькими агентами

В каждом из этих примеров полезно заручиться вкладом нескольких компьютеров для достижения определённой цели скоординированным образом. В распределённом прикладном проекте части обработки, таким образом, назначаются отдельным хостам, которые обмениваются данными для предоставления данной услуги. В языке Ada доля полного приложения, назначенного каждому хосту, называется разделом.

Распределённый дизайн может быть реализован с использованием прямых вызовов услуг связи, предоставляемых средой, что позволяет обмен данными между разделами. Однако это чрезвычайно громоздко и подвержено ошибкам. Поэтому были определены модели распределения, которые представляют собой множества абстракций высокого уровня, позволяющие программисту выражать взаимодействия между компонентами распределённого приложения - возможно, расположенными на разных разделах - в удобных условных понятиях высокого уровня.

Модели распространения поддерживают различные шаблоны связи. Простейшие поддерживают простую передачу сообщений. Более сложные модели также предоставляют более структурированные шаблоны, такие как удалённые вызовы подпрограмм (основанные на границах естественной абстракции, представленные подпрограммами) и распределённые (удалённые) объекты, расширяющие удалённые вызовы подпрограмм в случае вызовов методов в объектно-ориентированном дизайне.

Услуги, предоставляемые промежуточным программным обеспечением распространения (distribution middleware - т. е. реализация модели распределения), могут быть по-разному доступны программисту. Могут использоваться явные API-интерфейсы распространения. В качестве альтернативы, распространение может быть включено в средства, предоставляемые языком программирования. Ada 95 и Ada 2005 включают такие функции, как часть дополнительного Приложения E Справочного руководства (Annex E of the Reference Manual).

В этом первом вводном примере мы рассмотрим простое приложение, управляющее публичной доской объявлений, которую мы хотим сделать доступной для публикации из нескольких разделов. DSA позволяет предлагать услугу очень просто: вы просто пишете декларацию пакета:

```
package Bulletin_Board is
```

```
  pragma Remote_Call_Interface;
  --  This makes the package a Remote Call Interface (RCI), so the subprograms
  --  below are remotely callable.

  --  This pragma enforces some restrictions on the unit to ensure that any
  --  visible subprogram can actually be called remotely, and in particular
  --  that the types of the parameters are suitable for transport over a
  --  communication link from one partition to another.

  subtype Length is Natural range 0 .. 100;
  type News_Item (Author_Length, Message_Length : Length := 0) is record
     Author  : String (1 .. Author_Length);
     Message : String (1 .. Message_Length);
  end record;

  type News_Items is array (Positive range <>) of News_Item;

  procedure Post (Item : News_Item);
  function Whats_Up return News_Items;
end Bulletin_Board;
```

Затем можно написать простой клиент, который будет просто вызывать эти подпрограммы. Тот факт, что эти вызовы могут выполняться удалённо, полностью прозрачен в коде.

```
with Ada.Text_IO;    use Ada.Text_IO;
with Bulletin_Board; use Bulletin_Board;
procedure Post_Message is
   Author, Message : String (1 .. 140);
   Author_Length, Message_Length : Natural;
begin
   Put ("Author name: ");
   Get_Line (Author, Author_Length);
   Put ("Message    : ");
   Get_Line (Message, Message_Length);

   Post (News_Item'
     (Author_Length  => Author_Length,
      Message_Length => Message_Length,
      Author         => Author (1 .. Author_Length),
      Message        => Message (1 .. Message_Length)));
   --  This subprogram call may be remote, but we write it exactly in the
   --  usual way.
end Post_Message;
```

Аналогично, процедура, отображающая все сообщения, может быть записана следующим образом:

```
with Ada.Text_IO;    use Ada.Text_IO;
with Bulletin_Board; use Bulletin_Board;
procedure Display_Messages is
begin
   loop
      Put_Line ("----- all messages -----");
      declare
         Contents : constant News_Items := Whats_Up;
      begin
         for J in Contents'Range loop
            Put_Line (Contents (J).Author & " says:");
            Put_Line (Contents (J).Message);
            New_Line;
```

```
         end loop;
         delay 2.0;
      end;
   end loop;
end Display_Messages;
```

Эта процедура может выполняться на том же самом разделе, что и там, где находится `Bulletin_Board` (язык требует, чтобы каждый модуль `Remote_Call_Interface` был назначен точно одному разделу). Однако, поскольку он использует только видимую подпрограмму, объявленную в `Bulletin_Board` (`Whats_Up`), он также может отлично работать в другом разделе.

Назначение единиц для разделов не обязательно должно быть очевидным в источниках. Один и тот же набор источников может использоваться даже для разных конфигураций разбиения (или используется без разбиения на разделы для создания монолитной версии приложения, и в этом случае вообще нет накладных расходов на распространение).

Реализация процесса разделения приложения DSA определяется реализацией. В GNAT это делается с помощью инструмента `gnatdist` и файла конфигурации `po_gnatdist`. Синтаксис этого файла описан в Руководстве пользователя PolyORB (PolyORB User's Guide).

Вот пример конфигурации приложения для доски объявлений:

```
configuration Dist_App is
   pragma Starter (None);
   -- User starts each partition manually

   ServerP : Partition := (Bulletin_Board);
   --  RCI package Bulletin_Board is on partition ServerP

   ClientP : Partition := ();
   --  Partition ClientP has no RCI packages

   for ClientP'Termination use Local_Termination;
   --  No global termination

   procedure Display_Messages is in ServerP;
   --  Main subprogram of master partition

   procedure Post_Message;
   for ClientP'Main use Post_Message;
   --  Main subprogram of slave partition
end Dist_App;
```

После запуска `po_gnatdist` в этом файле конфигурации создаются два исполняемых файла: `serverp` и `clientp`. `Serverp` будет циклически отображать все опубликованные сообщения, а `clientp` позволит отправлять сообщение на сервер. В этом примере показано, как простой проект клиент / сервер может быть реализован в Ada без какого-либо сетевого программирования.

В будущем Gem мы обсудим проекты удалённых объектов, которые обеспечивают гибкую динамическую связь между разделами.

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

Thomas Quinot (Томас Аквинский) имеет степень инженера от Télécom Paris и доктора философии от Université Paris VI. Основной вклад его исследовательской работы - определение гибкой архитектуры промежуточного (middleware) программного обеспечения, направленной на взаимодействие модели распределения параллельных процессов с целью достижения функциональной совместимости. Он присоединился к AdaCore как Главный Разработчик программного обеспечения (a Senior Software Engineer) в 2003 и ответственен за технологии распределения. Он также участвует в разработке, обслуживании и поддержке компилятора GNAT.

Last Updated: 10/13/2017
Posted on: 4/19/2010

Обсуждение…