M2MGate

M2MGate Solution

коммуникационная платформа

Техническое описание

M2MGATE

Краткое резюме

Для объединения распределённых программных компонентов существует множество различных решений. M2MGATE SOLUTION - это универсальная телекоммуникационная платформа на основе языка Java, оптимизированная специально для коммуникации через узкополосное соединение, такое как GPRS.

Решение разделено на терминальный компонент, поддерживающий J2ME и серверный компонент, работающий с J2SE. Обмен данными между обоими компонентами производится с помощью метода взаимных вызовов (RMI).

В этом документе описаны обе части продукта, а также части кода в качестве примера, наглядно поясняющего использование Framework.

M2MGATE

Содержание

1 Архитектура

M2MGATE SOLUTION – это коммуникационное решение, разработанное компанией INSIDE M2M GmbH.

Программный комплекс позволяет выполнять удалённые запросы, чтобы обеспечить взаимодействие между распределёнными компонентами решения.

При этом в фокусе оказывается технология GPRS, которая является безопасной, стабильной, узкополосной средой передачи данных.

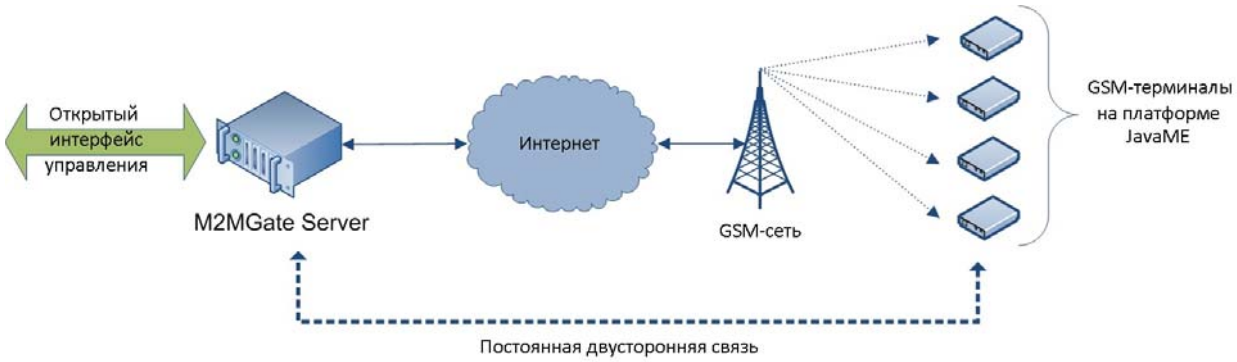

Рисунок 1: Архитектура M2MGATE SOLUTION

Решение включает в себя серверный (M2MGate Server) и терминальный компоненты (M2MGate DeviceServer).

Обе части являются основой для различных специальных применений и могут быть функционально изменены под конкретные требования проекта. Любой используемый в решении GSM-модем постоянно поддерживает TCP-соединение с сервером M2MGate, это позволяет подключённым удалённым устройствам быть всегда «on line».

1.M2MGate Server

Серверное программное обеспечение - это Java-приложение, работающее на JVM (Java Virtual Machine), которая, в свою очередь, поддерживает JavaSE начиная с версии 5.0. Благодаря этому, серверное ПО является независимым от программных платформ и от операционных систем. При необходимости в M2MGate Server можно встроить интерфейс к уже имеющимся информационным системам. К примеру, можно использовать сетевой сервис или интерфейс CORBA. Тогда M2MGate Server будет использоваться в качестве прокси-сервера, предоставляющего доступ к соединённым с ним терминалам.

