M2MGate Network
Kommunikationsframework

Whitepaper

INSIDE M2M GmbH

18. April 2011

Kurzfassung

Zur Kommunikation verteilter Softwarekomponenten existiert bereits eine Vielzahl von Frameworks. M2MGate Network ist ein in Java entwickeltes Framework, das speziell für die Kommunikation über eine schmalbandige und vergleichsweise instabile Netzwerkverbindung wie GPRS optimiert ist.

Das Framework ist in eine J2ME-konforme Terminal- und eine J2SE-konforme Serverkomponente unterteilt. Die Kommunikation zwischen beiden Komponenten erfolgt mittels gegenseitiger Methodenaufrufe.

Dieses Dokument beschreibt beide Teile des Frameworks sowie eine Beispielimplementierung, die die Verwendung des Frameworks verdeutlicht.

Inhaltsverzeichnis

1 Architektur

M2MGate Network ist ein von der INSIDE M2M GmbH entwickeltes Kommunikationsframework. Das Framework ermöglicht es, entfernte Methodenaufrufe durchzuführen, um so die Kommunikation zwischen verteilten Softwarekomponenten zu gewährleisten. Dabei liegt der Fokus bei GPRS als Übertragungsmedium und dem Umgang mit den damit verbundenen Problematiken in Bezug auf eine sichere, stabile und schmalbandige Datenübertragung.

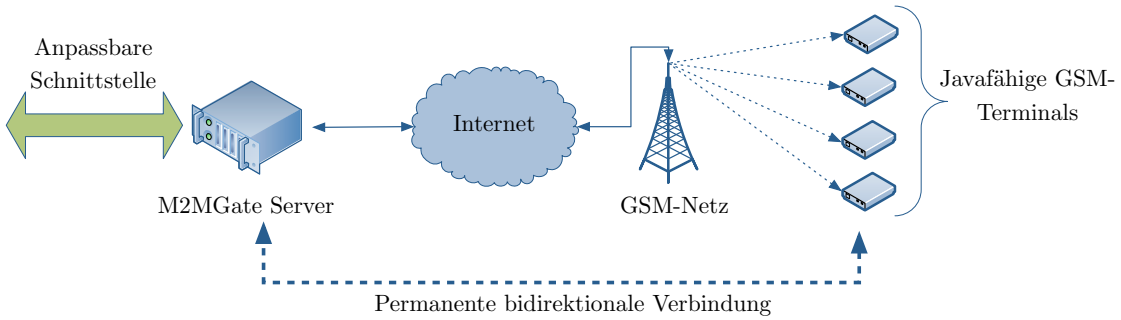

Abbildung 1: Die M2MGate Network Architektur

Das Framework umfasst eine Serverkomponente (M2MGate Server genannt) und eine Terminalkomponente (M2MGate DeviceServer genannt). Beide Teile bilden eine Basis und können für spezielle Anwendungen erweitert werden. Jedes GPRS-Terminal hält permanent eine TCP-Verbindung zum M2MGate Server aufrecht. Über diese Verbindungen erfolgt sämtliche Kommunikation mit den Terminals.

1.1 M2MGate Server

Die Serverkomponente ist eine Java-Anwendung, die innerhalb einer JVM (Java Virtual Machine), die JavaSE ab der Version 5.0 unterstützt, läuft. Damit ist die Komponente plattform- und betriebssystemunabhängig. Je nach Anwendung kann eine Schnittstelle zu bereits vorhandenen Systemen in den M2MGate Server integriert werden. So kann beispielsweise ein Webservice oder eine CORBA Schnittstelle zur Verfügung gestellt werden. Der M2MGate Server dient dann als eine Art Proxy, der einen Zugang zu den mit ihm verbundenen Terminals zur Verfügung stellt.

1.2 M2MGate DeviceServer

Die Terminalkomponente ist ebenfalls eine Java-Anwendung. Als Hardwareplattform können prinzipiell beliebige javafähige GPRS-Terminals verwendet werden. Typischerweise kommen Terminals, die mit Engines der Firma Cinterion Wireless Modules vom Typ TC65, TC65i oder XT65 bestückt sind, zum Einsatz. Diese stellen eine eingeschränkte Java-Laufzeitumgebung, die als JavaME bezeichnet wird, zur Verfügung. Die M2MGate DeviceServer-Software ist sowohl für die Laufzeitumgebung als auch für die relativ geringe Rechenleistung, die das Terminal zur Verfügung stellt, optimiert.

