Основные паттерны J2EE
Business Delegate

Ситуация

В многоярусной распределенной системе для отправки данных через ярусы и для их получения необходимо использовать вызовы удаленного метода. Уязвимым местом клиентов является сложность при работе с распределенными компонентами.

Задача

Компоненты яруса презентации взаимодействуют с бизнес-службами напрямую. Подобные прямые взаимодействия раскрывают для яруса презентации детали интерфейса приложения (API) бизнес-службы, лежащие в основе реализации. В итоге, компоненты яруса презентации будут уязвимы для изменений в реализации бизнес-служб. А именно: при изменении реализации бизнес-служб раскрываемый код реализации в ярусе презентации также должен изменяться.

Кроме того, существует риск нанесения ущерба производительности сети, в виду того, что компоненты яруса презентации, использующие API бизнес-службы, осуществляют в сети слишком много вызовов. Подобное происходит в том случае, когда API, лежащий в основе компонентов яруса презентации используется напрямую, без кэширования на стороне клиента или агрегирующей службы.

И, наконец, раскрытие интерфейсов API службы напрямую клиенту вынуждает его взаимодействовать с продуктами действия сети, связанными с распределенной природой технологии Enterprise JavaBeans (EJB).

Требования

- Клиентам яруса презентации необходимо получать доступ к бизнес-службам.
- Различным клиентам, таким как устройства, Web-клиенты и «толстые» клиенты, необходим доступ к бизнес-службе.
- Интерфейсы API бизнес-служб могут изменяться, в соответствии с развитием требований бизнеса.
- Желательно минимизировать связность между клиентами яруса презентации и бизнес-службой так, чтобы скрыть детали реализации службы, лежащие в ее основе (например, поиск и организация доступа).
- Клиентам может потребоваться реализовать механизм кэширования информации бизнес-службы.
- Желательно уменьшить сетевой трафик между клиентом и бизнес-службами.

Решение

Для уменьшения связности между клиентами яруса презентации и бизнес-службами используйте Business Delegate. Business Delegate скрывает детали реализации бизнес-службы, лежащее в ее основе. Ими могут быть поиск и детали получения доступа к EJB-архитектуре.

Business Delegate действует, как клиентская часть бизнес-абстракции. Он предусмотрен для абстракции, и тем самым скрывает реализацию бизнес-служб. Использование Business Delegate уменьшает связность между клиентами яруса презентации и бизнес-службами системы. В зависимости от стратегии реализации Business Delegate может защищать клиентов от возможной изменчивости в реализации API бизнес-службы. Это потенциально уменьшает количество изменений, которые должны совершаться в коде клиентской части яруса презентации при изменениях в API бизнес-службы или лежащей в его основе реализации.

Между тем, при изменениях API, лежащего в основе бизнес-службы, методы интерфейса в Business Delegate все еще могут требовать модификации. Тем не менее, более предпочтительно, чтобы изменения были сделаны в бизнес-службе, а не в Business Delegate.

Чаще всего разработчики скептически относятся к таким подходам, как абстрагирование бизнес-слоя, так как это требует выполнения дополнительной предварительную работы. Однако использование данного паттерна или его стратегий обеспечивает значительные преимущества. Таким преимуществом является скрытие деталей основной службы. К примеру, клиент может стать прозрачным для служб именования и поиска. Business Delegate также обрабатывает исключения, брошенные бизнес-службами. Примером таких исключений являются java.rmi.Remote и исключения Java Messages Service (JMS). Business Delegate может перехватывать подобные исключения уровня службы и генерировать вместо них исключения уровня приложения. Клиентам легче обрабатывать исключения уровня приложения, это может быть удобно для пользователя. Business Delegate способен также прозрачно осуществлять (в случае необходимости) любые операции повтора или восстановления в событии служебной ошибки, не раскрывая клиенту проблемы до тех пор, пока не будет установлено, что такая проблема не является разрешаемой. Все эти преимущества представляют собой неопровержимые доводы в пользу применения паттерна.

Еще одним преимуществом является то, что delegate может кэшировать результаты и ссылки для удаленных бизнес-служб. Кэширование может значительно улучшить производительность, так как оно ограничивает лишние и потенциально дорогостоящие двусторонние запросы в сети.

