DEVELOPING A CROSS-PLATFORM MOBILE SMART METER APPLICATION USING HTML5, JQUERY MOBILE AND PHONEGAP

Alexander Zibula[1] and Tim A. Majchrzak[2]
[1]*Best Practice Consulting AG (bpc), Münster, Germany*
[2]*Department of Information Systems, University of Münster, Münster, Germany*

Keywords: App, Mobile Application, Cross-platform, iOS, Android, HTML5, PhoneGap, jQuery Mobile, Smart Meter, iPhone, Apache Cordova, Apache Callback.

Abstract: Mobile computing devices are becoming more prevalent. Mobile applications extend their scope and utility. Developing such apps introduces a problem: developers are being forced to develop their applications to target each mobile platform from scratch or to rely on Web-based solutions that do not provide a "native" look and feel. In order to investigate novel methods of multi-platform app development and to review HTML5, we built an app using several cutting-edge development frameworks. Our scenario—a smart meter tool—is an active area of research in itself. We present approaches in the field of multi-platform app development and illustrate the technological background. We provide an overview of our development process. Finally, we evaluate our app and discuss our findings in order to provide generalizable results.

1 INTRODUCTION

Since the advent of the Internet as a universal technology, mobile computing has continuously increased in importance. Beginning with smartphones, in particular the iPhone in 2007 (Macedonia, 2007), mobile Internet usage has greatly increased. For example, in Germany it rose from 9 % to 16 % from 2009 to 2010 (Mobile Internetnutzung, 2012). At the same time, mobile devices—i.e. mobile phones, smartphones, and tablet computers (*pads*)—have been equipped with increasing amounts of memory and computational power. Consequently, there is a demand for software that makes the most of these devices.

Mobile applications already generate significant revenue—USD 5.2 billion in 2010 according to Gartner (Gartner, 2012). These so called *apps* have become commonplace, even though technologically they are "ordinary" computer programs. In the simplest form, a Web site optimized for mobile devices (*Webapp*) is already considered an app. In a more dedicated form, Web-based content is rendered by a mobile device's internal browser but displayed within an app installed on it. This can offer access to device features but might not provide much more than an icon on the device's home screen, changing the users' perception. The third category are native apps that have been written for a specific platform—either a virtual machine or a mobile (device) operating system.

Developing apps is far from hassle-free. Due to rapid progress in hardware development, software development has not kept up the pace (as illustrated in Section 2).[1] To some extent it is unclear whether classical software development methods can be applied to mobile devices. Even worse, there are a number of incompatible platforms. Development of PC programs for multiple platforms is much more convenient than it used to be. A Java program runs on any system that can host an adequate virtual machine. Even programs natively developed for one platform can usually be cross-compiled for other platforms (e.g. in C++ via Gtk+ (Logan, 2001)) if they do not heavily make use of platform-dependent functionality. Traditional development for mobile apps means writing programs for each platform from scratch. Only few approaches existed to circumvent this problem.

To investigate possibilities of cross-platform development, we designed an app using HTML5 (HTML5, 2012), PhoneGap (PhoneGap, 2012), and

[1]This surprisingly aligns with what is known as the *software crisis* (Dijkstra, 1972) for stationary computers.

jQuery Mobile (jQuery Mobile, 2012). Our aims were to understand how they are used, how development differs from classical applications, and how well the apps perform. To add rigor while working on a highly relevant topic, and to broaden the scope of our research, we selected *smart metering* (Popa et al., 2010) as our scenario. We thereby developed a new kind of app that is demanding w.r.t. mobile device resource utilization.

Smart grids have been proposed as an energy grid upgrade (Javadi and Javadi, 2010) in order to more efficiently distribute and use energy, particularly in light of regenerative sources (Knab et al., 2010). Despite the need for cautions (Meehan, 2010) their emergence can be taken for granted. Smart meters are part of smart grids. They measure energy throughput and offer the possibility to better control consumption (Lee et al., 2011). For example, energy-pricing throughout a day can be adjusted by demand. An analysis of benefits from smart metering has been described by Gnilka, Meyer-Spasche & Folta (Gnilka et al., 2009). The architecture of smart meters and smart grids is complex because it involves various communication partners, communication channels, and intermediaries.

Our work makes several contributions. Firstly, it presents a novel approach for developing mobile applications against multiple platforms with a single code-base. Secondly, it introduces an innovative smart meter application. Thirdly, it discusses the findings in the light of future development projects. And fourthly, it briefly compares current approaches for cross-platform app development.

This paper is structured as follows. Section 2 outlines related work and alternative approaches. Our work's technological background is sketched in Section 3. Section 4 explains the app development. Evaluation and discussion are given in Section 5. Section 6 draws a conclusion.

2 RELATED WORK AND ALTERNATIVE APPROACHES

Related work can be discussed from various perspectives. Firstly, other approaches for developing mobile applications for multiple platforms are highlighted. Secondly, papers on HTML5 are assessed. Finally, smart meter apps are reviewed. With regard to approaches for multi-platform development, we leave out Webapps and apps that can be installed natively but only use the internal browser to render HTML 4.01 or XHTML 1.1. Even if Web pages are optimized for mobile devices, they cannot be expected to have a "native" look and feel and they are unable to use device-specific functionality.