2.M2MGate DeviceServer

Терминальный компонент – это также Java-приложение. В качестве аппаратной платформы можно в принципе использовать любые GSM-терминалы с поддержкой Java. Обычно используются терминалы, оснащённые процессорами фирмы Cinterion Wireless Modules типов TC65, TC65i или XT65. Эти терминалы поддерживают программную среду Java для устройств с ограниченной производительностью, обозначаемую как JavaME. Программа M2MGate DeviceServer оптимизирована для работы в устройствах с относительно малой вычислительной мощностью, предоставляемой в распоряжение терминалом

Рисунок 2: GSM-терминал Cinterion TC65 Terminal с поддержкой Java

2 Вызов удалённых методов (RMI)

Связь между терминальным и серверным приложением устанавливается по программному интерфейсу вызова удалённых методов (Remote Method Invocation). Благодаря этому обеспечивается чёткое разделение между собственно приложением и процессом обмена данными. Поэтому внутри приложения нет необходимости обращаться к сетевому коду, например, для опроса портов. Вместо этого производится вызов метода, и данные передаются средствами телекоммуникационной платформы.

2.1 Техническая реализация

Техническая реализация сильно зависит от уже имеющегося межплатформенного программного обеспечения, ориентированного на связь, такого, к примеру, как Java RMI. Создаются два объекта заместителя. Они обозначаются как “Stub” и “Skeleton”. Объект “Stub” имеет задачу принять локальный вызов метода внутри виртуальной машины, сериализировать все параметры и передать по сетевому каналу противоположной стороне. Объект “Skeleton”, работающий в пределах другой виртуальной машины, принимает данные, восстанавливает параметры и вызывает, собственно, метод.

Если метод возвращает значение, то оно сериализируется объектом “Skeleton” и передаётся по сетевому каналу объекту “Stub”. Объект “Stub” принимает данные, восстанавливает значение, которое вернулось и предоставляет его изначальному локальному вызову метода, созданному на станции. Рисунок 3 представляет собой графическое изображение описанного процесса.

Важным преимуществом по сравнению с другими решениями является то, что при передаче данных поддерживается очень низкий уровень служебных данных. Весь обмен данных происходит в бинарном формате и сокращён до необходимого минимума.

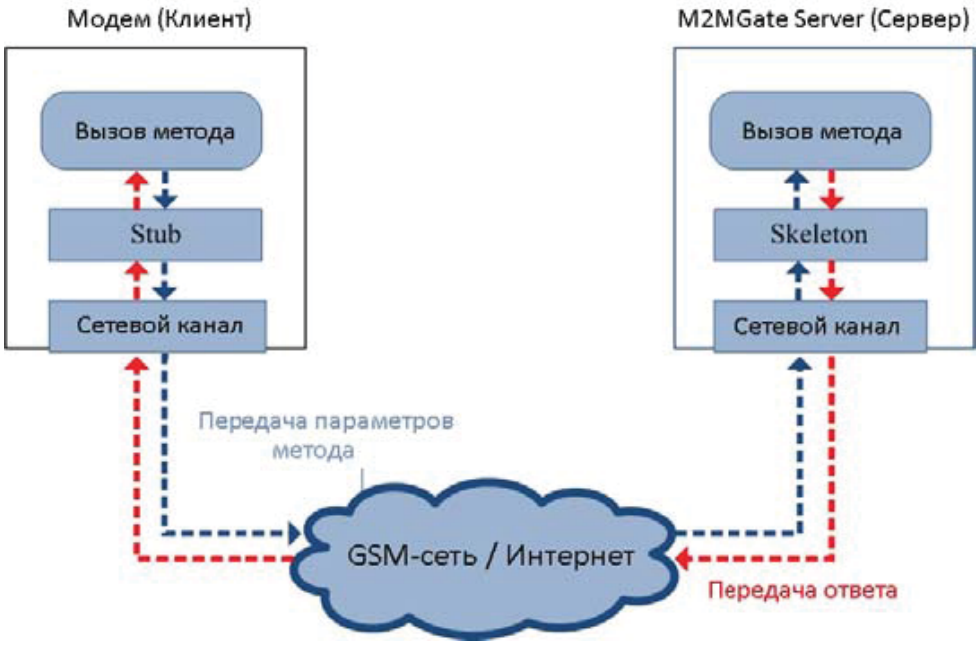

Рисунок 3: Процесс удалённого методического вызова

2 Пример реализации

Следующий пример демонстрируется UML-диаграммой на рисунке 4.

