УДК 681.03

А.В. Островский

ПОДХОД К ОБЕСПЕЧЕНИЮ ВЗАИМОДЕЙСТВИЯ ПРОГРАММНЫХ СРЕД JAVA И MS.NET

В работе ставится проблема взаимодействия между компонентами в разнородных средах. Рассматриваются теоретические основы обеспечения взаимодействия с помощью распределенных систем, а также их реализация в технологии CORBA. Полученные результаты применяются для решения практической задачи – наладки взаимодействия между платформами Java и Microsoft .NET с использованием CORBA.

Вступление

В наше время с развитием языков программирования (ЯП) и сред разработки программ все большее значение приобретает проблема взаимодействия между компонентами, созданными в этих средах. Множество готовых программ хранится в специализированных библиотеках (например, Matlab, Mathematica, SAS), в среде систем общего назначения, в Интернете. При этом они написаны на разных ЯП и предназначены для работы в разных средах, так что задача успешной коммуникации между несколькими такими компонентами носит нетривиальный характер. Очевидна и практическая ценность данной задачи: адаптация программы в необходимую среду или написание ее «с нуля» зачастую экономически невыгодны или попросту невозможны.

В данной работе рассматриваются основные принципы построения модели взаимодействия компонентов в разнородных средах в контексте модели разработки программного обеспечения. Вводятся базовые понятия, связанные с распределенными системами; рассматривается архитектура распределенной системы CORBA с теоретической и практической точки зрения. Результаты используются для налаживания взаимосвязи между программными средами Microsoft .NET и Java для реализации алгоритма шифрования с открытым ключом RSA. При этом в приложении используются преимущества каждой из сред: поддержка вычислений с произвольно большими целыми числами в Java и простота написания пользовательского интерфейса в MS.NET.

1. Принципы взаимодействия программ в распределенных системах

Модель взаимодействия компонентов повторного использования (КПИ), а также модели вариабельности и жизнеспособности обоснованы в рамках фундаментального проекта Института программных систем НАН Украины «Разработка теоретического фундамента генерирующего программирования (ГП) и инструментальных средств его поддержки» (шифр III-01-07, 2007−2011) и образуют составные части модели разработки программного обеспечения и программных систем (ПС) [1−3]. К настоящему времени сформировался модельный подход к разработке ПС. Появилось множество разных моделей, среди которых основными и наиболее часто используемыми в практике разработки ПС являются следующие:

1) генерирующая (порождающая) модель GDM (Generative Domain Model);

2) сервисно-ориентированная модель SOA (Service-Oriented Architecture);

3) архитектурно-ориентированная модель MDA (Model Driven Architecture);

4) модель OSI (Open Systems Interconnection) и др.

В данной работе рассматриваются принципы взаимодействия разных компонентов в рамках генерирующей модели ПС, при которой приложение порождается из отдельных элементов – компонентов, сервисов, каркасов и т. п. В основе этой

модели лежит объектно-ориентированный подход, дополненный механизмами порождения КПИ и средствами их взаимодействия. К модели взаимодействия (interconnection), дополняющей модель GDM, в рамках рассматриваемого подхода ГП предъявляется требование обеспечивать взаимодействие разнородных компонентов, работающих в разных операционных средах.

Определяющим аспектом модели взаимодействия является понятие распределенной системы, в рамках которого вводится несколько принятых вспомогательных определений.

Определение 1. Хост – компьютер, на котором выполняются компоненты, входящие в единую систему.

Для обеспечения взаимодействия между компонентами, размещенными на разных хостах, теоретически можно пользоваться примитивами, предоставляемыми сетевой операционной системой (т. е. используя напрямую транспортные каналы) либо промежуточным слоем.

Определение 2. Промежуточный слой (middleware) – слой между сетевой операционной системой и приложениями, предназначенный для решения проблем неоднородности (различия между операционными системами и средами выполнения по отношению к форматам данных хоста) и распределения (размещение компонентов на разных хостах).

В разных распределенных системах используются различные подходы к построению промежуточного слоя. В системах распределенных баз данных, таких как IBM CICS и Tuxedo, используется транзакционно-ориентированный промежуточный слой, который обеспечивает одинаковый механизм обработки транзакций для разных БД. Промежуточный слой, ориентированный на сообщения, применяется для обеспечения надежного обмена сообщениями между компонентами (примером может служить Java Message Queue). Удаленный вызов процедур (Remote Procedure Call, RPC), применяемый в операционных системах MS Windows, позволяет вызывать процедуры за пределами хостов. Такие технологии как CORBA компании OMG, Microsoft COM, удаленный вызов методов (Remote Method Invocation, RMI) в системе Java, используют объектно-ориентированный промежуточный слой, считающийся на сегодняшний день более перспективным.