The following five multi-platform development approaches will likely become important in the near future. Firstly, *HTML5* (HTML5, 2012) promises to combine the convenience of developing Web user interfaces (UI) with the ability to use a device's specific functionality. It is introduced in Section 3. Secondly, *hybrid apps* combine the features of HTML with native code that is provided by specialized frameworks (Barney, 2009). In general, programmers do not require detailed knowledge of the target platform but only of Web technologies. An example is PhoneGap (PhoneGap, 2012). Thirdly, *interpreted apps* can be used. Instead of writing platform-dependent code, scripting languages that are interpreted on the target platform are used. An example is Appcelerator Titanium (Appcelerator, 2012). Fourthly, with *cross-compilers* code is written once and compiled to platform-dependent apps. An example is XMLVM (XMLVM, 2012). Fifthly, *model-driven software development* (MDSD) can be applied. The idea is to describe an app in an abstract notation (the model) that is transformed and compiled to platform-dependent apps. An example is applause (Applause, 2012), which is based on a domain-specific language (DSL). Neither of the approaches can yet be seen as a definite solution. Further frameworks exist that fall in between the above sketched categories. Cross-platform app development not only is a topic of academic research but also for practitioners (Allen et al., 2010).

Although HTML5 is still considered a "working draft" (HTML5, 2012), standardization is at an advanced stage. This is also reflected by the appearance of practitioners' textbooks such as that by PILGRIM (Pilgrim, 2010). Even Webapp development for Android and iOS using HTML is covered (Oehlman and Blanc, 2011; Layon, 2010). Most scientific articles that have been published in this field do not directly relate to our work. They cover issues such as Web3D (Di Cerbo et al., 2010), accessibility (Pfeiffer and Parker, 2009), mashups (Aghaee and Pautasso, 2010), and HTML5 usage in general (Harjono et al., 2010). We did not identify scientific papers on HTML5 in the context of app development besides a paper by MELAMED & CLAYTON (Melamed and Clayton, 2009). The authors compare HTML5 to J2ME and native app development in the context of *pervasive media applications*. They find that HTML is a "good solution" for "creating and distributing most pervasive media applications" (Melamed and Clayton, 2009). Only few non-scientific sources exist that develop proof-of-concept apps (Rogers, 2010; Suhonos, 2010).

It is not feasible to review papers on smart grids or on smart meter devices. A myriad of articles exist but they are beyond the scope of this work since all facets of smart metering are addressed. From the relevant papers, some address distinctive implementation issues (e.g. (Song et al., 2011; McLaughlin et al., 2010)) or software that concerns its embedding into infrastructures (for instance (Capodieci et al., 2011)). However, these approaches have little in common with our app.

The work presented by WEISS, MATTERN, GRAML, STAAKE, & FLEISCH (Weiss et al., 2009) is similar to ours. They propose a Web-based API to access smart meters and visualize data on mobile phones. The device used appears to be an iPhone; however, WEISS et al. focus on smart metering aspects rather than on the app itself. Therefore, their work and ours are complementary. WASSERMAN (Wasserman, 2010) compiled a short article on "software engineering issues" for app development. He thereby predicts some of our findings.

Many smart meter apps can be identified that have not been developed with scientific progress in mind. We expect that several proof-of-concept apps exist that are not publicly described. Two free examples are Google PowerMeter (Google PowerMeter, 2012) and Vattenfall Smart Meter App (Vattenfall Smart Meter App, 2012). They are depicted in Figure 1. Google PowerMeter is a Webapp that allows users to view energy consumption by day, week, or month. It also provides some predictive analysis. The app is based on Web technology. It offers a lot of functionality, is well-documented, and has an open API. Unfortunately, it has been discontinued. The Vattenfall solution is a native app for iOS. Both apps are similar in concept to our work although not using any advanced technology for cross-platform development.

3 TECHNOLOGICAL BACKGROUND

HTML5 is expected to change the Web's technological landscape (Vaughan-Nichols, 2010). The term HTML5 is usually understood in a broad sense that covers the core markup specification as well as other Web standards, technologies, and APIs (Lubbers et al., 2010). This definition is also assumed in this paper even though some aspects are actually published in separate specifications[2].

Differences between HTML4 and HTML5 have been compiled into a concise document (HTML5 differences, 2012). HTML5 features a new short document type definition to trigger standard rendering mode as well as a shorter charset definition as shown in Listing 1. It adds, changes, and removes several elements and attributes, including new semantic sectioning elements such as `<header>`, `<footer>`, `<section>`, `<article>`, `<nav>`, and `<figure>` as well as custom data (`data-*`) attributes. Furthermore, it specifies advanced forms with form validation and several new input types including `tel`, `search`, `url`, `email`, `number`, `range`, `color`, `date` and `time` (HTML5 differences, 2012).

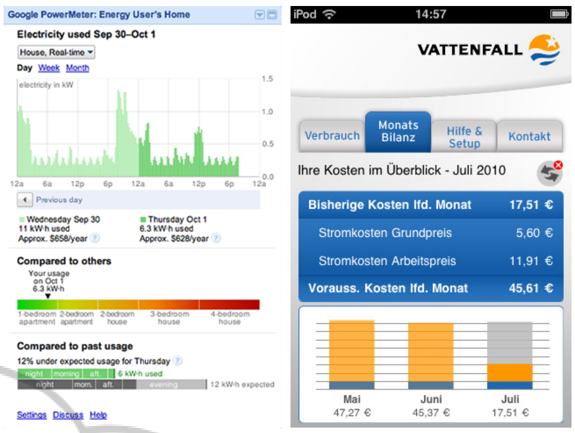


Figure 1: Google PowerMeter (Google PowerMeter, 2012) and Vattenfall Smart Meter App (Vattenfall Smart Meter App, 2012).