Abbildung 2: Java-fähiges GPRS-Terminal der Firma MC Technologies mit Cinterion TC65 Engine [2]

2 Entfernte Methodenaufrufe

Die Kommunikation zwischen Terminal- und Serverkomponente erfolgt durch gegenseitige Methodenaufrufe. Dadurch ergibt sich eine klare Trennung zwischen der eigentlichen Anwendung und der Datenkommunikation. Innerhalb der eigentlichen Anwendung muss daher kein Netzwerkcode, wie z.B. der Umgang mit Sockets, geschrieben werden. Stattdessen erfolgt der Aufruf einer Methode und die Daten werden mit Hilfe des Frameworks übertragen.

2.1 Technische Umsetzung

Die technische Umsetzung lehnt sich stark an bereits vorhandene kommunikationsorientierte Middlewarelösungen wie z.B. Java RMI [1] an. Es werden zwei Stellvertreterobjekte erzeugt. Diese werden als Stub und Skeleton bezeichnet. Das Stub-Objekt hat die Aufgabe, einen lokalen Methodenaufruf innerhalb einer VM entgegen zu nehmen, sämtliche Parameter zu serialisieren und über einen Netzwerkkanal der Gegenseite zu senden. Das Skeleton-Objekt, welches innerhalb einer anderen VM läuft, empfängt die Daten, stellt die Parameter wieder her und ruft die eigentliche Methode auf.

Falls die Methode einen Rückgabewert liefert, wird dieser von dem Skeleton-Objekt serialisiert und über den Netzwerkkanal zum Stub-Objekt übermittelt. Das Stub-Objekt nimmt die Daten entgegen, stellt den Rückgabewert wieder her und liefert dem ursprünglichen lokalen Methodenaufruf den Rückgabewert, der auf der Gegenstelle erzeugt wurde. Abbildung 3 stellt den beschriebenen Ablauf grafisch dar.

Ein wesentliches Merkmal im Vergleich zu anderen Lösungen ist, dass bei der Datenübertragung der Overhead sehr gering gehalten wird. Sämtliche Kommunikation erfolgt in einem Binärformat und ist auf das Nötigste reduziert.

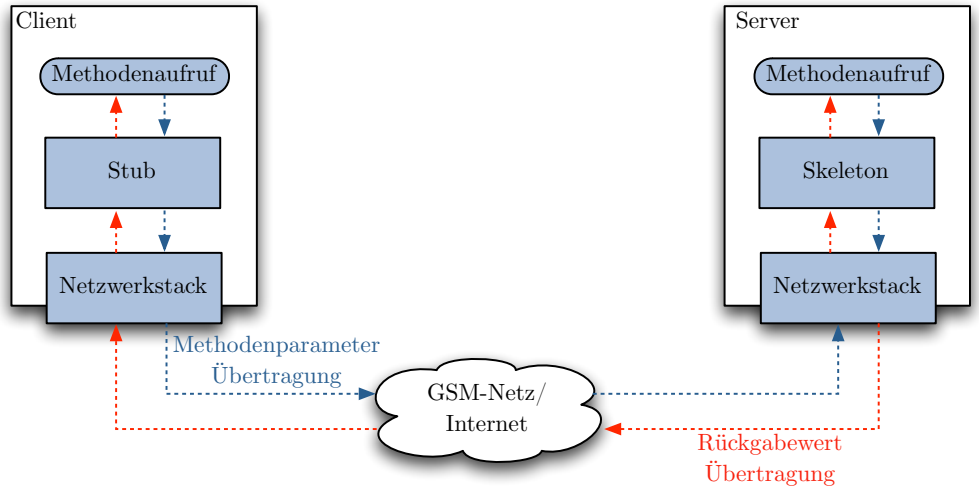

Abbildung 3: Ablauf eines entfernten Methodenaufrufs

3 Beispielimplementierung

Das nachfolgende Beispiel richtet sich nach dem UML-Diagramm in Abbildung 4.