Определение 3. Распределенная система – система, состоящая из компонентов, которые расположены на объединенных в сеть хостах, взаимодействуют через промежуточный слой, обладают свойством прозрачности (transparency) и внешне их механизмы взаимодействия не отличаются от таковых между компонентами в централизованной системе.

Общая модель взаимодействия компонентов в распределенной системе показана на рис. 1.

При реализации распределенной системы применяется стандартная семиуровневая модель ISO/OSI. В этой модели нижние уровни взаимодействия (физический, канальный, сетевой и транспортный) обеспечиваются сетевой операционной системой, а самый верхний уровень – прикладной – средствами прикладных программистов. Промежуточный слой реализует уровень представления и уровень сессии. Уровень представления отображает структуры данных приложения в однородной форме (не зависящей от среды выполнения). Преобразование данных от прикладного представления к транспортному называется маршалингом, а обратное преобразование – демаршалингом. Для того чтобы маршалинг и демаршалинг данных происходили автоматически и учитывались на этапе компиляции приложения, промежуточный слой применяет клиентские и серверные стабы (stub) – специальные интерфейсы, используемые соответственно компонентом, который осуществляет удаленный вызов, и компонентом, который этот вызов обрабатывает. Что касается уровня сессии, он обеспечивает связь между компонентами за счет одного или нескольких транспортных каналов, отображая ссылки на объекты в хосты, которые эти объекты содержат. В большинстве распределенных систем с объектно-ориентированным промежуточным слоем уровень сессии имеет достаточно сложную

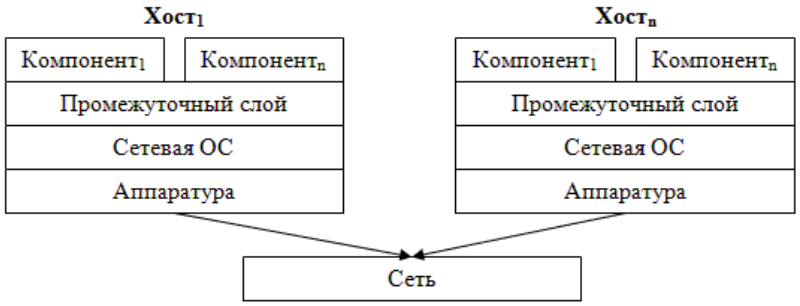

Рис. 1. Модель работы распределенной системы

структуру и в данной работе не рассматривается [4].

2. Распределенная система CORBA

Технология CORBA (Common Request Broker Architecture, общая архитектура брокера объектных запросов) позволяет организовать взаимодействие между компонентами с произвольными средами разработки и выполнения, а также предоставляет возможность удаленного взаимодействия. Объектно-ориентированный промежуточный слой, используемый в CORBA, предусматривает объединение программного кода в объекты, интерфейс доступа к которым описывается на языке описания интерфейсов IDL (Interface Definition Language) [5]. В нем определены атомарные типы: `boolean` (логическая переменная), `char` (символ), `short` (целое число, 2 байта), `long` (целое число, 4 байта), `float` (число с плавающей запятой, 4 байта), `string` (символьная строка) и другие. С помощью атомарных типов строятся более сложные: списки (`sequence`), структуры (`struct`), массивы (`array`), объединения (`union`). Компоненты повторного использования определяются с использованием ключевого слова `interface`; объектная модель IDL поддерживает ограничение доступа к полям объектов и обработку исключительных ситуаций. Типы данных, поддерживаемые ЯП, определенным образом отображаются в типы IDL и обратно. Стандартом установлены отображения для языков Ada, C++, C, Lisp, Smalltalk, Java, COBOL, Object Pascal, PL/1, Python; созданы отображения и для других языков и сред разработки, в том числе для языков среды .NET [6].