Business Delegate использует компонент под названием Lookup Service (служба поиска). Lookup Service отвечает за скрытие деталей основной реализации кода поиска бизнес-службы. Lookup Service может быть написан, как часть Delegate, однако рекомендуется реализовать его в виде отдельного компонента, как это описано в паттерне Service Locator (См. главу «Service Locator»).

Обычно при использовании Business Delegate вместе с Session Facade между ними существует отношение «один к одному». Такое отношение существует за счет того, что логика, которая должна быть инкапсулирована в Business Delegate и относиться к его взаимодействию с множеством бизнес-служб (создание отношения «один ко многим»), будет часто выноситься обратно в Session Facade.

Следует также обратить внимание, что данный паттерн может быть использован не только для яруса презентации и бизнес-яруса, но и для уменьшения связности других ярусов.

Структура

На рисунке 8.1 показана классовая диаграмма паттерна Business Delegate. Клиент запрашивает BusinessDelegate для получения доступа к основной бизнес-службе. Для нахождения требуемого компонента BusinessService, BusinessDelegate использует LookupService.

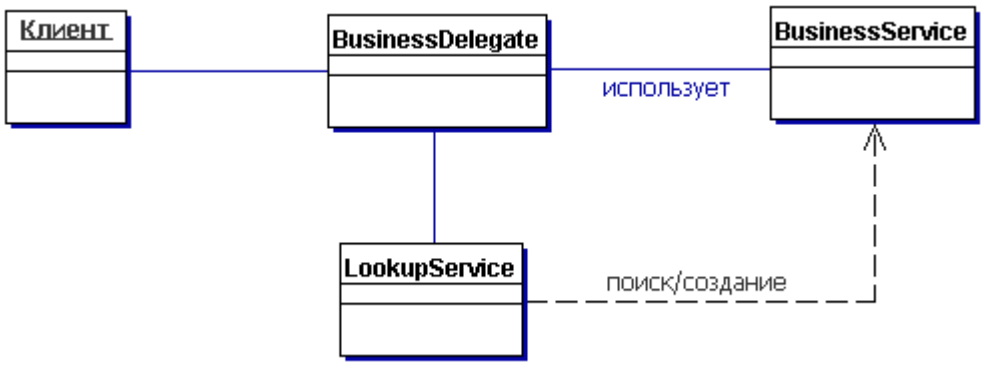

Рисунок 8.1 Классовая диаграмма паттерна BusinessDelegate

Участники и обязательства

На рисунках 8.2 и 8.3 представлены циклограммы, иллюстрирующие обычные взаимодействия в паттерне Business Delegate.