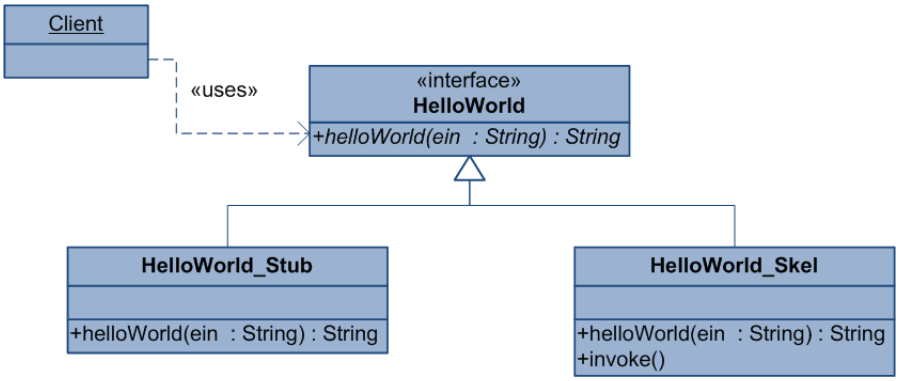

Рисунок 4: UML-диаграмма.

Пример удалённого вызова метода. Диаграмма упрощена из соображений наглядности и отображает не все методы и классы. Отображённые классы обозначены также как “удалённые компоненты”.

Прежде всего, определяется интерфейс, включающий в себя методы, которые позднее должны быть вызваны с удалённого компьютера (см. листинг 1).

```
1  public interface HelloWorld {
2    public String helloWorld ( String myname ) throws IOException , M2MException ;
3  }
```

Листинг 1: Интерфейс Hello World

На следующем этапе реализуются классы Stub и Skeleton, выполняющие сериализирование параметров и возвращаемого значения.

```
1  public class HelloWorld_Stub extends M2MComponent_Stub implements HelloWorld
       {
2
3    public String helloWorld ( String myname ) throws IOException , M2MException {
4      if ( closed ) {
5        throw new M2MClosedException ();
6      }
7      final Trafo trafo = this . trafo ;
8      try {
9        RequestHandler msg = trafo . getRequestHandler ();
10       DataOutput out = msg. invoke (1, ref , 10);
11       out . writeUTF ( myname );
12       DataInput in = msg. fetch ();
13       String result = in. readUTF ();
14       return result ;
15     } catch ( IOException ioex ) {
16       trafo . exceptionOccured ( ioex );
17      throw ioex ;
18     }
19   }
20 }
```

Листинг 2: HelloWorld Stub

Класс «HelloWorld Stub» в листинге 2 расширяет класс M2MComponent Stub. Тем самым предоставляется в распоряжение основная функциональная ориентация, позволяющая связаться с так называемым трансформатором через сеть со станцией. В строке 10 станции сообщается, что производится вызов метода, причём число 10 передаётся в качестве указателя метода. Затем параметр Typ String сериализируется и записывается в сетевой поток. Вызов метода «fetch()» сигнализирует о том, что все параметры записываются в сетевой поток, и что метод может быть выполнен на станции. Затем считывается возвращаемое значение выполненного метода с помощью «readUTF» и значение передаётся обратно.