Промежуточный слой в технологии CORBA обеспечивается брокером объектных заявок (ORB). ORB производит компиляцию IDL-описаний компонентов, создание клиентских и серверных стабов (скелетонов в терминологии CORBA). Существует большое количество реализаций брокеров, такие как Borland Enterprise Broker, MICO, omniORB, JacORB; брокер объектных заявок входит в пакет Java Development Kit. Помимо этого, существуют пакеты, которые расширяют функции брокера заявок, позволяя компилировать IDL-описания и стабы для дополнительных ЯП (например, для языков платформы Microsoft .NET с помощью пакета IIOPNet). Схема работы распределенной системы CORBA изображена на рис. 2.

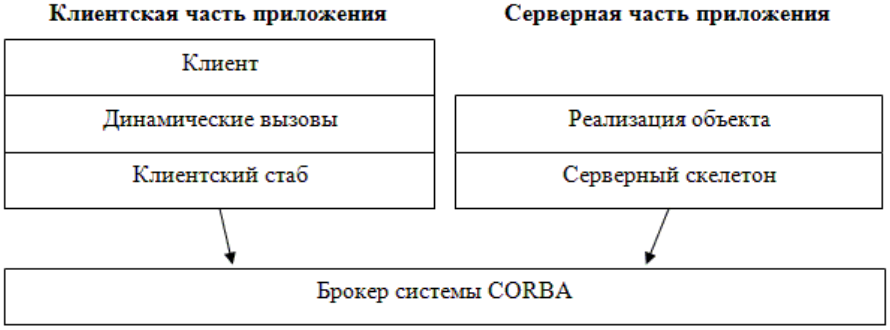

Рис. 2. Схема работы распределенной системы CORBA

3. Практическая реализация взаимодействия компонентов в распределенной системе

Фрагменты кода на языках Java [7], [8] и C# [9], (для платформы .NET), приведенные далее, не претендуют на полноту и используются для демонстрации основных принципов программирования распределенной системы.

3.1. Описание практической задачи

В качестве теста для проверки взаимодействия сред используется программа шифрования с открытым ключом RSA. Согласно алгоритму шифрования, случайным образом выбираются два достаточно больших простых числа p и q. Затем вычисляется их произведение $N = pq$ и функция Эйлера от него $\phi(N) = (p-1)(q-1)$. В качестве открытого ключа используется пара (e, N), где e – некоторое небольшое число (одно из чисел Ферма 17, 257, 65537); в качестве секретного – пара (d, N), где число d взаимно обратно к e по модулю $\phi(N)$. Кодирование числа M состоит в вычислении $X = M^e \pmod N$, раскодирование – в вычислении $M = X^d \pmod N$. Надежность шифрования для алгоритма зависит от величины числа N – в настоящее время надежным считается 1024-битный ключ. Для эффективной реализации RSA нужно, чтобы среда программирования поддерживала операции над большими числами. Одной из таких сред является Java, в ней большие числа реализованы в виде класса `BigInteger` из модуля `java.math`, который предоставляет всю необходимую функциональность: арифметические операции, генерацию случайных простых чисел, модульную арифметику. Таким образом, в данном случае использование Java для серверной части приложения вполне естественно.

CORBA реализует объектный подход к проблеме взаимодействия, поэтому один из главных вопросов при работе с этой технологией состоит в определении структуры объектов. Объект `RSAService`, имплементирующий алгоритм шифрования RSA, содержит следующие методы:

– **newKey()** – генерирует открытый и секретный ключи и возвращает открытый ключ;

– **encode(key, message)** – кодирует текстовое сообщение message открытым ключом key. Поскольку алгоритм предназначен для кодирования чисел, сообщение разбивается на фрагменты, каждый из которых определенным образом переводится в число, не превосходящее N, и затем кодируется согласно вышеописанному методу;

– **decode(key, code)** – декодирует закодированное сообщение секретным ключом, соответствующим открытому ключу key;

– **checkKey(key)** – проверяет открытый ключ key на подлинность. Фактически, этот метод проверяет, существует ли секретный ключ с числом N таким же, как и в открытом ключе. Методы encode и decode будут выполнять эту проверку перед началом вычислений.

При такой архитектуре объекта клиентам передаются только открытые ключи, а соответствующие им секретные хранятся на сервере. Это обеспечивает сокращение длины открытого ключа (вместо пары чисел (e, N) достаточно хранить только N, поскольку e можно оставлять фиксированным), следовательно, уменьшается и количество данных, которыми обмениваются клиент и сервер.

