MyMIDP and MyMIDP-Client: Direct Access to MySQL Databases from Cell Phones

Hagen Höpfner and Jörg Schad and Sebastian Wendland and Essam Mansour

International University in Germany
School of Information Technology
Campus 3, D-76646 Bruchsal, Germany

hoepfner@acm.org {*Joerg.Schad*|*Sebastian.Wendland*|*Essam.Mansour*}*@i-u.de*

1 Introduction and Motivation

Cell phones are no longer merely used to make phone calls or to send short or multimedia messages. They more and more become information systems clients. Recent developments in the areas of mobile computing, wireless networks and information systems provide access to data at almost every place and anytime by using this kind of lightweight mobile device. But even though mobile clients support the Java Mobile Edition or the .NET Micro Framework, most information systems for mobile clients require a middleware that handles data communication. Oracle Lite [Ora04a, Ora04c, Ora04b] and IBM's DB2 Everyplace [IBM04a, IBM04b] use a middle-ware approach for synchronizing data between client and server. Microsoft's SQL Server CE [Mic08] needs Active Sync and Sybase Adaptive Server Anywhere [Syb08] either uses SQL-Remote and its message oriented replication or MobiLink as a session based approach. All these systems are designed for handling replicated data [KRTH07] but not for simple client/server data access. In previous works [CIIH07, ICH07] we used a simple web service that forwards queries to the server and returns the result to the requesting client using an HTTP-connection. However, this approach is comparable to the middle-ware solutions and requires additional software (the web service) that might be an additional point of failure. Java's JDBC provides a standard way to access databases in Java, but this interface is missing in Java ME. In this paper we present our implementation of an MIDP-based Java ME driver [HSWM09] for MySQL similar to JDBC that allows direct communication of MIDP applications to MySQL servers without a middleware. We illustrate the usage of the driver by our prototype MySQL client for MIDP enabled mobile phones.

2 Overall Architecture

Before we started the development we set ourselves four design goals: (1) keep the driver API as near to the JDBC specification as possible, (2) keep the `.jar`-file size below 32kB

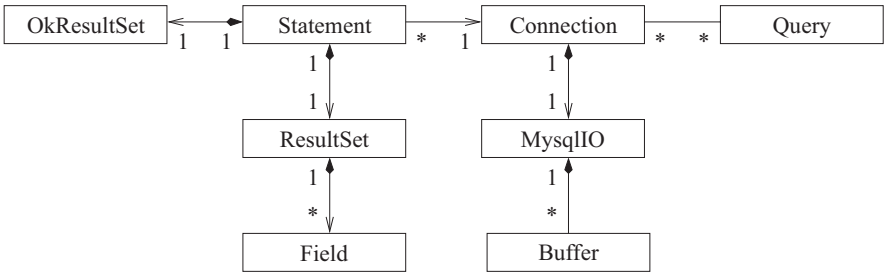

Figure 1: Class Overview

– half the popular 64kB limit for cell phones – to leave enough space for the application, (3) keep the implementation code as simple and performant as possible. These goals were mostly achieved. Our current development version provides database access sufficient for most applications in just 27kB. (In comparison, the MySQL JConnector JDBC driver has more than 500kB.) On the other hand we had to cut short on some aspects like parametrized queries and meta data usage. Figure 1 shows the basic class diagram of the driver.

The *Buffer* class is responsible for encoding and decoding packet fields as well as for the conversion between MySQL and Java data types. The *MysqlIO* class handles the communication with the database server. The *MysqlIO* class uses two *Buffer* instances, one for sending and one for receiving, to which it has exclusive access, ensuring strict task separation between the classes. The *Connection* class is very similar to the *Connection* interface of JDBC. It owns an instance of *MysqlIO* and uses it to provide connection specific methods like opening and closing a connection and changing the database. It also works as factory for *Statement* and *Query* objects. The *Statement* class is very similar to the JDBC *Statement* interface. It provides methods to execute database queries and fetch the the result. For this it implements the packet sequence logic necessary, but relies on the instance of the *MysqlIO* class hold by the *Connection* class factory for doing all packet processing. The *Query* class provides basic functionality for parametrized queries. The *OKResultSet* is an simple query information storage and is solely used by the *Statement* class. It is not directly available to the application developer but must be accessed through methods provided by the *Statement* class. The *ResultSet* class performs the same job as the JDBC *ResultSet* interface, providing exactly the same row-pointer based access methods. It uses an array of *Field* class instances to store and process all column specific data. In fact, the *ResultSet* does only act as a facade to the *Field* class, managing the row dimension of the database result set. The *Field* class provides column wise storage for database result sets and meta data. A large number of simple methods provide access to specific column- and meta data. It is only used internally by the *ResultSet*. Three additional helper classes provide a number of static methods for common tasks not really part of the driver (like string operations). The *Constants* class contains all the necessary constants. Finally, there is one *SQLException* class used throughout the driver.

3 Using MyMIDP

Every MIDP 2.0 compatible device should be able to use the driver when the following, additional requirements are fulfilled: It must support socket connections (optional in MIDP 2.1). It must support JSR 177[1] (needed for MySQL authentication via SHA-1). It should have at least one megabyte of free heap memory (depending on the implementing application).

