User-Customizable Database Access Application on Smart Devices for Small Businesses

Daisuke Kikuchi  Toshiaki Ito  Masaru Tezuka
R&D department
Hitachi Solutions East Japan
{daisuke.kikuchi.hz, toshiaki.ito.tu, masaru.tezuka.fd}@hitachi-solutions.com

Abstract

In this paper, we propose DAAFS (Database Access Application For Small businesses), a user-customizable database access application for smart devices. With DAAFS, users can define both the user interface for smart device applications and the data access interface for the database servers, simply by editing XML files. As a native android application, DAAFS also has robust offline capability by default, and exhibits good performance under a variety of runtime conditions.

Keywords: User-customizable, smart phone, tablet PC, database access

1 Introduction

Modern smart devices enjoy a number of advantages over previous computing hardware, including user friendliness, versatility, availability, and low cost. These advantages have motivated an extraordinary shift in development toward smart mobile platforms and, more recently, their application to enterprise software systems. Unfortunately, while larger enterprises surge forward in the mobile space, most smaller businesses (i.e. those with fewer than 20 employees) have been deterred by the high cost of integrating new mobile clients with existing enterprise databases (for production, sales, inventory, etc.).

To lower this barrier to entry, we propose a customizable database access application for smart devices. The aim of this application is to allow users to define both the data interface of the smart device application and the connection interface of the database server using XML files. This should allow small businesses to make fuller use of mobile platforms in their enterprise systems, without the need for specialized program-

2 Requirements for Database Application of Smart Devices

We investigated a number of actual cases of mobile application development for small- and mid-sized businesses involved in housing inspection, manufacturing process management, shipping instruction, and so on.

All of the applications we examined ran on handheld terminals, i.e. specialized portable devices assigned to site workers and used to create, retrieve, update and delete (CRUD) data from proprietary databases in real-time. The common features of the user interfaces were (1) viewing data as list and (2) editing details of a selected record. The types of data varied from primitives such as strings and numbers to binary multimedia, such as digital photos and voice.

Our aim in each case was to replace the handheld terminals with smart devices, thereby cutting costs and increasing versatility. To this end, we established three distinct requirements for our smart device applications:

1) Balance among flexibility, ease of use, and cost of customization
Greater flexibility generally implies greater cost and greater complexity. Despite the greater flexibility of smart devices, our application must present a low-cost, easy-to-use substitute for handheld terminal applications.

2) Offline use capability
Since workers in the field or at production facilities often lack access to wireless networks, our mobile application must work offline as well as online.

3) Run-time performance
The computational performance of smart devices is still lower than that of PCs, but is generally better than that of handheld terminals. To
minimize user frustration, our application must preserve this performance advantage, meeting or exceeding the performance of handheld terminals across most functions.

3 Structure of DAAFS

Since our main target users are small businesses, we have chosen to name our application DAAFS (“Database Access Application For Small businesses”). Given the need to run both online and offline, DAAFS is implemented as a native smart platform (Android) application. A Web platform application was deemed unsuitable, as it could not be used in an offline environment, and would not perform as well as a natively compiled application.

3.1 Customization capability based on business case analysis

To find a good balance among flexibility, ease of use, and cost of customization, we examined the business cases of our target clients.

The essential functions of client applications in standard enterprise systems are encompassed by the CRUD (Create-Read-Update-Delete) methods on data. In our business cases, client devices were mostly used to view, store, and synchronize data managed in local clients and on server databases, so CRUD methods need to be easily callable from client applications and configurable for connection with any database, whether local or remote. Thus, we chose to limit application scope to basic CRUD functions.

3.2 System structure

The structure of DAAFS is shown in figure 1. Note that it is a multilayer architecture [1] consisting of a “DAAFS client” and a “DAAFS server.” The DAAFS client is a smart device application corresponding to the presentation layer. The DAAFS server corresponds to the data access layer, which connects to database servers in an enterprise system.

Each application has a definition file that includes a “screen definition” for generating client GUIs, and a “data definition,” for configuring interfaces for data access. Customizing the application is a matter of creating and editing these definitions.

Note that, with proper customization, a wide variety of data (files, photos, etc.) should be accessible from and manageable through the application, in both offline and online environments.