Так как в клиентской среде .NET отсутствует аналог больших чисел, в приложении используется текстовое представление ключа и закодированных сообщений в системе исчисления с достаточно большим основанием (чтобы получившиеся строки имели меньшую длину). Таким образом, все данные, которыми обмениваются клиент и сервер, имеют строковый тип. Как в ЯП Java, так и среде .NET строки являются неизменяемыми (immutable) экземплярами класса `String`, причем поля и методы класса определяются поразному. Из-за этого прямое преобразование строк при переходе от одной среды к другой затруднено. При этом в обеих средах программирования строки хранятся

внутри объектов `String` в виде последовательности символов Unicode фиксированной длины и поэтому соответствуют стандартному типу данных CORBA `wstring`, т. е. передача данных через CORBA не вызывает проблем.

3.2. Описание серверной части приложения

Технология CORBA интегрирована в среду программирования Java. Регистрация объекта в этой распределенной системе подразумевает создание трех базовых частей приложения:

1) интерфейс объекта, содержащий описание всех удаленно доступных полей и методов объекта. Этот интерфейс наследуется от стандартного интерфейса `Remote` из модуля `java.rmi`; каждый его метод декларируется как потенциально вызывающий исключение `RemoteException`, поскольку это исключение возбуждается серверным скелетоном при возникновении каких-либо неполадок. С учетом описанной в разделе 3.1 архитектуры объекта, декларация интерфейса будет выглядеть так, как показано на рис. 3. Помимо четырех функций, упомянутых ранее, в описании интерфейса также объявляется функция `getVersion`, возвращающая версию объекта;

2) класс, являющийся потомком класса `PortableRemoteObject` из модуля `javax.rmi`, который реализует описанный интерфейс. Этот класс определяется с публичным конструктором без аргументов (у `PortableRemoteObject` конструктор имеет область видимости `protected`), причем в простейшем случае никаких действий, помимо вызова родительского конструктора, внутри него не производится;

3) сервер, публикующий ссылку на класс, делая его доступным для других программ с помощью метода `rebind` сервиса именования `Context`. Этот метод связывает имплементацию объекта с его именем в виде строки; другие приложения, использующие технологию CORBA, после этого могут обращаться к этому объекту по его имени.

Как уже упоминалось в разделе 2, в комплект разработчика Java Development Kit встроен брокер объектных заявок. С помощью приложения rmic из этого пакета описание интерфейса на языке Java преобразуется в IDL-описание (рис. 4). Метод `getVersion`, исходя из его имени, при этом автоматически превращается в поле строкового типа с доступом только для чтения (аналогичным образом преобразуются методы, начинающиеся со слова `set`). Это же приложение генерирует стабы, именуемые стандартным образом сообразно имени интерфейса. В данном случае клиентский стаб и серверный скелетон будут называться `_RSAService_Stub` и `_RSAServiceImpl_Tie` соответственно, при этом фактически приложение будет использовать только скелетон, так как клиентская часть приложения написана не на Java.

```
public interface RSAService extends Remote {
   public abstract String getVersion() throws RemoteException;
   public abstract String newKey() throws RemoteException;
   public abstract boolean checkKey(String key) throws
         RemoteException;
   public abstract String encode(String key, String text) throws
         RemoteException;
   public abstract String decode(String key, String code) throws
         RemoteException;
}
```

Рис. 3. Описание интерфейса доступа к объекту в среде Java

```
interface RSAService {
   readonly attribute ::CORBA::WStringValue version;
   ::CORBA::WStringValue newKey( );
   boolean checkKey( in ::CORBA::WStringValue arg0 );
   ::CORBA::WStringValue encode(
      in ::CORBA::WStringValue arg0,
      in ::CORBA::WStringValue arg1 );
   ::CORBA::WStringValue decode(
      in ::CORBA::WStringValue arg0,
      in ::CORBA::WStringValue arg1 );
};
```

Рис. 4. Описание объекта на языке IDL

3.3. Описание клиентской части приложения

В отличие от Java, платформа .NET не располагает встроенными средствами работы с технологией CORBA. Для их взаимодействия используются пакеты сторонних разработчиков, одним из которых является IIOPNet с открытым исходным кодом. С помощью утилиты из этого пакета IDL-описание объекта преобразуется в CLS-совместимую библиотеку, которая затем присоединяется к приложению. Для рассматриваемого приложения библиотека будет содержать описание интерфейса `RSAService`.