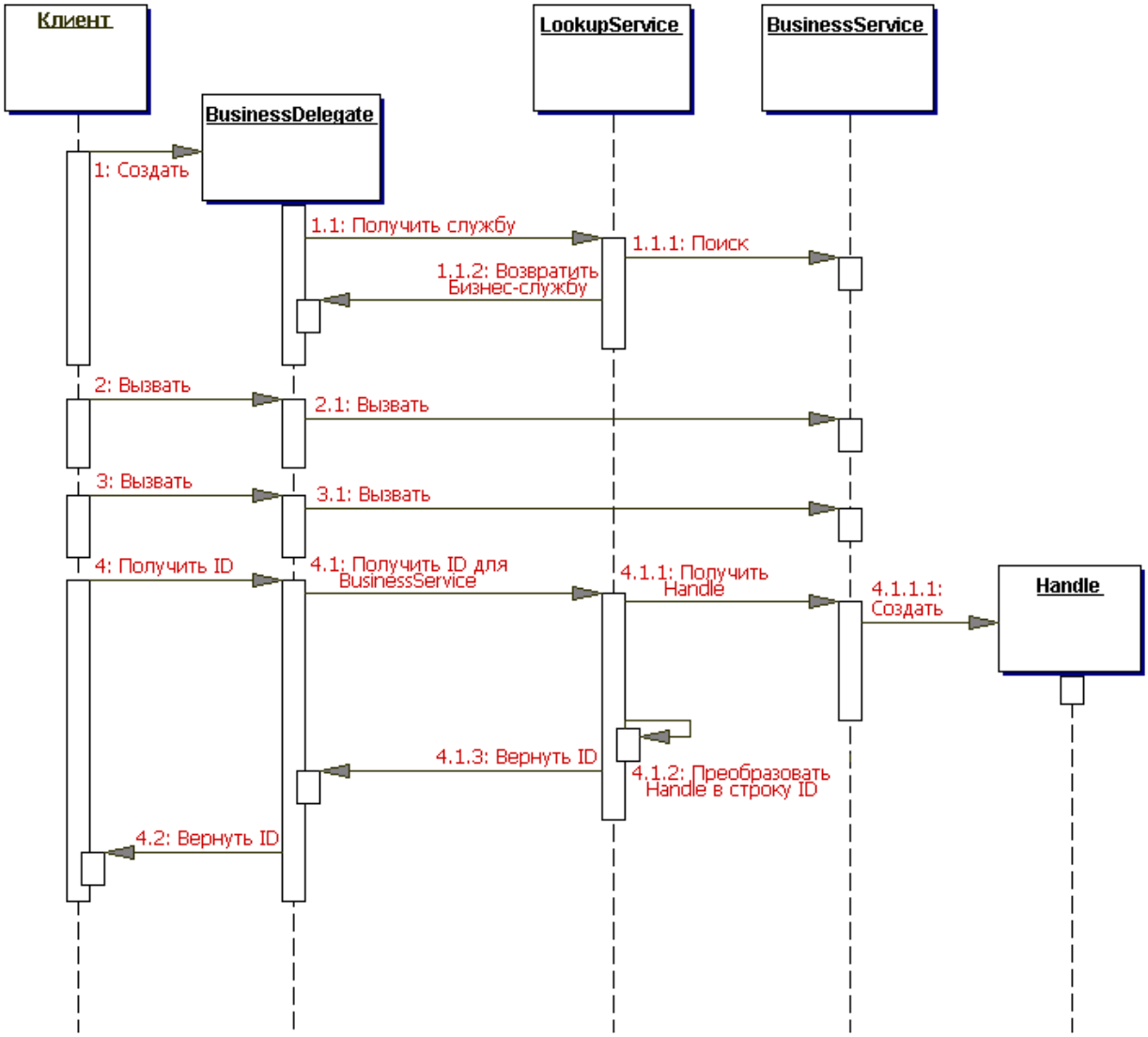

Рисунок 8.2 Циклограмма паттерна BusinessDelegate

Для нахождения бизнес-службы BusinessDelegate использует LookupService. Бизнес-служба используется для вызова бизнес-методов от лица клиента. Метод GetID показывает, что BusinessDelegate может получить для бизнес-службы String-версию handle (такую, как объект EJBHandle), после чего возвратить его клиенту как String. Клиент может использовать String-версию handle позднее для повторного подключения к бизнес-службе, используемой им при получении handle. Подобная методика позволяет избежать новых поисков, так как при использовании handle можно повторно подключаться к экземпляру его бизнес-службы. Следует заметить, что объекты handle реализованы поставщиком контейнера и могут не являться переносимыми между контейнерами разных поставщиков.

Циклограмма, представленная на рисунке 8.3 показывает получение BusinessService (такого, как сессия или компонент-сущность) с использованием его handle.