3.3 DAAFS Client

The DAAFS client generates application GUIs for smart devices based on screen definitions. These GUIs allow a user to view and store data in enterprise databases. The GUIs are designed to support all basic CRUD functions.

The core aim of the DAAFS client is to establish a one-to-one correspondence between a table in the (local/remote) database and a screen to list data in the table. We consider this the most basic “data viewer” application.

We have also designed the DAAFS client to allow definition of what and how fields in a selected record are to be shown and edited, and what and how the GUI will transition between
these modes. These capabilities should cover the fundamental use cases of database-accessing applications in an enterprise system.

3.4 DAAFS Server

The DAAFS server provides common interfaces for hosted databases. Requests for authentication to connect with databases and commit CRUD transactions from DAAFS clients are executed via RESTful (Representational State Transfer) APIs [2], exposed via HTTP or HTTPS protocols. The data access interfaces are configured according to data definitions, which can be generated automatically from existing database schemas.

To support offline use, DAAFS includes a batch downloading function, which synchronize download multiple tables from a server to a local (client-side) database or vice versa. The definition file declares what data should be synchronized and how.

Definition files are stored on the DAAFS server and sent to the DAAFS client as necessary. Thus, the client can be automatically updated when data definitions change.

3.5 Advantages of DAAFS

We identify five main benefits of DAAFS for small enterprise users of smart devices:

1) DAAFS can access existing enterprise databases.
2) Users can customize DAAFS by themselves easily.
3) DAAFS has off-line use capability.
4) DAAFS maintains good run-time performance.
5) DAAFS can utilize smart device-specific features.

Those advantages improve usefulness of smart device applications for small businesses.

4 Evaluation of DAAFS

We implemented DAAFS and assessed its effectiveness in customizing database access applications. In this section, we discuss the platform and middleware DAAFS runs on, and provide examples of GUIs and data presented by the application. We also demonstrate its customizability.

<table>
<thead>
<tr>
<th>Table 1. Middleware used in DAAFS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client</strong></td>
</tr>
<tr>
<td>OS</td>
</tr>
<tr>
<td>Application Framework</td>
</tr>
<tr>
<td>Database</td>
</tr>
</tbody>
</table>

4.1 DAAFS platform and middleware

The DAAFS system is implemented on the middleware shown in table 1.

The DAAFS client is built as an Android application using the Android SDK [3]. Client-side data is stored in a SQLite [4] database for offline use.

The DAAFS server is constructed with ASP.NET MVC Framework [5] and connects with SQL Server databases for online use.

4.2 Example of Customization

Here we provide an example customization supporting the management of orders for some commodity. Figure 2 shows a sample data definition and figure 3 a sample screen definition. Note that the XML format is quite readable; most users should have little difficulty parsing and pattern matching this format for further customization.

Based on the data definition in figure 2, DAAFS finds a database with two tables, “Orders” and “Goods”. The attribute “id” of the <Table> and <Column> elements is the nominal identifier used by the database. The table “Orders” consists of five columns: “OrderID”, “GoodsName”, “Quantity”, “UnitPrice”, and “PickUpDate”. We can define the data types of each column using its “type” attribute. Available data types include “String”, “Integer”, “Float”, “Date”, “Photo” and similar. Note that these are abstract types defined within DAAFS, and will be cast to suitable types for the given database at runtime. Whether the value is a primary key or is nullable is determined by further attributes.

We also show screen images of the Android application dynamically generated with DAAFS client from the screen definition.

Based on the screen definition in figure 3, the client application generates two GUI screens for the “Orders” table: the list view screen shown in figure 4 and the detail view screen shown in fig-
In DAAFS, each accessible table always corresponds to both a list view screen and a detail view screen.

The list view screen displays the list of records in the table. The columns to be displayed in the list are defined in the screen definition. In the example above, columns “OrderID”, “GoodsName” and “Quantity” are designated with <ViewItem> tags under the <ListView> grouping. Thus, the list view screen in figure 4 displays those three columns for the “Orders” table.

The transition screen for when a record in the list view is selected is defined as the value of the “onSelect” attribute of the <ListView> tag. In figure 3, the value for this attribute is “Detail,” indicating that the detail view screen of the corresponding table should be displayed when a record is selected. Similarly, we can transit to another list view screen for the “Goods” table by selecting the appropriate menu option, defined under the <ListViewMenu> grouping.