Для связи с брокером ORB пакет IIOPNet предоставляет канал связи `IiopClientChannel` и сервис именования `NamingContext`, позволяющий получить ссылку на объект по его имени. Канал фактически играет роль клиентского стаба, выполняя маршалинг и демаршалинг передаваемых через него данных. Метод `resolve` сервиса именования возвращает ссылку на объект по его имени, которое было задано вызовом метода `rebind`. В платформе Microsoft .NET возвращаемое значение имеет тип `MarshalByRefObject`, что позволяет трактовать все обращения к полям и методам объекта как удаленные вызовы. Использование канала и сервиса именования в клиентской части приложения показано на рис. 5.

Общая последовательность шагов построения приложения такая:

1) скомпилировать серверную часть приложения;

2) с помощью утилиты rmic создать серверный скелетон и описание объекта на языке IDL;

3) пользуясь пакетом IIOPNet, преобразовать полученное IDL-описание в CLS-совместимую библиотеку;

4) присоединить библиотеку к клиентской части приложения для использования сервиса.

```
//зарегистрировать канал связи с CORBA
channel = new IiopClientChannel();
ChannelServices.RegisterChannel(channel, false);
// получить сервис через его имя
CorbaInit init = CorbaInit.GetInit();
NamingContext nameService = init.GetNameService("localhost",
1050);
NameComponent[] name = new NameComponent[] {
      new NameComponent("RSAService", "")
};
service = (RSAService)nameService.resolve(name);
//узнать версию сервиса
string result = service.version;
```

Рис. 5. Использование объекта в клиентской части приложения

На рис. 6 и рис. 7 показана реализация интерфейса программы: серверная часть является консольным приложением, клиентская использует графический интерфейс.