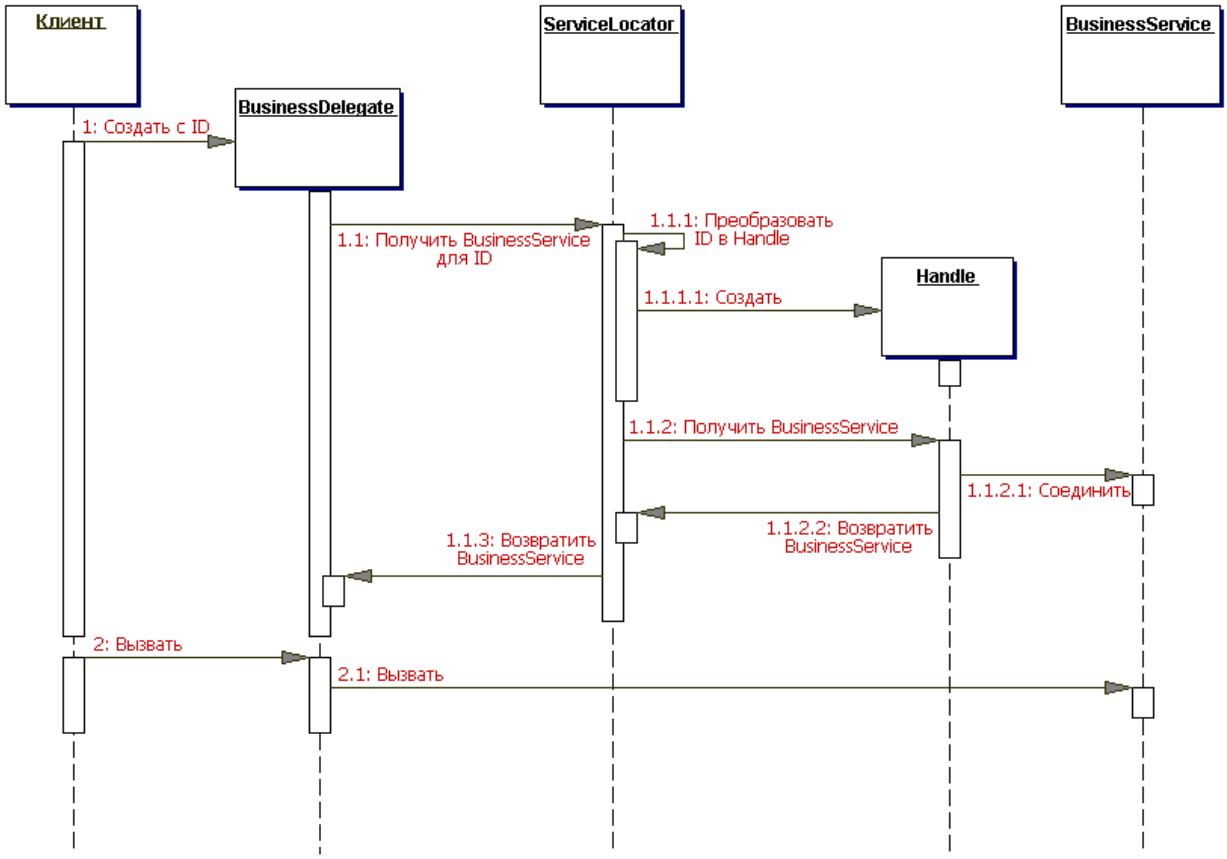

Рисунок 8.3 Циклограмма BusinessDelegate с ID

BusinessDelegateBusinessDelegate служит для обеспечения управления и для защиты бизнес-службы. BusinessDelegate может раскрывать для клиента два типа конструкторов. Один тип запроса создает экземпляр BusinessDelegate без ID, а другой создает экземпляр с ID, где ID является String-версией ссылки на удаленный объект, такой как, EJBHome или EJBObject.

При инициализации без ID BusinessDelegate запрашивает службу из Lookup Service, которая обычно реализована как Service Locator (см. раздел «Service Locator»). Она возвращает Service Factory (например EJBHome). BusinessDelegate определяет, что Service Factory находит, создает или удаляет BusinessService (к примеру, корпоративный компонент).

При инициализации с ID BusinessDelegate использует строку ID для повторного соединения с BusinessService. Таким образом, BusinessDelegate скрывает от клиента детали основной реализации присваивания имен и поиска BusinessService. Более того, клиент яруса презентации никогда не осуществляет удаленные вызовы BusinessSession напрямую. Вместо этого он использует BusinessDelegate.

LookupService

Для нахождения BusinessService BusinessDelegate использует LookupService. LookupService инкапсулирует детали реализации поиска BusinessService.

BusinessService

BusinessService является компонентом бизнес яруса (например корпоративным компонентом или JMS-компонентом), который предоставляет клиенту требуемую службу.