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
```

Listing 1: Document type and charset definition (HTML5 differences, 2012).

Classic Web applications lacked persistent, client side storage with sufficient size. Cookies and proprietary solutions like IE UserData or Adobe Flash Local Shared Objects have severe limitations. With HTML5, there are five specifications for new storage APIs: Web Storage, Indexed Database API, File API, Web SQL Database, and Offline Web applications. HTML5 also allows device access e.g. through the Geolocation API and provides further new functions (Device APIs WG, 2012).

Additional new technologies (Lubbers et al., 2010; Pilgrim, 2010) include WebSockets, which bring high performing, bi-directional, full-duplex

[2]Please note that we name several APIs that HTML5 relies on. We do not cite their specification since there are links in the HTML5 specification (HTML5, 2012) and detailed coverage is outside the scope of this work.

TCP connections; Cross Document Messaging, which enables secure cross-origin communication; and asynchronous JavaScript, which becomes possible through the Web Workers API. A highly discussed part of HTML5 concerns multimedia and graphics with the HTML5 tags `canvas`, `video`, `audio`, and the integration of SVG into HTML documents. A series of new CSS3 module specifications define new styling features such as media queries, selectors, backgrounds and borders, as well as transformations.

Browser support for HTML5 is becoming more prevalent but is still far from complete (HTML5 Test Suite, 2012; The HTML5 test, 2012). The recommended way for finding out if a certain HTML5 feature can be used is testing every needed feature instead of general user agent (browser) detection. Details are given by PILGRIM (Pilgrim, 2010). There is also a JavaScript library called *Modernizr* that is useful for this task (Lubbers et al., 2010).

JavaScript is an object-oriented, interpreted, and weakly-typed programming language (Crockford, 2008). It has become specifically important due to the proliferation of Ajax (Asynchronous JavaScript and XML). Framework support greatly improves development. For example, *jQuery* is a concise, open source, cross-browser JavaScript framework that simplifies development (Steyer, 2010). Its current use is predominant: more than 54 % of the 10.000 most visited websites (jQuery Usage Trends, 2012) employ it. The corresponding mobile-optimized plug-in is *jQuery Mobile* (Reid, 2011; Firtman, 2011).

JSON (JavaScript Object Notation) is a simple, text-based, human-readable data-interchange format. It is language independent but based on a subset of JavaScript (Crockford, 2008). JSON is used as a lightweight alternative to XML, e.g. in conjunction with Ajax or RESTful Web services (Richardson and Ruby, 2007; Webber et al., 2010).

Android is an open operating system and mobile device platform based on a modified Linux kernel. The main development language is Java (Ableson et al., 2009). Apps can not only be loaded from the *Android Market* but also from others sources.

iOS is Apple's system for its mobile devices. It has a layered system architecture; programming is done in Objective-C (Neuburg, 2011). For practical development and testing, there is no alternative to a Mac running Xcode and iOS SDK. Current releases of Android and iOS support HTML5 based on the WebKit (WebKit, 2012) layout engine.

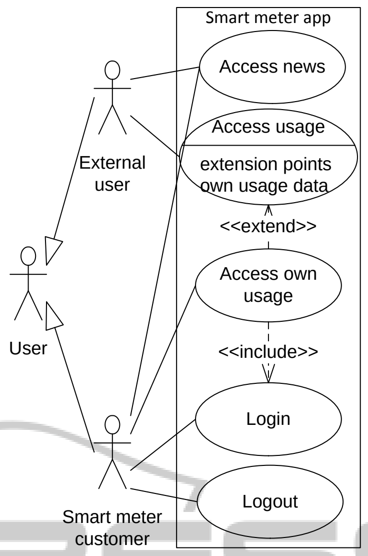


Figure 2: Use case diagram for our prototype.

4 DEVELOPMENT OF THE APP

In the following sections, we describe the development of the prototype by sketching the requirements, explaining the specification, and highlighting some implementation details.

4.1 Requirements Analysis

The basic idea for the project originates from an energy provider's request to *best practice consulting AG (bpc)*. However, requirements for our prototype were developed independently. The targeted case is the fictive energy providing company *EnergyXY* that has a substantial smart meter installation base and already runs a customer self-service smart meter Web portal. This portal is integrated with a comprehensive functional backend, such as *SAP for Utilities* (IS-U) (Frederick and Zierau, 2011).

The primary objective of the app is to enable customers to access their utility consumption and thereby provide cost and usage benefits, insights on consumption, environmental protection, and energy efficiency. Further goals concern cross-platform support, usability, and technology evaluation. Platforms that have to be supported are Android and iOS. However, easy extension to other mobile platforms such as Blackberry or Windows Phone should be possible.

In terms of usability, the app has to address specific mobile challenges such as users' attention span, smaller displays of various sizes and resolutions, and distinctive user interfaces (e.g. touchscreens) (Bieh,

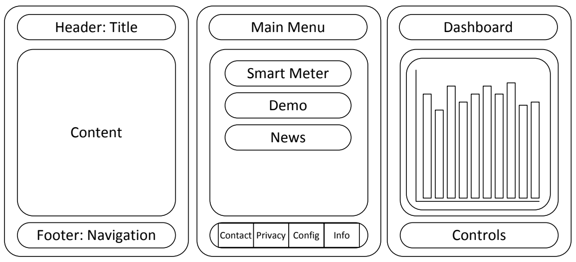


Figure 3: GUI illustrations.

2008). One important requirement is to sensibly support device orientation changes. Furthermore, the app should comply with standard UI guidelines for the target platforms such as the "iOS Human Interface Guidelines" (iOS Human Interface Guidelines, 2012) and "Android Design" (Android Design, 2012). Fulfilling these requirements covers a majority of mobile usability issues. However, these guidelines focus native apps for the specific platform and partly contradict each other, e.g. considering the usage of top and bottom bars.

The identified use cases for the app are summarized in Figure 2. The app should enable access to consumption data and news provided by EnergyXY. It should also provide a *demo mode* with random (but plausible) data. According to the use cases, a data model and a first GUI sketch were created. The latter is shown in Figure 3.

4.2 Design

The high-level sketch of the application architecture is shown in Figure 4. There is no direct connection between the smartphone and the smart meter assumed. Please note that there actually may be multiple (intermediary) information systems on the right hand side. The internal architecture of the app follows the hybrid approach, falling in between a native app and a Webapp. Native apps offer amenities in terms of device feature access, performance, marketing, and sales advantages through *app stores*. Webapps are beneficial in terms of portability, development costs, asset specificity, and development cycle and update cycle time (Stark, 2010; Barney, 2009).