«HelloWorld Skel» в листинге 3 реализует интерфейс «Skel». Среди прочего, он определяет метод «invoke(int,int,ResponseHandler,Trafo)», который вызывается инфраструктурой. С помощью параметра «mid» сигнализируется, какой метод вызывается со станции, что соответствует указателю метода в строке 10. В строках 12 и 13 параметры сетевого потока считываются, в строке 14 происходит вызов метода, а в строке 15 определяется возвращаемое на станцию значение.

```
1  public class HelloWorld_Skel implements Skel , HelloWorld {
2
3    public void invoke ( int cmd , int mid , ResponseHandler answer , Trafo src )
         throws IOException ,
4        M2MException {
5  switch (mid) {
6  case 0: { // close
7        answer.answer (0) ;
8        return ;
9    }
10   case 10: {
11   DataInput in = answer.paras () ;
12       String myname = in.readUTF () ;
13       String result = helloWorld (myname) ;
14       DataOutput out = answer.answer (1) ;
15       out . writeUTF ( result ) ;
16       answer.done () ;
17       return ;
18   }
19   default : {
20         throw new M2MException(« HelloWorld_Skel . invoke ()unknownMID=» + mid) ;
21     }
22     }
23   }
24
25   public String helloWorld ( String myname) {
26         System.out .println ( ‘ ‘ helloWorld executed on server , myname=’ ’+myname) ;
27         return “I am the server ’ ’;
28     }
29     \\...
30   }
```

Listing 3: HelloWorld Skel

3.1 Коннектор

Коннектор является частью терминального программного компонента и имеет следующие задачи:

- Установка TCP-соединения с сервером M2MGate.
- Инициализация всех зарегистрированных Stub-объектов, так, чтобы на сервере можно было выполнить удалённые вызовы.
- Вызов метода invoke у всех объектов «Skeleton», так, чтобы сервер M2MGate мог выполнять удалённые вызовы метода на терминале.
- Восстановление прерванной связи с M2MGateServer.

Исходный код в листинге 4 демонстрирует использование коннектора

```
1   class Example {
2     public static final String VERSION = «1»;
3     public static void runExample () throws Exception {
4
5               HelloWorld_Stub helloWorld = new HelloWorld_Stub ();
6               ReportStationaryDevice_Stub report =new ReportStationaryDevice_Stub ();
7               Stub [] stubs ={ report , helloWorld };
8               Invokable [] skels = {};
9              String imei = « IMEI « // query imei ..
10             String [] roots = new String []{« socket :// m2mgate .de :5000 «}
11             int reboots = 0; // query reboots
12             short midletVersion = 1;
13
14             StationaryTC65Connector connector ;
15             connector = new StationaryTC65Connector (imei , roots , invokables ,stubs ,0
                       x622 , VERSION , report , midletVersion , reboots , null );
16             connector . connect ();
17
18             String result ;
19             result = helloWorld . helloWorld (« Test «)
20        }
21    }
```

Листинг 4: Использование коннектора.

В строках с 5 по 11 инициализируются все переменные, необходимые для коннектора. Коннектор создаётся в строке 15 со следующими параметрами:

imei – однозначный идентификационный номер терминала

roots – данные соединения с M2MGate Server (IP-адрес и номер порта)

invokables - все параметры Skeletons, которые необходимо зарегистрировать. В этом случае передаётся пустой массив, поскольку M2MGate Server не имеет задачи выполнять методы на терминале.

stubs Stubs, которые нужно инициализировать. Передаются два компонента, среди которых представленный пример HelloWorld

0x622 – идентификационный номер, сообщающий на M2MGate Server, какие из Stubs и Skeletons доступны на терминале

VERSION - ПО терминала, в виде строки символов

report – компонент объекта «Stub», используемый коннектором для передачи информации о соединении на M2MGate Server

midletVersion - ПО терминала, в числовом виде

reboots – число запусков терминала. Это значение передаётся на M2MGate Server.

После инициализации коннектора устанавливается соединение с M2M GateServer в строке 16. Затем в строке 19 производится вызов удалённого метода, исполнение которого происходит на сервере M2MGate.
На рисунке 5 представлен вызов удалённого метода изображённого в качестве последовательной диаграммы.