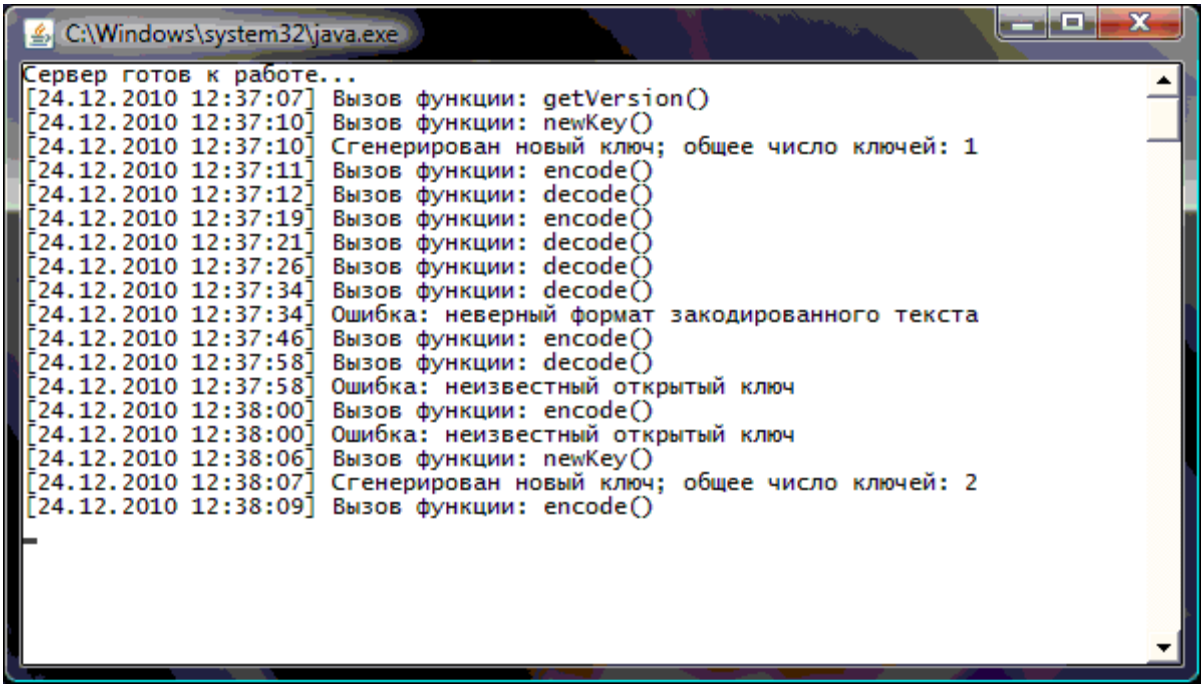

Рис. 6. Интерфейс серверной части приложения

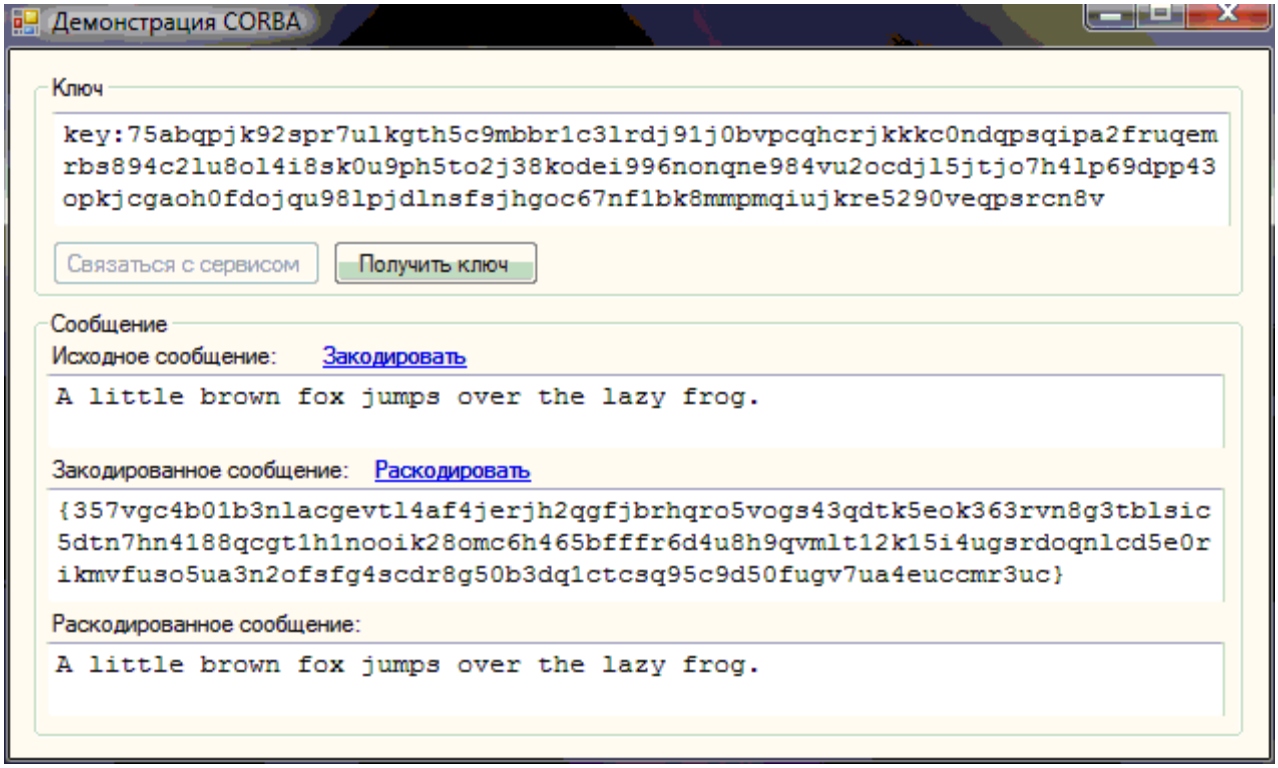

Рис. 7. Интерфейс клиентской части приложения

3.4. Обработка исключительных ситуаций

Часто необходимо, чтобы сервер передавал клиенту отчеты об исключениях, возникающих при обработке запросов. В случае рассматриваемого в данной работе сервиса шифрования с открытым ключом исключительные ситуации могут возникать в нескольких ситуациях:

1) используется некорректное строковое представление чисел (т. е. открытого ключа или закодированного сообщения) при вызове функций encode и decode;

2) предоставленный для кодирования или декодирования открытый ключ не является подлинным.

Спецификация CORBA включает в себя исключения, порождаемые методами объектов, однако использование стандартных исключений Java для этих целей невозможно, так как они не поддерживают маршалинг. Чтобы обойти эту проблему, в серверной части приложения определяется пользовательский тип исключения `RSAException`, имплементирующий интерфейс `java.io.Serializable`, так как в Java все объекты, в том числе и исключения, поддерживающие маршалинг, реализуют этот интерфейс. Способ преобразования данных к транспортному представлению и обратно при этом выбирается платформой Java автоматически. После изменения соответствующим образом деклараций методов `encode` и `decode` в интерфейсе `RSAService` при компиляции IDL-описания объекта будут автоматически сгенерированы описания объектного типа данных (ключевое слово языка IDL `valuetype`) `RSAException`, соответствующего исключению, и его обертки – CORBA-совместимого исключения (`exception`) `RSAEx`.