Two trends decrease the differences between the two alternatives and make it possible to combine advantages of both. Firstly, new Web technologies such as HTML5 provide features that enable offline applications, increase performance, enable device access, and provide user interfaces (UI) that come close to those of native apps. Secondly, it is possible to embed a Web application into a native application, thus forming a hybrid application. These are also known as *embedded apps*, which internally use the browser to render Web content but provide access to device features (Barney, 2009). We chose a hybrid approach as it combines the advantages and balances the features of both alternatives. Such apps can be distributed via online market stores and are portable in nature.

The development of hybrid applications can be dramatically simplified with the use of frameworks such as *PhoneGap* (PhoneGap, 2012), *Rhodes* (Rhodes, 2012), or *Appcelerator Titanium* (Appcelerator, 2012). All of these are free and support at least Android and iOS. They require knowledge of HTML, JavaScript, and CSS (as well as Ruby in case of Rhodes); knowledge of native device programming is not necessary. PhoneGap was chosen due to having greater platform support (including iOS, Android, Blackberry, WebOS, Windows Phone, Symbian, and Bada) (PhoneGap Supported Features, 2012), an active development community, a comprehensive API of exposed native features, and good documentation (cf. (Stark, 2010; PhoneGap, 2012)). PhoneGap is gaining attention with dedicated books appearing in late 2011 and early 2012 (Ghatol and Patel, 2012; Lunny, 2011; Myer, 2011). It has recently been contributed as an Apache project under the name *Apache Cordova* (Apache Cordova, 2012) (originally *Apache Callback*).

The design of the UI was mainly influenced by choosing a framework that supports interfaces for mobile Webapps *and* hybrid apps. We compared *jQuery Mobile* (jQuery Mobile, 2012) and *Sencha Touch* (Sencha Touch, 2012). jQuery Mobile is a jQuery plug-in and focuses Web UI programming through semantic markup (HTML) for a wide range of platforms (iOS, Android, Windows Phone, WebOS, Blackberry, Meego, and Symbian (Mobile Graded Browser Support, 2012). In contrast, UI definition in Sencha Touch is soley done in JavaScript, without a single manually written HTML element within the document body. Platform support is limited to iOS, Android and Blackberry. We chose jQuery Mobile because it is lightweight, allows basic Web frontend programming, and supports more platforms. It offers several UI components that are optimized for touch interaction on a multitude of screens and devices. Examples include toolbars at the top or bottom of a page, various buttons and other form elements, and list views. Moreover, it provides a comprehensive page, dialog, and navigation model as well as various animations.

In Figure 5, screen sketches of the application are depicted. They include the main menu and the dashboard with a yearly and a monthly chart. In the two dashboard views, different options for the selection

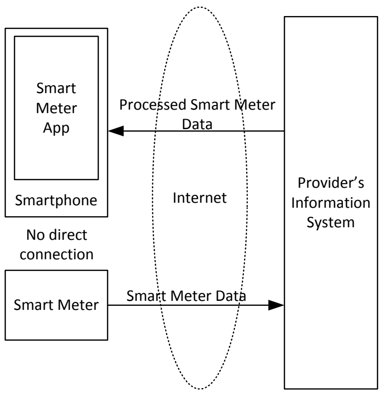


Figure 4: Simple smart meter architecture.

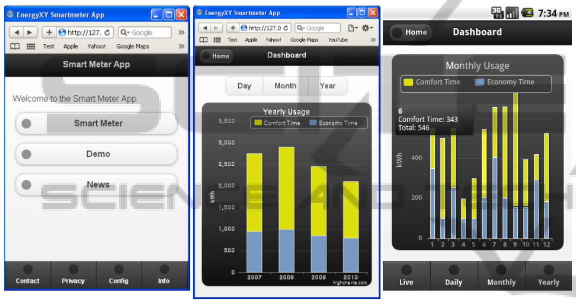


Figure 5: Screen sketches.

of the time interval were compared. The second solution with the navigation bar at the bottom was favored because of its visual consistency with the main menu. Possible navigation paths are visualized in a click graph (Figure 6).

For the generation of charts, we compared three different JavaScript charting libraries: *jqPlot* (jqPlot, 2012), *Highcharts* (Highcharts JS, 2012), and *gRaphaël* (gRaphaël, 2012). Highcharts was chosen because is provides highly interactive charts including animation during chart construction, click event handling, zooming, and mouseover events (e.g. for tooltips). It offers various chart types, a comprehensive API, good documentation and easily integrates with jQuery (Highcharts JS, 2012).

To improve accessibility, the app uses *progressive enhancement*, which is also employed in jQuery Mobile (Parker, 2010). This approach starts with a basic version with simple markup that provides a functional experience that is compatible with most browsers, platforms, and devices. Then, layers that enhance the functional experience are progressively added. The advantages of progressive enhancement are universal access, cleaner and more modular code, a single code base, and compatibility (Parker, 2010).

In order to realize loosely coupled, interoperable, and reusable communication between the smart meter app and the backend, Web services are generally suitable. However, rather than SOAP-based services, our app uses REST HTTP services (Fielding, 2000; Webber et al., 2010; Richardson and Ruby, 2007) that limit the available operations to the HTTP protocol. They offer advantages in terms of performance and simplicity. JSON as the data format offers similar advantages over XML. It considerably eases parsing to JavaScript objects and has minimal overhead. Both decisions are well suited to the small scale of the developed app. JSON Schema (JSON Schema, 2012) was used to define the interface. Instead of using an annotation-based approach found in JAX-RS implementations (Java API for RESTful Web Services (Burke, 2010)) and Spring MVC 3 (Walls, 2011), a simple Java backend application using hand-written servlets is sufficient. It provides services for login as well as retrieval of smart meter data and news.

Smart meter data is stored on the device. Out of the five possibilities introduced in Section 3, Web SQL is discontinued (Web SQL Database, 2012) and Indexed Database API is not yet supported on iOS and Android. We selected Web storage in the form of local storage, which is a simple key value store in the browser with a typical maximum size of 5 MB (increasable upon request) (Pilgrim, 2010). It is suitable for JSON formatted data conforming to the defined JSON schema.

4.3 Implementation

For the first prototype Android and iOS were selected as target platforms. Regarding the environment a Mac with Xcode and iOS SDK was used for iOS; for Android, the Android SDK was utilized along with the Android Developer Tools and Eclipse. In Listing 2, loading of the front page of the app is defined. In non-PhoneGap Android applications, the activity would simply inherit from the `Activity` class. In case of a PhoneGap app, it inherits from the PhoneGap provided class `DroidGap`. Invoking `onCreate()` loads the entry page of the Webapp. This is the only native Java Code that needs to be changed for running a PhoneGap application on Android. The remainder of the logic of PhoneGap—in particular the interfaces and implementations for providing device features—is bundled with the Java archive `phonegap.jar` and the JavaScript source `phonegap.0.9.5.js`. Furthermore, some changes will need to be applied to `AndroidManifest.xml`, concerning metadata and permissions. For the iOS application, the only change necessary was the definition of the supported interface orientations.

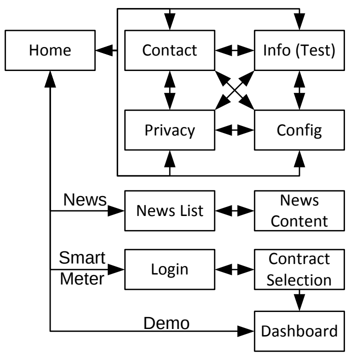


Figure 6: Click graph for our prototype.