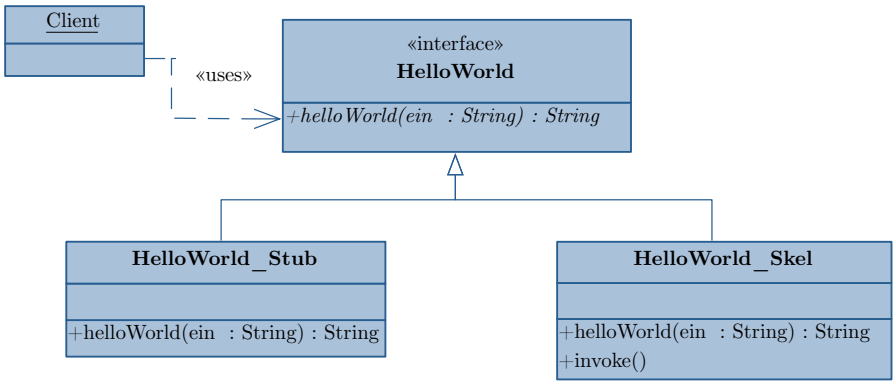

Abbildung 4: UML-Diagramm: Beispiel eines entfernten Methodenaufrufs. Das Diagramm ist vereinfacht und stellt aus Übersichtsgründen nicht alle Methoden und Klassen dar. Die dargestellten Klassen werden auch als “Remote-Komponenten” bezeichnet.

Zunächst wird ein Interface definiert, welches die Methoden enthält, die später entfernt aufgerufen werden sollen (siehe Listing 1).

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

Listing 1: Hello World Interface

Als nächstes werden die Stub- und Skeleton-Klassen implementiert, welche die Parameter- und Rückgabewertserialisierung durchführen.

```
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 }
```

Listing 2: HelloWorld_Stub

Die Klasse “HelloWorld_Stub” in Listing 2 erweitert die Klasse M2MComponent_Stub. Dadurch wird die Grundfunktionalität bereitgestellt, mit einem sogenannten Trafo über das Netzwerk mit der Gegenstelle kommunizieren zu können. In Zeile 10 wird der Gegenstelle mitgeteilt, dass eine Methode aufgerufen wird, wobei die Zahl 10 als Methodenreferenz übergeben wird. Danach wird der Parameter vom Typ String serialisiert in den Netzwerkstrom geschrieben. Der Aufruf der Methode “fetch()” signalisiert, dass alle Parameter in den Netzwerkstrom geschrieben wurden und die Methode auf der Gegenstelle ausgeführt werden kann. Anschließend wird der Rückgabewert der ausgeführten Methode mit “readUTF” eingelesen und zurück gegeben.
“HelloWorld_Skel” in Listing 3 implementiert das Interface “Skel”. Dieses definiert unter anderem die Methode “invoke(int,int,ResponseHandler,Trafo)”, welche vom Framework aufgerufen wird. Über den Parameter “mid” wird signalisiert, welche Methode die Gegenstelle aufrufen will, was der Methodenreferenz aus Zeile 10 entspricht. In den Zeilen 12 und 13 werden die Parameter aus dem Netzwerkstrom ausgelesen, in Zeile 14 wird die Methode aufgerufen und in Zeile 15 wird der Rückgabewert zur Gegenstelle übermittelt.

```
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() unknown MID=" + 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 Connector

Der Connector ist Teil der Terminal-Softwarekomponente und hat folgende Aufgaben:

- Herstellen einer TCP-Verbindung zum M2MGate Server.
- Initialisierung aller registrierten Stub-Objekte, sodass entfernte Methodenaufrufe auf dem Server durchgeführt werden können.
- Aufruf der invoke-Methode bei allen Skeleton-Objekten, sodass der M2MGate Server entfernte Methodenaufrufe auf dem Terminal durchführen kann.
- Wiederherstellung einer unterbrochenen Verbindung zum M2MGateServer.

Der Quellcode in Listing 4 demonstriert die Verwendung des Connectors und führt den Aufruf einer entfernten Methode durch.

