# *Gem #85: Приложение к стандарту языка Ada «Распределенные системы» (The Distributed Systems Annex)*

## *2 - Распределенные объекты*

**Автор: Thomas Quinot**, AdaCore

Краткое содержание: Gem Ada #85: Этот Gem - второй в серии публикаций, показывающих возможности вспомогательного приложения по распределённым системам, описанного в Справочном Руководстве языка Ada (Приложение E). В первой публикации было показано, как архитектура простого клиент/сервера может быть реализована с помощью приложения «Распределённые системы» (ПРС) (the Distributed Systems Annex (DSA)). В Gem Ada #85 представляются распределённые объекты, с помощью которых можно устанавливать динамические отношения между компонентами распределённого приложения

**Давайте начнём…**

В предыдущем DSA Gem (Gem Ada #84) мы показали, как подпрограммы в пакете могут быть сделаны удалённо вызываемыми с помощью pragma Remote_Call_Interface (RCI для краткости). Каждый модуль RCI присутствует только в одном разделе распределённого приложения, и любой вызов подпрограммы в таком модуле, выполненный из другого раздела, прозрачно обрабатывается библиотекой времени выполнения распространения (the distribution run-time library).

Этого достаточно для реализации простой связи клиент-сервер, где один раздел идентифицируется как поставщик службы (определяется пакетом RCI) и принимает запросы от других разделов. Разные службы могут предоставляться разными секциями, а службы могут быть клиентами друг друга. Однако эта схема является негибкой в том, что данная услуга может быть предоставлена только одним сервером. Кроме того, связь между службами и разделами является статической.

Однако в некоторых контекстах требуется более гибкое взаимодействие между компонентами приложения: несколько секций могут захотеть предоставить одну и ту же службу по причинам производительности или отказоустойчивости; серверам может потребоваться перезвонить своим клиентам; наконец, может потребоваться динамическое прямое (одноранговое) взаимодействие между секциями, не определяя заранее (до выполнения), кто с кем будет взаимодействовать.

Такая гибкая организация может быть реализована с помощью распределенных объектов. В нераспределённом объектно-ориентированном программировании объект представляет собой сущность с идентификатором (можно ссылаться на него или назначить его), внутренним состоянием и набором методов, общих для всех объектов, принадлежащих одному классу, и представляющих способы взаимодействия любого объекта класса с другими объектами. В распределенном мире эта парадигма естественным образом расширяется, позволяя ссылкам на объекты назначать объекты, расположенные в другой секции.

В DSA распределённые объекты создаются с помощью определённой директивы `pragma: Remote_Types`. Когда эта директива `pragma` применяется к пакету, определённые объявления типов имеют дополнительную семантику, специфичную для распространения. Если вы объявляете ограниченный закрытый Тип с тегами в таком пакете и соответствующий Тип доступа к классу, то этот тип доступа является удалённым доступом к классу (или RACW) и может назначать объекты, расположенные на разделах, отличных от текущего раздела.

Эти ссылки на удалённые объекты могут передаваться как параметры в удалённых вызовах подпрограмм. Например, они могут быть отправлены в пакет RCI или извлечены из него путём передачи их в качестве параметров в удалённых вызовах подпрограмм.

Методы удалённых объектов можно вызвать, просто написав обычный диспетчерский вызов на любой примитивной операции. Вся базовая связь прозрачно обрабатывается библиотекой времени выполнения распространения (the distribution run-time library).

Итак, давайте теперь предположим, что мы хотим позволить пользователям нашего приложения доски объявлений обмениваться прямыми сообщениями друг с другом. Каждый пользователь создаёт экземпляр объекта конкретного типа, производного от пользовательского типа:

```
package Chat_Users is
   pragma Remote_Types;
   --  This package declares a remote object type
   type User is abstract tagged limited private;
   --  Remote objects must be tagged, limited, and private
   type User_Ref is access all User'Class;
   --  This is a remote access-to-class-wide type
   function Name (Who : User) return String;
   procedure Say
     (From : User_Ref;
      To   : User;
      What : String);
   --  The controlling formal 'To' determines the object that calls are sent
to.
   --  The recipient object may be remote. Formal parameter 'From' is a
reference
   --  to the originating user, and can be used to call the user back at a
later
   --  time.
Private

   ...
end Chat_Users;
```

Каждое сообщение, размещенное на доске объявлений может содержать ссылку на автора сообщения :

```
with Chat_Users;
package Bulletin_Board is
  ...
  type News_Item (Message_Length : Natural) is
     Author  : Chat_Users.User_Ref;
     Message : String (1 .. Message_Length);
  end News_Item;
  ...
end Bulletin_Board;
```

Теперь каждый клиент может создать экземпляр конкретного типа, производного от `Chat_Users.User`, и передать `'Access` к этому объекту на доске объявлений, когда он публикует сообщения.

```
with Chat_Users;
package Client is
   --  This is a regular package, no pragma needed
   type Myself_Type is new Chat_Users.User with null record;
   function Name (Self : Myself_Type) return String;
   procedure Say
     (From : Chat_Users.User_Ref;
      To   : Myself_Type;
      What : String);
end Client;
with Ada.Text_IO; use Ada.Text_IO;
```

```
package body Client is
   function Name (Self : Myself_Type) return String is
   begin
      return "Jean-Pierre";
   end Name;
   procedure Say
     (From : Chat_Users.User_Ref;
      To   : Myself_Type;
      What : String)
   is
      pragma Unreferenced (To);
      --  Parameter 'To' is unused within the body. Its purpose is just to
cause
      --  dispatching to the appropriate object instance.
   begin
      Put_Line ("Got a message from " & From.Name);
      --  Dispatching call to Name to retrieve user name of the sender 'From'
      Put_Line (What);
      --  Display received message.
   end Say;
   Myself : aliased Myself_Type;
   ...
end Client;
```

Другие клиенты могут использовать компонент `Author`, извлечённый из доски объявлений, чтобы напрямую связаться с другими клиентами, используя метод `Say`:

```
   Say
     (From    => Myself'Access,
      To      => Some_Item.Author,
      Message => "I like it!");
```

Таким образом, с помощью распределённых объектов можно устанавливать произвольные взаимодействия между секциями. Точнее, это фактически обычные объекты с дополнительным свойством, которое они могут получить путём назначения из других разделов с помощью специальных типов доступа (RACWs). Блоки RCI служат в качестве коммутаторов для первоначального распространения ссылок на удалённые объекты через границы разделов. После распространения этих ссылок разделы могут взаимодействовать напрямую без посредничества RCI.

В будущем Gem (Gem #87) мы обсудим реализацию передачи сообщений на основе почтовых ящиков с помощью приложения распределённые системы (the Distributed Systems Annex - DSA).

### 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: 11/24/2017*
*Posted on: 5/3/2010*

**Обсуждение…**