```
public class SmartMeterActivity extends DroidGap {
  // Called when the activity is first created.
  @Override
  public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    super.loadUrl("file:////www/index.html");
  }
}
```

Listing 2: Loading of the front page of the Android app (PhoneGap Get Started Guide, 2012).

The Webapp is independent of the platform; the same code was used for Android and iOS. It uses a single HTML file with multiple internal pages that are part of jQuery Mobile's page model. jQuery Mobile handles dynamic *show* and *hide* of multiple pages via JavaScript and CSS. Pages requested via Ajax are instantly available. The markup for an internal page are nested `<div>` tags with `data-role` attributes `page`, `header`, `content` and `footer`.

The implemented internal pages correspond to the GUI Model and are connected as shown in Figure 6. jQuery Mobile uses various other custom data attributes for defining the user interface, such as dialogs, links, transitions, list views, and navigation bars. The JavaScript code has to wait for PhoneGap to complete loading before making API calls to it. The respective event is called `deviceready`. Afterwards, native features can be accessed via the PhoneGap API (PhoneGap API Reference, 2012). An example of using *vibration* is given in Listing 3. The click event on the button with the identifier `btnVibrate` is caught and the PhoneGap function `navigator.notification.vibrate` is called. The parameter defines the vibration length in milliseconds.