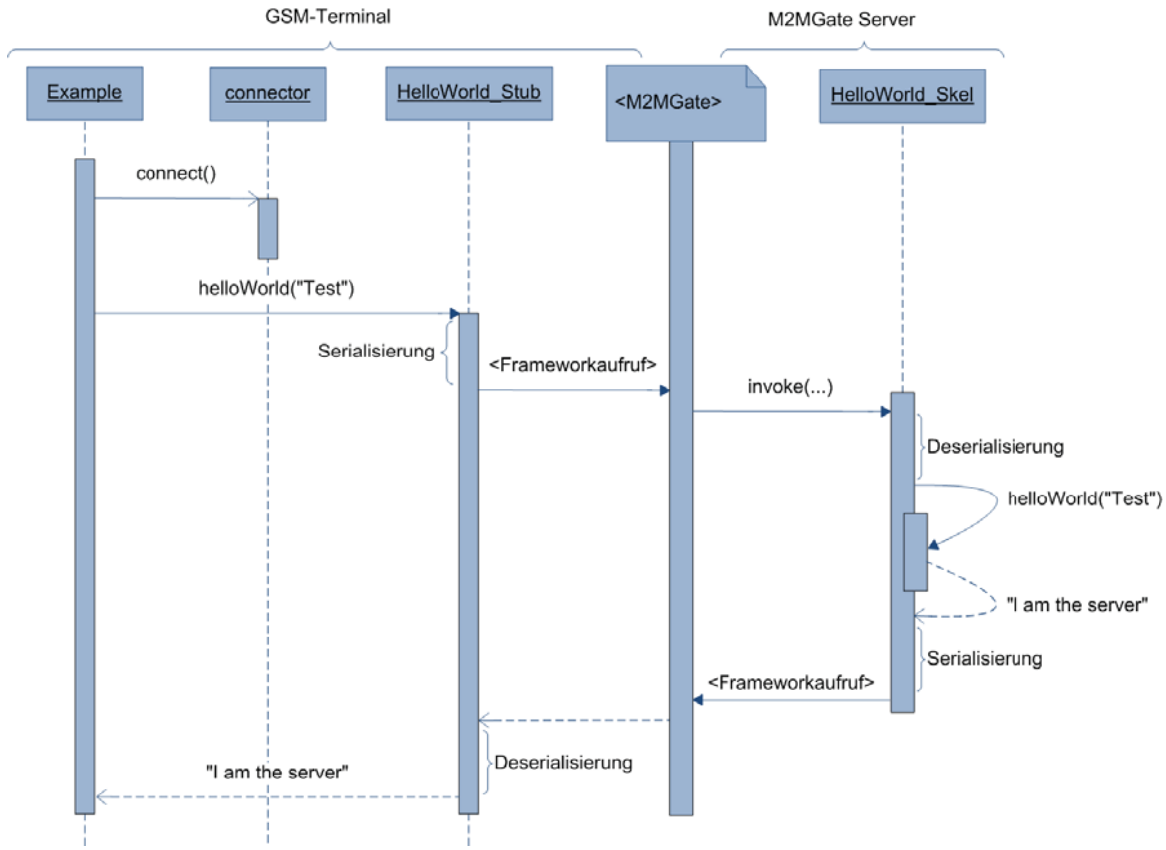

Рисунок 5: Упрощённая последовательная диаграмма для описания представленного примера.
Часть инфраструктуры, для лучшей визуализацией, изображена не поделённой на классы и вызовы удалённого метода.

3.2 Обработка ошибок

Во время вызова удалённого метода легко могут возникать сбои и ошибки. Терминал может завершить работу или попросту выключиться, к примеру, из-за неустойчивого качества линии. Эти сетевые сбои при вызове удалённого метода сигнализируются с помощью IOExceptions. При разработке приложения следует учесть обработку ошибок, которая в этом примере учтена не была, поскольку нельзя исходить из соединения, доступного всегда на все 100%.

Литература

Java RMI. http://java.sun.com/javase/6/docs/technotes/guides/rmi/, июль 2009.