Стратегии

Стратегия Delegate Proxy

Business Delegate раскрывает интерфейс, обеспечивающий клиенту доступ к основным методам API бизнес-службы. В данной стратегии Business Delegate предоставляет функцию прокси для передачи методов клиента сессионному компоненту, который он инкапсулирует. Кроме того, для увеличения производительности Business Delegate может кэшировать любые необходимые данные, включая удаленные ссылки на home сессионного компонента или на удаленные объекты. Business Delegate может также преобразовывать подобные ссылки в String-версии (ID) и наоборот, используя службы Service Locator.

Реализация данной стратегии рассматривается в примере кода.

Стратегия Delegate Adapter

Оказывается, что Business Delegate хорошо подходит для использования в среде B2B при взаимодействии со службами, основанными на платформе J2EE. Несовместимые системы могут использовать XML в качестве языка интеграции. Для интеграции одной несовместимой системы в другую обычно требуется Adapter [GoF]. Пример показан на рисунке 8.4.

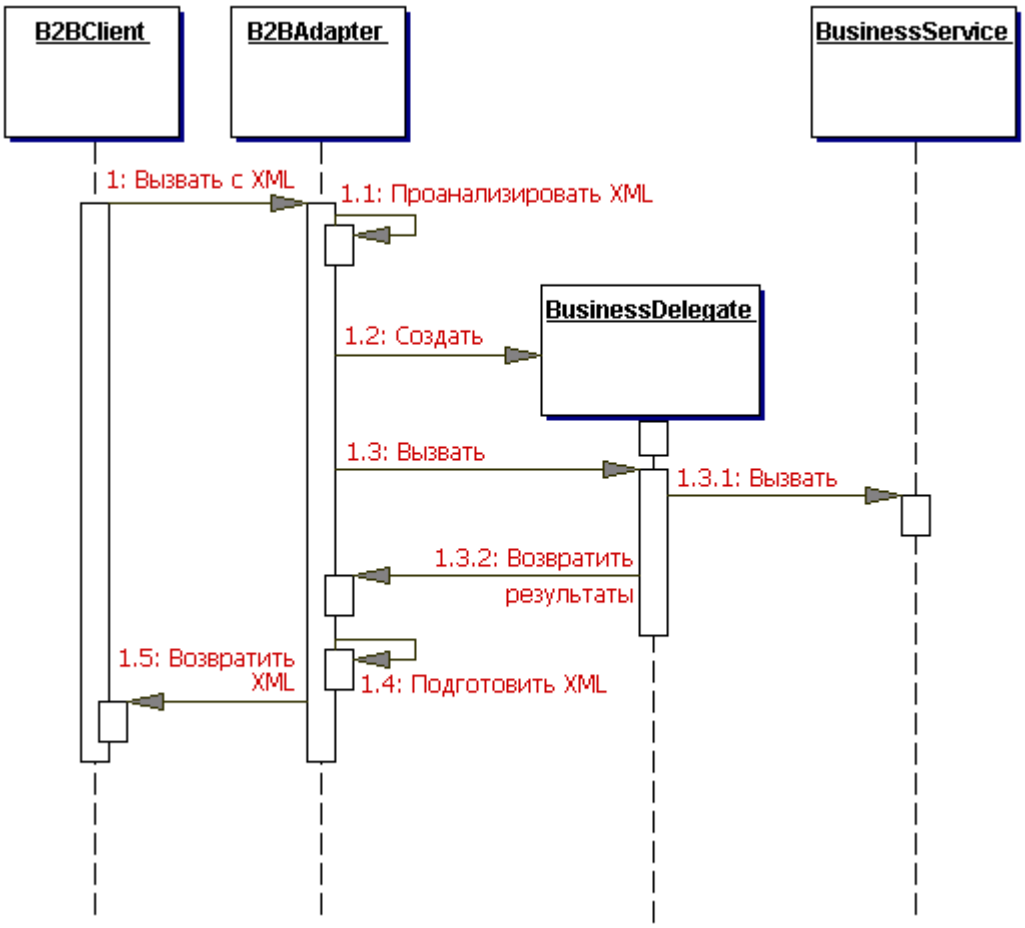

Рисунок 8.4 Использование паттерна Business Delegate со стратегией Adapter

Результаты

- **Уменьшение связности, улучшение управляемости**
  Business Delegate уменьшает связанность между ярусом презентации и бизнес ярусом путем скрытия всех деталей реализации последнего. Это упрощает управление изменениями, так как они централизованы в одной точке, а именно в Business Delegate.
- **Преобразование исключений бизнес-службы**
  Business Delegate отвечает за преобразования любых исключений (относящихся