```
$('#btnVibrate').click(function() {
  navigator.notification.vibrate(1000);
});
```

Listing 3: Example for using vibration.

jQuery Mobile detects device orientation changes and redraws the page using the `orientationchange` event. The earlier definitions of supported screen sizes for Android and iOS merely enabled the possibility of orientation changes but not the correct adaption of the content.

In order to load and display smart meter data, a `SmartMeterDataService` generates a statistical distribution of smart meter readings, aggregates them to the requested time scale, converts to the requested type and unit (e.g. consumption in kWh or CO_2 pollution in kg) and finally returns a JSON String to the responsible servlet. The mobile application requests this data through the jQuery Ajax function `$.ajax(...)`, stores it via `window.localStorage.setItem(...)`, and parses it to a JavaScript object via `$.parseJSON(...)` so that Highcharts is able to process and draw it.

5 EVALUATION AND DISCUSSION

To reflect our work, we discuss our findings, their implications, and current limitations.

5.1 Considering the App

The home screen of the app is used as the starting point for navigation (see Figure 6). jQuery Mobile's dialogs were used for notification in case of errors, for example due to a missing network connection during login, contract selection, or loading of new data. All other operations are available offline. The login screen uses an HTML5 email form field for which iOS provides an adapted conveniently usable on-screen keyboard with direct availability of the @ and the *dot* key. On the news screen, current news are displayed in a scrollable list view.

Figure 7 shows the dashboard of the app in landscape mode on the Android emulator (left) and the iPhone 4 simulator (right). Android shows data aggregated to years, differentiated between daytime (yellow) and nighttime (blue). Clicking columns yields a detailed time scale. Tooltips appear if a *touch and hold* gesture is performed on a column. The legend allows click events to show or hide a time zone using an animation. The iPhone shows a live view, which updates every two seconds. For a live view in a productive app, direct connection to the meter would be required; we thus used simulated data (cf. Figure 4).

The app satisfies the main aim of giving customers feedback on consumption. It runs cross-platform on Android and iOS. Extension to other platforms is

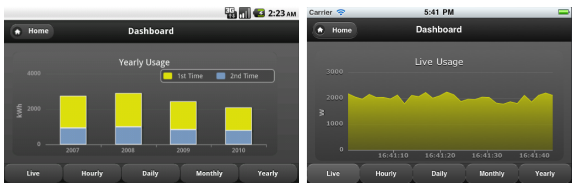
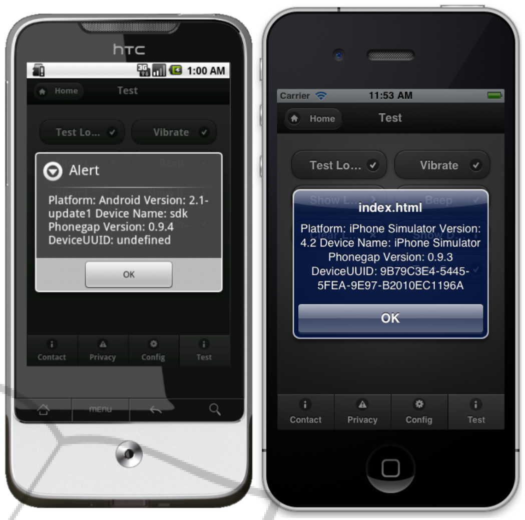


Figure 7: Comparing the Android and iOS interfaces.

possible due to the compatibility of PhoneGap and jQuery Mobile. Usability requirements were largely met. The app adapts to different screen sizes and resolutions regarding e.g. button sizes, device orientation, and animations. Moreover, it effectively utilizes the touchscreen.

The cross-platform development has highlighted the relevance of continuous testing against all platforms for several reasons: having a single code base for the embedded Webapp, the fact that bugs might be present or reproducible on only one platform, the observation that (framework) patches lead to ambiguous effects, and the low maturity of jQuery Mobile at the beginning of development (versions Alpha 2 to Alpha 4.1). Manual testing was performed on the Android emulator (versions 2.1 to 2.3.3 with various screen resolutions), two physical Android devices (HTC Desire HD running Android 2.2, HTC Legend running Android 2.1), the iOS simulator 4.2, an iPhone 4 running iOS 4.2.1, and various desktop browsers on Windows XP and Mac OS X 10.6. In addition to manual testing, Selenium (Selenium, 2012) was used for automated frontend testing. Debugging was done using Android Dalvik Debug Monitor Service (DDMS), Xcode Instruments, Firebug (Firebug, 2012), Chrome and Safari built-in developer tools, and the iPhone simulator Web debugging console. These tools were very useful to us; however, an elaborate discussion is outside the scope of this work.

Problems encountered during development concerned stability, performance, and UI differences. A special problem was the fixed positioning of the lower navigation bar, which is a prominent feature for native (looking) apps, especially on iOS. The jQuery Mobile implementation had severe bugs, which were mostly fixed in subsequent versions. The navigation bar was either shown multiple times, displaced, or did not stay at a fixed position while scrolling. Main performance issues were memory usage and processing speed of animations. Extensive memory usage of Highcharts could partly be countered against by manual freeing operations. It seems that Highcharts' resource consumption is designed primarily for PCs. Performance on the Android emulator was unsatisfactory. This did not vary noticeably with the SDK version or as-


Alert

Platform: Android Version: 2.1-update1 Device Name: sdk Phonegap Version: 0.9.4 DeviceUUID: undefined

index.html

Platform: iPhone Simulator Version: 4.2 Device Name: iPhone Simulator Phonegap Version: 0.9.3 DeviceUUID: 9B79C3E4-5445-5FEA-9E97-B2010EC1196A

Figure 8: Android and iOS native popup windows.

signed main memory size of the emulator. The performance test on the HTC Legend was acceptable; performance on the HTC Desire HD was considerably better. Updates of jQuery Mobile up to version Alpha 4.1 showed noticeable improvements in all configurations (not limited to Android).

In the iOS implementation, the device orientation change is shown with a smoother transition than on Android. In fact, the iPhone 4 had a general performance advantage, even comparing it to the faster Android device. Performance was even better in the iOS simulator, which does *not* emulate the iPhone hardware. As expected, best results were seen on desktop browsers. However, these were not a development target and were only tested for the sake of comparison.

5.2 Evaluation of the Design

The decision for a hybrid application was made for reasons including (1) device access, (2) evaluation of new technologies, (3) portability, (4) offline availability, and (5) access to online market stores. During the design phase of the app it turned out that only few device features (1) were actually necessary. Both versions use device orientation change, retrieve device information, and show native notifications. The latter are either audible (beeps), tactile (vibrations), or visual (popup windows). However, they were only used for testing and demonstration. An example is the visual native notification of device information loaded by a call to the PhoneGap API (Figure 8).

