Gem #90: Приложение к стандарту языка «Распределённые системы» (ПРС) (The Distributed Systems Annex) (DSA)

Часть 4 – ПСР (DSA) и язык С

Автор: Thomas Quinot, AdaCore

Краткое содержание: Этот Gem Ada #90 - четвёртый в серии публикаций, показывающих возможности вспомогательного приложения по распределённым системам, описанного в Справочном Руководстве языка Ada (Приложение E). В предыдущих публикациях серии (Gem Ada #84, Gem Ada #85, Gem Ada #87) было представлено приложение «Распределённые системы» (ПРС) (the Distributed Systems Annex (DSA)), и пояснена реализация различных парадигм взаимодействий. В Gem Ada #90 будет описано, как применять данные инструменты в программе на языке С.

Давайте начнём…

Предыдущие Gems (Gem ###94,95,97) DSA показали, как компоненты в чистом приложении Ada могут быть распределены по нескольким разделам и использовать статические или динамические удалённые вызовы для взаимодействия. Не было бы неплохо, если бы другие языки, такие как С, также могли бы извлечь выгоду из этих функций?

Конечно, вы можете встроить С-код в раздел Ada так же, как и в любое нераспределённое приложение. Ваш код С может также вызвать назад к коду Ada (до тех пор пока подпрограммы Ada имеют соглашения по вызовам языка С). Удалённые (RCI) подпрограммы могут таким образом быть вызваны от С. если вызов происходит на разделе, которому назначен RCI, ничего особенного не происходит, это – просто обычный вызов. На других разделах используются сгенерированные компилятором вызывающие заглушки, и это прозрачный удалённый вызов, как если бы это произошло в коде Ada: удалённая подпрограмма не имеет ничего особенного в точке вызова; вся магия делается в созданных заглушках.

Это все хорошо и прекрасно, но вы все равно должны написать своё полное приложение в Ada, и в частности иметь основную подпрограмму каждого раздела, объявленного в файле конфигурации GNATDIST.

Что делать, если вы хотите включить код клиента или сервера DSA в существующее приложение, написанное на языке С? Это может быть достигнуто путём объединения DSA с автономными библиотеками GNAT, функция, позволяющая разделу Ada генерировать загружаемый модуль, а не полноценный исполняемый образ. Вот как. здесь ...

Перестроить PolyORB с ключом -fPIC

Переключатель `" -fPIC "` предписывает компилятору генерировать так называемый Позиционно-независимый код (Position Independent Code), то есть код, который может динамически загружаться как общая библиотека.

Чтобы иметь раздел DSA в Автономной библиотеке (a stand-alone library), необходимо установить `CFLAGS= " -O2 -g -fPIC "` в вашей среде при вызове сценария настройки PolyORB. (Результирующая сборка PolyORB также может использоваться для обычных приложений.)

Создайте разделы Ada как обычно, также с помощью `-fPIC`

Предположим, например, что приложение имеет серверный раздел, полностью записанный в Ada, и клиентский раздел, предназначенный для встраивания в приложение C/C++ в качестве общего объекта. Раздел сервера будет построен с использованием:

```
po_gnatdist -fPIC xxxx.cfg server_partition
```

Создание фиктивной основной подпрограммы для клиентской стороны

Необходимо предоставить фиктивную основную подпрограмму для клиентского раздела. Вы должны сделать это подпрограммой библиотеки null, которая имеет предложения `WITH` для любого пакета (включая RCIs), на который вы хотите ссылаться со стороны C.

Кроме того, может быть удобно включить в это закрытие пакет "Exports", содержащий подходящие объявления подпрограмм для тех подпрограмм, которые вы хотите вызвать из C, с типами аргументов, совместимыми с C, и используя `Pragma Export`, чтобы дать им понятные имена C. (Обратите внимание, что это не относится к приложению распределённых систем: такой пакет интерфейса обычно создаётся каждый раз, когда вам нужно вызвать код Ada из кода C.)

```
with RCI_1;
...
with RCI_n;
with Exports;
procedure Client is
begin
   null;
end Client;
```

Сборка клиентской библиотеки

Это решающий момент. Построить раздел в качестве отдельной библиотеки, а не обычный исполняемый файл, прошедшие специальный аргументы к GNATDIST:

```
po_gnatdist -fPIC -g xxxx.cfg client_partition \
  -bargs rci_1.ali ... rci_n.ali polyorb-dsa_p-partitions.ali \
         -shared -LClientName \
  -largs -shared
```

В этой командной строке необходимо перечислить файлы ALI для всех пакетов RCI, на которые ссылается клиентский раздел (`rci_1.ali. . rci_n.ali`), а также для внутренних разделов `rci polyorb-dsa_p.ali`.

Вы можете заменить имя "`ClientName`" произвольным префиксом по вашему выбору (он используется для некоторых автоматически генерируемых символов, см. ниже).

Это создаст файл `client_partition`, который можно переименовать в `client_partition.so`.

Вызов клиентской библиотеки из кода C

После создания загружаемого объекта его можно загрузить из кода C с помощью стандартной функции dlopen (3).

Символы из библиотеки можно получить с помощью функции `dlsym(3)`. Сначала необходимо получить из библиотеки символы `ClientNameinit` и `ClientNamefinal`.

`ClientNameinit` в нем соответствует элаборации (elaboration) всех блоков Ada в библиотеке и должно вызываться один раз при загрузке модуля. Это запускает Ada PCS и соединяется с сервером имён DSA для получения начального местоположения единиц RCI.

`ClientNamefinal` соответствует завершению и должен быть вызван один раз, непосредственно перед выгрузкой модуля или завершением приложения (`ClientName` здесь – префикс, который вы передали в командной строке `GNATDIST`, смотри выше по тексту).

Наконец, Вы можете получить и вызвать символы для подпрограмм RCI или любой подпрограммы, экспортированной Вашими модулями Ada, и вызвать их, как если бы они были обычными подпрограммами C.

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: 9/14/2010

Обсуждение…