Since the driver API is a very similar to JDBC, a developer familiar with JDBC will not have any problems using our driver. And even developers new to database APIs will find our driver easy to use as it always follows four steps: (1) Create a database connection, (2) Create and execute a database statement, (3) Process the result set, (4) Close the connection. Steps two and three can be repeated in case more than one query must be executed. For illustration purposes the following listing shows a short usage example:

```
import de.iu.db.mysql.mini.Connection;
import de.iu.db.mysql.mini.ResultSet;
import de.iu.db.mysql.mini.Statement;
import de.iu.db.mysql.mini.exceptions.SQLException;

public class Demo {

    public static void main(String[] args) {

        try {
            // connecting to database 'catsanddogs' on server test.somenetwork.net:3006, user 'test', pwd 'run'
            Connection con = new Connection("test.somenetwork.net", 3006, "test", "run", "catsanddogs");
            // retrieve some data
            Statement st = con
                .createStatement
                ("SELECT name, age, owner FROM dogs");
            ResultSet rs = st.executeQuery();
            // loop through the result set
            for (; rs.current() < rs.getResultCount(); rs.next()) {
                // access the data using row and column pointer
                System.out.println("The Dog " + rs.getAsString(0) + " ("+ rs.getAsInt(1) + ") is owned by "
                    + rs.getAsString(2));
            }
            // adding some data
            long count = st.executeUpdate("INSERT INTO dogs (name, age, owner)" + "VALUES ('Angel', 12, 'Charlie')");
            System.out.println("Added " + count + " datasets with message: "+ st.getMessage());
            // end the session
            con.close();
        } catch (SQLException e) {
            // do some error handling
            e.printStackTrace();
        }
    }
}
```

Main differences to JDBC There are a few important usage differences to JDBC we would like to point out. First, the driver does not use the JDBC style connection URL but method parameters for simplicity and performance reasons. Second, the *Statement* object can be reused thus improving garbage collection. Third, it is not possible to execute multiple statements at the same time as they use the same *MysqlIO* class instance and thus share buffers.

[1] JSR 177: Security and Trust Services API for J2ME™: `http://jcp.org/en/jsr/detail?id=177`

4 A Prototype Implementation: MyMIDP-Client

As a prove of concept we implemented a prototype client (see Figure 2) that utilizes the MyMIDP driver. After connecting to a MySQL server the client provides query templates. The user can choose between select, insert, update, and delete. As these templates only support typing in queries the client only preinitializes the query string with the staring keyword of the query. The user has to complete the query string manually but might also remove the key word provided by the templates. After submitting the query to the server, the result is displayed. Due to the limited size of the display we decided to add a unique ID to each tuple and to display the first attribute only. Then, by selecting this ID, the user can display the tuple completely.

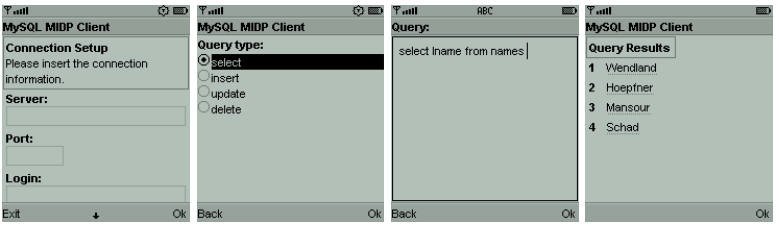

Figure 2: The MyMIDP-client in action

The MyMIDP sources and the MyMIDP-client prototype are GPL licensed and available at `http://it.i-u.de/dbis/MyMIDP`.

References

[CIIH07] Alexandru Caracaş, Iulia Ion, Mihaela Ion, and Hagen Höpfner. Towards Java-based Data Caching for Mobile Information System Clients. In Birgitta König-Ries, Franz Lehner, Rainer Malaka, and Can Türker, editors, *MMS 2007: Mobilität und mobile Informationssysteme; Proceedings of the 2nd conference of GI-Fachgruppe MMS*, volume P-104 of *LNI*, pages 97–101, Bonn, Germany, 2007. GI, Köllen Druck+Verlag GmbH.

[HSWM09] Hagen Höpfner, Jörg Schad, Sebastian Wendland, and Essam Mansour. MyMIDP: An JDBC driver for accessing MySQL from mobile devices. In *Proceedings of the 1st International Conference on Advances in Databases (DB 2009), March 1-6, 2009 - Gosier, Guadeloupe/France*. IEEE, 2009. accepted for publication.

[IBM04a] IBM Corporation. *IBM DB2 Everyplace Application and Development Guide Version 8.2*, August 2004.

[IBM04b] IBM Corporation. *IBM DB2 Everyplace Sync Server Administration Guide Version 8.2*, August 2004.

[ICH07] Iulia Ion, Alexandru Caracaş, and Hagen Höpfner. MTrainSchedule: Combining Web Services and Data Caching on Mobile Devices. *Datenbank-Spektrum*, 21:51–53, May 2007.

[KRTH07] Birgitta König-Ries, Can Türker, and Hagen Höpfner. Informationsnutzung und -verarbeitung mit mobilen Geräten – Verfügbarkeit und Konsistenz. *Datenbank-Spektrum*, 7(23):45–53, 2007. in German.

[Mic08] Microsoft Corporation. `http://msdn.microsoft.com/library/`, 2008.

[Ora04a] Oracle Corporation. *Oracle Database Lite, Administration and Deployment Guide 10g (10.0.0)*, June 2004.

[Ora04b] Oracle Corporation. *Oracle Database Lite, Developer's Guide 10g (10.0.0)*, June 2004.

[Ora04c] Oracle Corporation. *Oracle Database Lite, SQL Reference 10g (10.0.0)*, June 2004.

[Syb08] Sybase Inc. `http://www.sybase.com/ianywhere/products`, 2008.