Use of new technologies (2) and portability (3) are achievable without PhoneGap, solely by means of HTML5 and jQuery Mobile. Offline availability (4) would also be possible with HTML5 although slightly more complex. The majority of the features of PhoneGap were not used because they were not required by the app. This means that a mobile Webapp using the HTML5 offline cache may have also sufficed.

An important remaining question is to weigh the advantages of the marketing and distribution opportunities of the app stores (5) against the disadvantages of additional development complexity and the time and costs of registration as well as review processes. For a non-free application, the decision is simple: a hybrid app would be preferable because setting up a payment system that is accepted by a considerable number of customers is difficult. At the moment, smart meter applications are usually offered for free (cf. Section 2). They are financed through earnings from the smart metering product. Therefore, currently a Webapp is sufficient. However, an extension of scope to advanced topics such as smart homes, smart grids, e-mobility, or even augmented reality could profit from using additional device features such as cameras or other additional sensors.

One aim of HTML5 and Webapps is to make device features available and reduce the need for native apps. Consequently, frameworks such as PhoneGap are transitional technologies but likely to ultimately become obsolete. To a high extent, HTML5 is already usable. Many functions are stable. Local storage, custom data attributes, new form elements, and canvas (through Highcharts) were used in the implementation. However, due to incomplete universal support for many features, it is not seen as a general solution, yet (Mobile Web Metrics Report H2/2011, 2011).

Native apps will always perform better in comparison to Webapps. However, it is unclear whether these differences will continue to be relevant for the majority of apps. For performance intensive applications such as 3D games, there will likely continue to be certain performance considerations. Most others will be realizable using Webapps. POGUE (Pogue, 2007) argues that mobile Webapps are only a bridge technology, i.e. to bridge the gap after the release of the iPhone until the opening of Apple's app store. He predicted that they would vanish again (Pogue, 2007). We do not share this estimation. Strong evidence for the success of standards can be seen in the success of Web applications on non-mobile devices.

5.3 Discussion of Frameworks Decisions

The decisions for PhoneGap and jQuery Mobile proved to be reasonable except for some stability issues due to limited maturity (especially for the latter). By now, PhoneGap 1.4.1 and jQuery Mobile 1.0.1 are available, offering stability and performance improvements, new features, and expanded platform support.

jQuery was mainly used for event handling and Ajax communication, where it performed well. However, the release of jQuery 1.5 with a rewritten Ajax implementation broke jQuery Mobile's Ajax navigation when combined with PhoneGap. This was rapidly fixed with version 1.5.1. A problem with Highcharts on Android (*SVG* compatibility) could be solved by falling back to an older release (Highcharts 1), which uses HTML5 canvas.

5.4 Limitations

Due to the novelty of our research and the use of many technologies that have not yet been assessed in a scientific context, our work has limitations. The backend did not yet use real data from an IS-U system but used simulated data. Connecting with an IS-U backend or a third party meter data system using e.g. Web services is a feasible task, though. More detailed and rigorous evaluation of the usability and the efficiency of the application is required. A comparative implementation with Sencha Touch or in pure HTML5 without any framework usage would be insightful. Nevertheless, neither of the limitations detracts the significance of our findings and the recommendations made.

6 CONCLUSIONS AND FUTURE WORK

We presented work on an app that was developed for multiple platforms. While many new technologies have emerged and several ways of developing hybrid apps have been proposed, there is no *universal* solution, yet. Consequently, we did not identify a large body of work that was directly related to our scenario, a smart meter app. Development of our prototype using PhoneGap, jQuery Mobile, and HTML5 has been described in detail. We presented notable insights about this way of building hybrid apps. In conclusion, our choice of technology is viable and advisable for similar projects. However, we expect rapid progress and predict that many of the current approaches will be succeeded by standardized technology. At the same time, new frameworks might enable hybrid apps in currently unsupported fields.

The novelty of the used technologies underlines a need for future work. We need to keep track of the development and keep pace with emerging mobile Web technologies. Future questions do not only concern the technical dimension. Usability and value for users also need to be assessed. It needs to be investigated how apps will be used in the future, and how the business plans for app development could look like.

Future work could comprise extension to more platforms, inclusion of additional features, and more detailed evaluations. Eventually, testing should be executed by actual users. Despite being only an example in this work, research on smart metering is a very interesting area, and yet demanding. Our work will include the evaluation of other frameworks and approaches for hybrid development. We intend to become able to provide project-specific advice on technology and framework selection.

ACKNOWLEDGEMENTS

We would like to thank Marc Fischer and Florian Blenkle from best practice consulting AG for inspiring this paper's topic, for professional support, and for helpful discussions. Furthermore, we are grateful for stylistic suggestions made by Tim Kane and Veronika Hassel.

REFERENCES

Ableson, F., Collins, C., and Sen, R. (2009). *Unlocking Android*. Manning, Greenwich, CT, USA.