The detail view screen shows the detail of the associated record. The columns to be displayed on the screen (“OrderID”, “GoodsName”, “Quantity”, “UnitPrice”, and “PickUpDate”) are defined with <ViewItem> tags under the <DetailView> grouping. The values displayed can be edited using native GUI components, such as a simple text box, a Date Picker, and so on. Suitable GUI components are displayed according to each data type.

We can also refer to values in other tables when editing. In this example, <Reference table=”Goods” column=”GoodsName”> is specified in <ViewItem>. As a result, when we press the reference button which has a magnifying glass, the values of the “GoodsName” column in the table “Goods” are listed for selection. Whether this column is editable can also be expressed in the screen definition.

All GUI screens are dynamically generated based on templates, so we do not need to specify further presentation details for screens that only differ in data displayed.

4.3 Ease of customization

As a point of comparison, we can estimate the
SLOC (source lines of code) required to develop a new application equivalent to the customized DAAFS. Our estimate assumes the following:

- The application accesses one database consisting of five tables
- 10 columns for each table
- One list view screen and one detail view screen is equipped for each table
- Other five screens are provided to synchronize database, to configure settings, and so on

Conventionally, Android applications are implemented in Java. If a new application matching the above profile were developed from scratch, the scale of the project would be roughly as outlined in Table 2, requiring 6000 lines total, 3000 for the presentation layer and 3000 for the data access layer. The reason this SLOC is so large is that we have to write presentation and data access code for each screen and table, even though their implementations are similar. In terms of developer time, it would likely take a month or more to deliver such an application.

Table 3 shows the SLOC for our DAAFS configuration files. Note that we can customize the application with only 325 lines total, 250 for the presentation layer and 75 for the data access layer. This represents only a few days of developer time.

### 5 Related Works

There are a number of frameworks supporting development of database access applications. Microsoft offers Visual Studio LightSwitch[6] for rapid development of business applications that access various data sources, including SQL Server, SharePoint, and so on. The approach is almost same as in our framework, since LightSwitch uses an application definition to develop the user interfaces and data access interfaces of an application dynamically. LightSwitch offers high flexibility and ease of customization, but cannot leverage smart de-

---

**Table 2. SLOC of Java Implementation Codes**

<table>
<thead>
<tr>
<th>Layer</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation</td>
<td>3000 (Total of 15 screens, average 200 lines for each screen)</td>
</tr>
<tr>
<td>Data Access</td>
<td>3000 (2000 lines for connection with server database, and 1000 lines for connection with local database and data synchronization)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>6000</strong></td>
</tr>
</tbody>
</table>

**Table 3. SLOC of definition files**

<table>
<thead>
<tr>
<th>Layer</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation</td>
<td>250 (Total of 5 tables, average 20 lines for each list view screen, and 30 lines for each detail view screen)</td>
</tr>
<tr>
<td>Data Access</td>
<td>75 (Total of 5 tables, average 15 lines for each table)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>325</strong></td>
</tr>
</tbody>
</table>
vice-specific features such as local database and camera input.

Filemaker also offers a platform [7] for rapid development of iOS applications that access Filemaker and/or ODBC connectable databases. The applications are tuned to iOS and can leverage certain capability of iOS smart devices, and they can also run in offline environments by design. However, a Filemaker Pro server is needed to connect with existing online databases, significantly increasing development cost.

6 Conclusions

Our case analysis indicates that DAAFS would meet the needs of a wide range of small- to medium-sized businesses. Its ease of use, customizability, and robustness set it apart from existing solutions.

In future work, we plan to extend our work on DAAFS in several ways:

- **More exhaustive evaluation:** Though we have confirmed baseline reductions in application development cost, to show how effective DAAFS is in practical settings, we need to evaluate its applicability to small businesses solutions based on functional and non-functional requirements.
- **User-friendly tools for writing definition files:** We are currently developing a graphical screen editor, and plan to develop a data definition generator that makes use of database schemas or entity-relationship diagrams.
- **Adaptive GUI component selection:** We hope to improve the usability of the client by offering a larger selection of GUI components for supported data types, including multimedia types.
- **Handling business logic:** Simple use of CRUD methods through RESTful interfaces is not sufficient for modeling general business logic. We are currently investigating ways to express business logic in DAAFS using an XML format. This feature set may ultimately appear as an optional plug-in.

References