```
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 }
```

Listing 4: Benutzung des Connectors

In den Zeilen 5 bis 11 werden alle für den Connector benötigten Variablen initialisiert. Der Connector wird in Zeile 15 mit den folgenden Parametern erzeugt:

imei eindeutige Identifikationsnummer des Terminals

roots Verbindungsdaten zum M2MGate Server (IP-Adresse und Portnummer)

invokables Alle Skeletons, die registriert werden sollen. In diesem Fall wird ein leeres Array übergeben, da der M2MGate Server keine Methoden auf dem Terminal ausführen soll.

stubs Stubs, die initialisiert werden sollen. Es werden zwei Komponenten übergeben, darunter auch das vorgestellte HelloWorld-Beispiel

0x622 eine Identifikationsnummer, die dem M2MGate Server mitteilt, welche Stubs und Skeletons auf dem Terminal verfügbar sind

VERSION Terminal-Software Version als Zeichenkette

report eine Stub-Komponente, die vom Connector verwendet wird, um Verbindungsinformationen an den M2MGate Server zu übertragen

midletVersion Terminal-Software Version als Zahl

reboots Anzahl der Startvorgänge des Terminals. Dieser Wert wird an den M2MGate Server übertragen.

Nach der Initialisierung des Connectors wird die Verbindung zum M2MGate Server in Zeile 16 hergestellt. Anschließend erfolgt in Zeile 19 der Aufruf einer entfernten Methode, deren Ausführung auf dem M2MGate Server stattfindet.

Abbildung 5 stellt einen entfernten Methodenaufruf des vorgestellten Beispiels als Sequenzdiagramm dar.

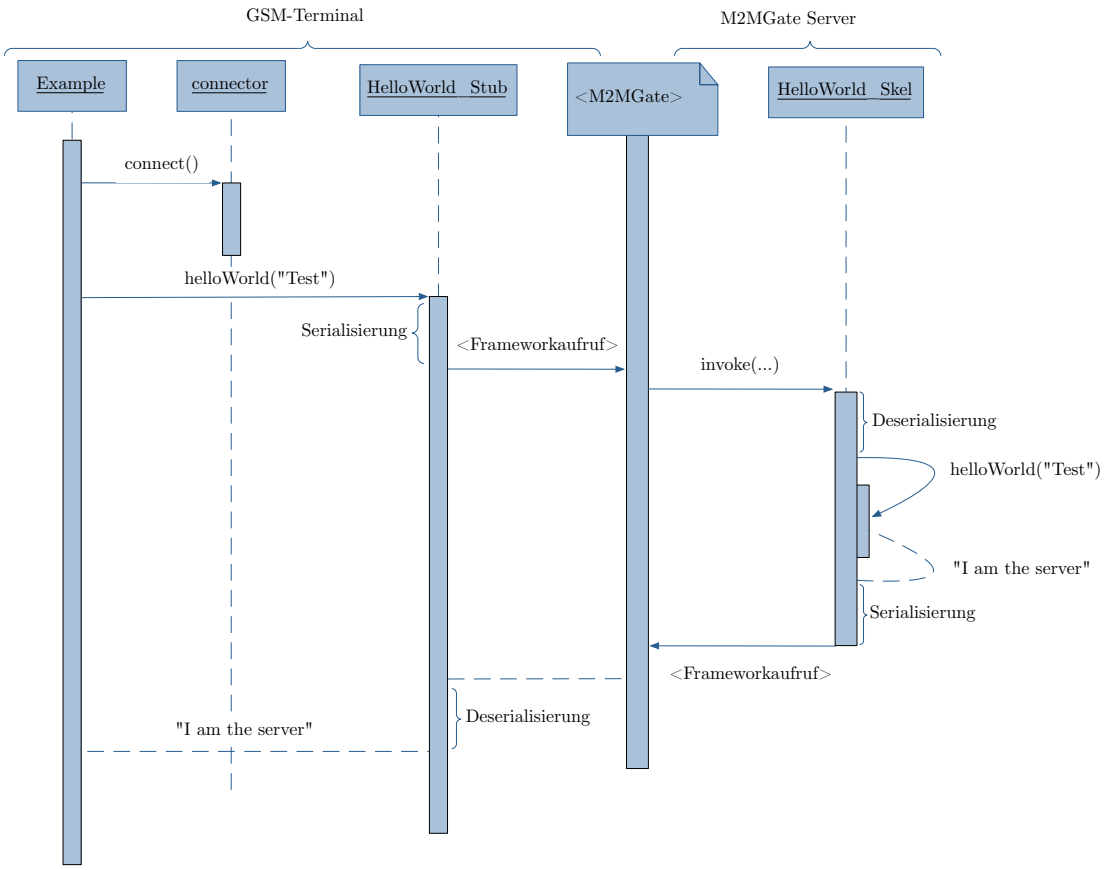

Abbildung 5: Vereinfachtes Sequenzdiagramm zur Beschreibung des vorgestellten HelloWorld-Beispiels. Der Frameworkteil ist aus Übersichtsgründen nicht in einzelnen Klassen und Methodenaufrufen dargestellt.

3.2 Fehlerbehandlung

Während eines entfernten Methodenaufrufs können leicht Fehler auftreten. Das Terminal kann sich beispielsweise in einem Funkloch befinden oder ist einfach ausgeschaltet. Diese Netzwerkfehler werden beim Aufruf einer entfernten Methode durch IOExceptions signalisiert. Beim Entwickeln der Anwendung muss eine Fehlerbehandlung, die in diesem Beispiel vernachlässigt wurde, vorgesehen werden, da nicht von einer 100-prozentig verfügbaren Funkverbindung ausgegangen werden kann.

4 Aufzeichnung von Meta-Informationen

Der M2MGate Server ist in der Lage im laufenden Betrieb Informationen über den Zustand und die Aktionen der von ihm verwalteten GPRS-Terminals aufzuzeichnen. Hierzu sendet jedes GPRS-Terminal diese Informationen in definierbaren Zeitabständen an den M2MGate Server, der die Daten zur weiteren Auswertung in einer Datenbank ablegen kann. Nachfolgend werden die hierzu verwendeten Tabellen beschrieben.

4.1 Beschreibung der Tabellen

4.1.1 cellhistory

In der Tabelle cellhistory wird gespeichert, wie lange ein GPRS-Terminal in einer Mobilfunkzelle eingebucht war.