Aghaee, S. and Pautasso, C. (2010). Mashup development with HTML5. In *Proc. Mashups '09/'10*, pages 10:1–10:8, New York, NY, USA. ACM.

Allen, S., Graupera, V., and Lundrigan, L. (2010). *Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution*. Apress.

Android Design (2012). "Android Design".

Apache Cordova (2012). Apache Cordova.

Appcelerator (2012). Appcelerator.

Applause (2012). applause.

Barney, L. S. (2009). *Developing Hybrid Applications for the iPhone*. Addison-Wesley.

Bieh, M. (2008). *Mobiles Webdesign*. Galileo Press.

Burke, B. (2010). *RESTful Java with JAX-RS*. O'Reilly.

Capodieci, N., Pagani, G. A., Cabri, G., and Aiello, M. (2011). Smart meter aware domestic energy trading agents. In *Proc. IEEMC '11*, pages 1–10, New York, NY, USA. ACM.

Crockford, D. (2008). *JavaScript: The Good Parts*. O'Reilly.

Device APIs WG (2012). "Device APIs Working Group".

Di Cerbo, F., Dodero, G., and Papaleo, L. (2010). Integrating a Web3D interface into an e-learning platform. In *Proc. Web3D '10*, pages 83–92, New York, NY, USA. ACM.

Dijkstra, E. W. (1972). The Humble Programmer. *Communications of the ACM*, 15:859–866.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.

Firebug (2012). Firebug.

Firtman, M. (2011). *jQuery Mobile: Up and Running*. O'Reilly.

Frederick, J. and Zierau, T. (2011). *SAP for Utilities*. SAP Press, Bonn.

Gartner (2012). "Gartner Says Worldwide Mobile Application Store Revenue Forecast to Surpass $15 Billion in 2011".

Ghatol, R. and Patel, Y. (2012). *Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5*. Apress.

Gnilka, A., Meyer-Spasche, J., and Folta, N. (2009). *Smart Metering*. LBD-Beratungsgesellschaft.

Google PowerMeter (2012). "Google PowerMeter's first device partner".

gRaphaël (2012). gRaphaël.

Harjono, J., Ng, G., Kong, D., and Lo, J. (2010). Building smarter web applications with HTML5. In *Proc. CASCON '10*, pages 402–403, New York, NY, USA. ACM.

Highcharts JS (2012). Highcharts JS.

HTML5 (2012). HTML5.

HTML5 differences (2012). "HTML5 differences from HTML4".

HTML5 Test Suite (2012). "HTML5 Test Suite Conformance Results".

iOS Human Interface Guidelines (2012). "iOS Human Interface Guidelines".

Javadi, S. and Javadi, S. (2010). Steps to smart grid realization. In *Proc. CEA'10*, pages 223–228, Stevens Point, Wisconsin, USA. WSEAS.

jqPlot (2012). jqPlot.

jQuery Mobile (2012). jQuery Mobile.

jQuery Usage Trends (2012). "jQuery Usage Trends".

JSON Schema (2012). JSON Schema.

Knab, S., Strunz, K., and Lehmann, H. (2010). *Smart grid*. TU Berlin.

Layon, K. (2010). *The Web Designer's Guide to iOS Apps*. New Riders Pub., Thousand Oaks, CA, USA.

Lee, J., Park, G.-L., Kim, S.-W., Kim, H.-J., and Sung, C. O. (2011). Power consumption scheduling for peak load reduction in smart grid homes. In *Proc. SAC '11*, pages 584–588, New York, NY, USA. ACM.

Logan, S. (2001). *Gtk+ Programming in C*. Prentice Hall, Upper Saddle River, NJ, USA.

Lubbers, P., Albers, B., Smith, R., and Salim, F. (2010). *Pro HTML5 Programming*. Apress.

Lunny, A. (2011). *PhoneGap Beginner's Guide*. Packt Pub.

Macedonia, M. (2007). iPhones Target the Tech Elite. *Computer*, 40:94–95.

McLaughlin, S., Podkuiko, D., Delozier, A., Miadzvezhanka, S., and McDaniel, P. (2010). Embedded firmware diversity for smart electric meters. In *Proc. HotSec'10*, pages 1–8, Berkeley, CA, USA. USENIX Association.

Meehan, E. (2010). The smart grid: the smart choice? In *Proc. InfoSecCD '10*, pages 173–176, New York, NY, USA. ACM.

Melamed, T. and Clayton, B. J. C. (2009). A Comparative Evaluation of HTML5 as a Pervasive Media Platform. In *Proc. 1st Int. ICST Conf. MobiCASE*, pages 307–325. Springer.

Mobile Graded Browser Support (2012). "Mobile Graded Browser Support".

Mobile Internetnutzung (2012). "Mobile Internetnutzung über das Handy 2010 stark gestiegen".

Mobile Web Metrics Report H2/2011 (2011). "mobile web metrics report h2/2011".

Myer, T. (2011). *Beginning PhoneGap*. Wrox.

Neuburg, M. (2011). *Programming iOS 4*. O'Reilly.

Oehlman, D. and Blanc, S. (2011). *Pro Android Web Apps: Develop for Android using HTML5, CSS3 & JavaScript*. Apress.

Parker, T. (2010). *Designing with Progressive Enhancement*. New Riders.

Pfeiffer, S. and Parker, C. (2009). Accessibility for the HTML5 <video> element. In *Proc. W4A '09*, pages 98–100, New York, NY, USA. ACM.

PhoneGap (2012). PhoneGap.

PhoneGap API Reference (2012). "PhoneGap API Reference".

PhoneGap Get Started Guide (2012). "PhoneGap Get Started Guide".

PhoneGap Supported Features (2012). "PhoneGap Supported Features".

Pilgrim, M. (2010). *HTML5: Up and Running*. O'Reilly.

Pogue, D. (2007). *iPhone: the Missing Manual*. O'Reilly.

Popa, M., Ciocarlie, H., Popa, A. S., and Racz, M. B. (2010). Smart metering for monitoring domestic utilities. In *Proc. INES'10*, pages 43–48. IEEE Press.

Reid, J. (2011). *jQuery Mobile*. O'Reilly.

Rhodes (2012). Rhodes.

Richardson, L. and Ruby, S. (2007). *Restful Web Services*. O'Reilly.

Rogers, R. (2010). Developing portable mobile web applications. *Linux J.*, 2010.

Selenium (2012). Selenium.

Sencha Touch (2012). Sencha Touch.

Song, K., Seo, D., Park, H., Lee, H., and Perrig, A. (2011). OMAP: One-Way Memory Attestation Protocol for Smart Meters. In *Proc. ISPAW '11*, pages 111–118, Washington, DC, USA. IEEE CS.

Stark, J. (2010). *Building Android Apps with HTML, CSS, and JavaScript*. O'Reilly.

Steyer, R. (2010). *jQuery*. Addison-Wesley.

Suhonos, M. J. (2010). Building a Location-aware Mobile Search Application with Z39.50 and HTML5. *Code4Lib*.

The HTML5 test (2012). "The HTML5 test".

Vattenfall Smart Meter App (2012). "Vattenfall Smart Meter App".

Vaughan-Nichols, S. J. (2010). Will HTML 5 Restandardize the Web? *Computer*, 43(4):13–15.

Walls, C. (2011). *Spring in Action*. Manning.

Wasserman, A. I. (2010). Software engineering issues for mobile application development. In *Proc. FoSER '10*, pages 397–400, New York, NY, USA. ACM.

Web SQL Database (2012). Web SQL Database.

Webber, J., Parastatidis, S., and Robinson, I. (2010). *REST in Practice*. O'Reilly.

WebKit (2012). WebKit.

Weiss, M., Mattern, F., Graml, T., Staake, T., and Fleisch, E. (2009). Handy feedback: connecting smart meters with mobile phones. In *Proc. MUM '09*, pages 1–4, New York, NY, USA. ACM.

XMLVM (2012). XMLVM.