к структуре сети или инфраструктуре) в исключения на уровне бизнеса, скрывая от клиента особенности основной реализации.

- **Реализация восстановления в случае отказа и синхронизация потока**
  В случае отказа бизнес-службы Business Delegate может реализовать функции автоматического восстановления без раскрытия проблемы клиенту. Если процесс восстановления прошел успешно, клиенту не обязательно знать об ошибке. Если попытка восстановления завершилась неудачно, Business Delegate сообщает клиенту об ошибке. Кроме того, при необходимости, методы business delegate могут быть синхронизированы.

- **Раскрытие бизнес ярусу упрощенного универсального интерфейса**
  Для лучшего обслуживания клиентов Business Delegate может обеспечить вариант интерфейса, предоставляемый основными корпоративными компонентами.

- **Влияние на производительность**
  Для часто используемых запросов службы Business Delegate может предоставлять ярусу презентации службу кэширования (это улучшает производительность).

- **Внедрение дополнительного слоя**
  Business Delegate может рассматриваться как дополнительный слой между клиентом и службой, что добавляет сложность и уменьшает гибкость. Некоторым разработчикам может показаться, что разработка Business Delegates с реализациями, которые используют стратегию Delegate Proxy, является излишней. Однако польза от использования данного паттерна обычно перевешивает недостатки.

- **Скрытие удаленности**
  Прозрачное расположение является одним из преимуществ данного паттерна, однако существует проблема, которая может возникнуть перед разработчиком, рассматривающим удаленную службу в качестве локальной. Подобное случается, если разработчик клиента не понимает до конца, что Business Delegate является клиентской частью модуля доступа к удаленной службе. Обычно вызовы метода Business Delegate приводят к удаленному вызову упакованного метода. Игнорируя это, разработчик может иметь тенденцию к созданию нескольких вызовов метода для выполнения одиночной задачи, что в свою очередь увеличивает сетевой трафик.

Пример кода

Реализация паттерна Business Delegate

Рассмотрим приложение Professional Services Application (PSA), где клиенту яруса Web-узлов требуется получить доступ к сессионному компоненту, реализующему паттерн Session Facade. Паттерн Business Delegate может применяться для разработки класса Delegate под названием ResourceDelegate, который инкапсулирует сложность работы с сессионным компонентом ResourceSession. Такая реализация ResourceDelegate представлена в примере 8.1, а соответствующий удаленный интерфейс для компонента Session Facade под названием ResourceSession показан в примере 8.2.