Помимо этого, будут сгенерированы описания некоторых стандартных классов Java, связанных с пользовательским исключением, например, `Exception` и `Throwable` из модуля `java.lang`, являющиеся его предками в иерархии классов. IDL-файлы в процессе компиляции преобразуются в интерфейсы и помещаются в CLS-совместимую библиотеку, так что их имплементация обеспечивается клиентской частью приложения. Исключение `RSAEx` после компиляции станет потомком класса пользовательского исключения `AbstractUserException`, определенного в пакете IIOPNet; аналогично IDL-описанию, оно будет содержать в себе экземпляр объекта `RSAException`.

Итак, проверка на возникновение исключительных ситуаций в клиентской части приложения будет осуществляться так, как показано на рис. 8.

Написание реализаций интерфейсов является задачей не программиста, пишущего клиентскую часть приложения, а ответственного за серверную часть. Поэтому в реальных приложениях имплементации пользовательских исключений и классов Java компонуются в отдельную CLS-совместимую библиотеку, которая поставляется вместе с интерфейсом CORBA-объекта.

```
try {
   string decMessage = service.decode(key, code);
} catch (RSAEx rsaE) { //Пользовательское исключение
   MessageBox.Show(rsaE.value.message);
   //Действия по обработке исключения
} catch (Exception exc) { //Какое-то другое исключение
   MessageBox.Show(exc.Message);
   //Действия по обработке исключения
}
```

Рис. 8. Обработка исключительных ситуаций клиентом

Заключение

В работе описана реализация задачи взаимодействия прикладных компонентов, разработанных в программных средах Microsoft .NET (клиентская часть приложения) и Java (серверная часть) через CORBA. Приведены общие принципы взаимодействия в распределенных системах, в частности в технологии CORBA, а также принципы написания КПИ в программных средах MS.NET и Java. Разработана структура распределенного приложения, реализующего алгоритм шифрования с открытым ключом с возможностью обработки исключительных ситуаций. Полученный результат взаимодействия демонстрирует эффективность использования каждой из названных сред программирования.

Автор благодарен руководителю фундаментального проекта Е.М. Лаврищевой за постановку задачи исследования и помощь в подготовке статьи.

1. *Лавріщева К.М.* Генерувальне програмування програмних систем і сімейств // Проблеми програмування. – 2009. – № 1. – С. 3−16.
2. *Андон П.І., Лавріщева К.М.* Розвиток фабрик програм в інформаційному світі // Вісник НАН України. – 2010. – № 10. – С. 15−41.
3. *Лаврищева Е.М.* Проблемы интероперабельности разнородных объектов, компонентов и систем. Подходы к ее решению. Спец. выпуск международной конференции “УКРПрО – 2010”, Проблеми програмування.– 2010. – № 2-3. – С. 28−41.
4. *Эммерих В.* Конструирование распределенных объектов. Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft/COM и Java/RMI.: Пер. с англ. – М.: Мир, 2002. – 510 с., с ил.
5. *Лаврищева Е.М., Петрухин В.А.* Методы и средства инженерии программного обеспечения. Учебное пособие. – М.: МФТИ, 2007. – 415 с.
6. *Троэлсен Э.* Язык программирования C# 2005 и платформа .NET 2.0. 3-е издание.: Пер. с англ. – М.: ООО изд. дом Вильямс, 2007. – 1168 с.
7. *Naughton P., Schildt H.* Java 2: The Complete Reference, Third Edition. – Osbourne Publishing, 1999. – 1108 с.
8. http://download.oracle.com/javase/6/docs/ – документация Java.
9. C# Language Specification. Version 3.0. – Microsoft, 2007. – 503 с.

Получено 04.01.2011

Об авторе:

Островский Алексей Викторович,
студент 5 курса Киевского отделения Московского физико-технического института при ФТННЦ НАН Украины,
факультет управления и прикладной математики.

Место работы автора:

Институт программных систем
НАН Украины,
инженер-конструктор I кат.
e-mail: ostrovski.alex@gmail.com