Name	Beschreibung	Datentyp
id (PK)	Fortlaufende Nummer	BIGINT
reportid	ID des zugehörigen Reports aus regularreport (Abschnitt 4.1.8)	BIGINT
timeincell	Zeitspanne die das Terminal in der Zelle eingebucht war in sec	INT
cellid	ID der Zelle, vom Provider vergeben.	INT

4.1.2 connects

Die Tabelle connects ist eine IP Verbindungsübersicht der GPRS-Terminals.

Name	Beschreibung	Datentyp
id (PK)	Fortlaufende Nummer	BIGINT
connectdate	Verbindungsdatum in Java_Millis	BIGINT
imei	GPRS-Terminal ID - International Mobile Equipment Identity	VARCHAR(45)
ip	IP Adresse mit der sich das GPRS-Terminal beim Server anmeldet	VARCHAR(45)
internal_ip	IP Adresse, die das GPRS-Terminal vom Provider bezogen hat	VARCHAR(45)

4.1.3 connectsfailed

In der Tabelle connectsfailed werden die fehlgeschlagenen Verbindungsversuche der GPRS-Terminals gespeichert.

Name	Beschreibung	Datentyp
id (PK)	Fortlaufende Nummer	BIGINT
connectdate	Verbindungsdatum in Java_Millis	BIGINT
ip	IP Adresse mit der sich das GPRS-Terminal beim Server anmeldet	VARCHAR(45)

4.1.4 deviceobject

Die Tabelle deviceobject enthält die Stammdaten der einzelnen GPRS-Terminals, wie etwa den APN und die IMEI.

Name	Beschreibung	Datentyp
imei (PK)	GPRS-Terminal ID - International Mobile Equipment Identity	VARCHAR(45)
alias	Aliasname für das GPRS-Terminal	VARCHAR(45)
cardid	ID der aktuell eingesetzten SIM Karte	VARCHAR(45)
lai	Aktuell verwendete LAI(Location Area Identifier)	VARCHAR(45)
phonenumber	Telefonnummer der aktuellen SIM Karte	VARCHAR(45)
midletversion	Aktuelle Softwareversion	SMALLINT
status	Aktueller Verbindungsstatus	TINYINT
firstregistered	Datum (Java_Millis) der ersten Verbindung	BIGINT
lastconnect	Datum (Java_Millis) der letzten Anmeldung	BIGINT
csqmin	Minimale Signalqualität	SMALLINT
csqmax	Maximale Signalqualität	SMALLINT
csqmean	Durschnittliche Signalqualität	SMALLINT
cellid	Aktueller Zell-ID	VARCHAR(45)
reboots	Anzahl der Reboot-Vorgänge	INT
restart	Anzahl der Restart-Vorgänge	INT
ignitions	Anzahl der Ignition-Vorgänge	INT
apn	Aktueller APN(Access Point Name)	VARCHAR(45)
imsi	Aktuelle IMSI(International Mobile Subscriber Identity)	VARCHAR(45)
missingsince	Datum (Java_Millis) seit dem das GPRS-Terminal offline ist	BIGINT
lastdata	Datum (Java_Millis) der letzten empfangen Daten	BIGINT

4.1.5 laiprovider

Die Tabelle laiprovider ist eine einfache Relation zwischen LAI und Providernamen. Sie kann von Third Party Applikationen genutzt werden, um eine benutzerfreundliche Darstellung der verwendeten LAIs zu erzeugen.

Name	Beschreibung	Datentyp
lai (PK)	LAI(Location Area Identifier)	VARCHAR(45)
provider	Name des Providers	VARCHAR(75)

4.1.6 missingdevice

Die Tabelle missingdevice enthält eine Übersicht der GPRS-Terminals die aktuell nicht mit dem Server verbunden sind.