Пример 8.1 Реализация паттерна Business Delegate – ResourceDelegate

```
// импорт
...

public class ResourceDelegate {

  // Удаленная ссылка для Session Facade
  private ResourceSession session;

  // Класс для объекта Home компонента Session Facade
  private static final Class homeClazz =
  corepatterns.apps.psa.ejb.ResourceSessionHome.class;
```

```
// Default Constructor. Поиск home и соединение
// с сессией путем создание нового.
public ResourceDelegate() throws ResourceException {
  try {
    ResourceSessionHome home = (ResourceSessionHome)
      ServiceLocator.getInstance().getHome(
        "Resource", homeClazz);
    session = home.create();
  } catch(ServiceLocatorException ex) {
    // Преобразовать исключения Service Locator в
    // исключение приложения
    throw new ResourceException(...);
  } catch(CreateException ex) {
    // Преобразовать исключения Session Create в
    // исключение приложения
    throw new ResourceException(...);
  } catch(RemoteException ex) {
    // Преобразовать исключения Remote в
    // исключение приложения
    throw new ResourceException(...);
  }
}

// Конструктор, принимающий ID (Handle id) и
// повторно соединяющийся с предыдущим сессионным компонентом
// вместо создания нового
public BusinessDelegate(String id)
  throws ResourceException {
  super();
  reconnect(id);
}

// Возвратить String ID, который клиент может использовать
// позднее для повторного соединения с сессионным компонентом
public String getID() {
  try {
    return ServiceLocator.getId(session);
  } catch (Exception e) {
    // Бросить исключение приложения
  }
}

// метод для повторного соединения с использованием String ID
public void reconnect(String id)
  throws ResourceException {
  try {
    session = (ResourceSession)
              ServiceLocator.getService(id);
  } catch (RemoteException ex) {
    // Преобразовать исключение Remote в
    // исключение приложения
    throw new ResourceException(...);
  }
}

// Далее следуют бизнес-методы, замещенные в
// Session Facade. При обнаружении любого исключения
// службы эти методы преобразовывают его в исключение
// приложения, такое как ResourceException,
// SkillSetException, и т.д.

public ResourceTO setCurrentResource(
  String resourceId)
  throws ResourceException {
  try {
    return session.setCurrentResource(resourceId);
  } catch (RemoteException ex) {
    // Преобразовать исключение службы в
    // исключение приложения
    throw new ResourceException(...);
  }
}

public ResourceTO getResourceDetails()
```

```
    throws ResourceException {

    try {
      return session.getResourceDetails();
    } catch(RemoteException ex) {
      // Преобразовать исключение службы в
      // исключение приложения
      throw new ResourceException(...);
    }
  }

  public void setResourceDetails(ResourceTO vo)
    throws ResourceException {
    try {
      session.setResourceDetails(vo);
    } catch(RemoteException ex) {
      throw new ResourceException(...);
    }
  }

  public void addNewResource(ResourceTO vo)
    throws ResourceException {
    try {
      session.addResource(vo);
    } catch(RemoteException ex) {
      throw new ResourceException(...);
    }
  }

  // все другие методы proxy для сессионного компонента
  ...
}
```

Пример 8.2 Удаленный интерфейс для ResourceSession

```
// импорт
...
public interface ResourceSession extends EJBObject {

  public ResourceTO setCurrentResource(
    String resourceId) throws
    RemoteException, ResourceException;

  public ResourceTO getResourceDetails()
     throws RemoteException, ResourceException;
  public void setResourceDetails(ResourceTO resource)
     throws RemoteException, ResourceException;

  public void addResource(ResourceTO resource)
      throws RemoteException, ResourceException;

  public void removeResource()
      throws RemoteException, ResourceException;

  // методы для управления blockout time
  // ресурсом
  public void addBlockoutTime(Collection blockoutTime)
      throws RemoteException, BlockoutTimeException;

  public void updateBlockoutTime(
    Collection blockoutTime)
      throws RemoteException, BlockoutTimeException;

  public void removeBlockoutTime(
    Collection blockoutTime)
      throws RemoteException, BlockoutTimeException;

  public void removeAllBlockoutTime()
      throws RemoteException, BlockoutTimeException;

  // методы для снабжения skillsets time
  // ресурсом
  public void addSkillSets(Collection skillSet)
      throws RemoteException, SkillSetException;

  public void updateSkillSets(Collection skillSet)
```

```
        throws RemoteException, SkillSetException;

    public void removeSkillSet(Collection skillSet)
        throws RemoteException, SkillSetException;

    ...
}
```

Родственные паттерны

- **Service Locator**
  Паттерн Service Locator может использоваться для создания Service Locator паттерна Business Delegate, скрытия деталей кода реализации поиска бизнес-службы и кода доступа.
- **Proxy [GoF]**
  Business Delegate может действовать в качестве прокси, обеспечивая дублирование объектов в бизнес ярусе.
- **Adapter [GoF]**
  Business Delegate может использовать паттерн Adapter для соединения с несовместимыми системами.
- **Broker [POSA1]**
  Business Delegate выступает в роли посредника при отделении объектов бизнес яруса от клиентов в других ярусах.