Name	Beschreibung	Datentyp
id (PK)	Fortlaufende Nummer	BIGINT
detectiondate	Datum (Java_Millis) der Detektierung	BIGINT
imei	GPRS-Terminal ID - International Mobile Equipment Identity	VARCHAR(45)
reason	Grund der Detektierung	VARCHAR(254)

4.1.7 reboots

Die Tabelle reboots speichert detaillierte Daten über die Reboots der GPRS-Terminals.

Name	Beschreibung	Datentyp
id (PK)	Fortlaufende Nummer	BIGINT
imei	GPRS-Terminal ID - International Mobile Equipment Identity	VARCHAR(45)
reportdate	Datum (Java_Millis) des Reports	BIGINT
reb	Anzahl der Reboot-Vorgänge	INT
res	Anzahl der Restart-Vorgänge	INT
ign	Anzahl der Ignition-Vorgänge	INT
imsi	Aktuelle IMSI(International Mobile Subscriber Identity)	VARCHAR(45)
cardid	ID der aktuell eingesetzten SIM Karte	VARCHAR(45)
phonenumber	Telefonnummer der aktuellen SIM Karte	VARCHAR(45)
midletversion	Aktuelle Softwareversion	SMALLINT
apn	Aktueller APN(Access Point Name)	VARCHAR(45)
lai	Aktuell verwendete LAI(Location Area Identifier)	VARCHAR(45)

4.1.8 regularreport

Die Tabelle regularreport speichert eine erweiterte Verbindungsübersicht der einzelnen GPRS-Terminals.

Name	Beschreibung	Datentyp
id (PK)	Fortlaufende Nummer	BIGINT
deliveryDate	Verbindungsdatum in Java_Millis	BIGINT
imei	International Mobile Equipment Identity	VARCHAR(45)
minutes		SMALLINT
csqMin	Minimale Signalqualität	SMALLINT
csqMax	Maximale Signalqualität	SMALLINT
csqMean	Durschnittliche Signalqualität	SMALLINT
pingCounter	Anzahl Pings	SMALLINT
pingMin	Minimale Ping-Zeit in ms	SMALLINT
pingMax	Maximale Ping-Zeit in ms	SMALLINT
pingMean	Durchschnittliche Ping-Zeit in ms	SMALLINT
connectSuccess	Anzahl der erfolgreichen Connects	SMALLINT
connectFailed	Anzahl der fehlgeschlagenen Connects	SMALLINT
reattachmentSuccess	Anzahl der erfolgreichen GPRS-Attachments	SMALLINT
reattachmentFailed	Anzahl der fehlgeschlagenen GPRS-Attachments	SMALLINT
timeNotRegistered	Zeit ohne GSM Verbindung in ms	INT
timeHome	Verbindungszeit beim standard Provider in ms	INT
timeSearching	Zeit GSM Netzwerksuche in ms	INT
timeDenied	Zeit in der Verbindung abgelehnt wurde in ms	INT
timeRoaming	Verbindungszeit im Roaming in ms	INT
lai	Aktuell verwendete LAI(Location Area Identifier)	VARCHAR(45)
apn	Aktueller APN(Access Point Name)	VARCHAR(45)
biterror_0	Anzahl Bitfehler	SMALLINT
biterror_1	Anzahl Bitfehler	SMALLINT
biterror_2	Anzahl Bitfehler	SMALLINT
biterror_3	Anzahl Bitfehler	SMALLINT
biterror_4	Anzahl Bitfehler	SMALLINT
biterror_5	Anzahl Bitfehler	SMALLINT
biterror_6	Anzahl Bitfehler	SMALLINT
biterror_7	Anzahl Bitfehler	SMALLINT
biterror_8	Anzahl Bitfehler	SMALLINT

Literatur

[1] Java RMI. http://java.sun.com/javase/6/docs/technotes/guides/rmi/, Juli 09.

[2] Mc product catalog. http://www.mc-technologies.net/downloads/katalog.pdf, Juli 09.