ClayUI: A Framework for Delivering Object Properties to Native Mobile Application Components

Andrew Lize
Marquette University

CLAYUI: A FRAMEWORK FOR DELIVERING OBJECT PROPERTIES TO NATIVE MOBILE APPLICATION COMPONENTS

by

Andrew Lize

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

August 2012

ABSTRACT
CLAYUI: A FRAMEWORK FOR DELIVERING OBJECT PROPERTIES TO NATIVE MOBILE APPLICATION COMPONENTS

Andrew Lize

Marquette University, 2012

As technology advances in the field of mobile computing with smartphones and tablet computers becoming less expensive, people are adopting these devices into their daily lives. Due to this adoption, many developers are finding opportunities to develop apps that target this growing form factor. One of the issues that developers come across when developing for these multiple platforms is that there is a need to redesign common elements of their applications for each of their platforms. They are also challenged with the decision of breaking the prescribed design guidelines for each of the platforms they are developing for so that they are able to provide support for these applications in the future.

In this thesis, we propose a solution to this problem by generalizing common user interface elements and configure them outside of the application. Our solution, called ClayUI, uses a client server model to house and publish user interface elements to a mobile application using an API that is written in the target platform's native programming language. Our solution allows a developer to create a mobile application that adheres to the platform's design guidelines with the flexibility of being able to port it to other platforms without having to do a full redesign of the application. Our solution also introduces features that assist the developer with the process of creating local and remote database storage for the configured elements.

ACKNOWLEDGMENTS

Andrew Lize

For his support and encouragement to complete this thesis, I would like to thank Dr. Sheikh Iqbal Ahamed. Without his inspiration, I was on the fence about whether or not to attempt this goal. Dr. Ahamed has always been a source of encouragement in my academic program, providing challenges and advice wherever I needed them.

I would also like to thank Dr. Thomas Kaczmarek and Dr. Rong Ge for serving on my thesis committee and helping me despite my hectic schedule.

Most importantly, I would like to thank my wife, Kristine Lize, for supporting me through this program. I am immensely gracious for her patience and loving support of me.

Finally, I would like to give thanks to my employer, Douglas Dynamics, Inc. for providing financial support of my academic career.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1: Introduction

Since the birth of the smartphone in 1993 [Bellsouth1], mobile computing has fast become a popular method of accessing data and information. With the advent of more powerful mobile hardware, technologies like cloud computing and creatively designed and applied application software, smartphones and tablet computers are beginning to replace the role of personal computers and laptops [Nagamine12a][Nagamine11a][Miller11] [Albanesius11] [Nagamine11b] [Fried11]. As evidence to the popularity of smartphones and tablet PCs, the fourth quarter sales for 2011 showed declines in PC sales growth [Nagamine12c] while the growth of smartphone sales hit a record high [Nagamine12c] and tablet sales outpaced their forecast [IDC12] in the same quarter.

Smartphones and tablets are also becoming more pervasive in enterprise environments [Haywood11] [Reed11]. However, as enterprises are accustomed to providing custom software to their end users, the development and support of software on these devices, which have unique design guidelines, several form factors, and various screen sizes, is challenging and time consuming.

The purpose of this paper is to introduce and demonstrate a design pattern, framework and API called ClayUI that assists in the development of mobile apps that adhere to their respective design guidelines. In this paper, we will show that certain portions of apps may be generalized enough to allow for dynamic layout reflow and runtime design changes which typically might require code changes and application updates. This paper will also propose the idea that by using this technique, an organization could develop a unified storage location and methodology for data created on mobile devices.

This paper will demonstrate an administrative user interface that a developer can use to define generalized portions of the user interface called app parts. The configuration for these app parts are stored in a database local to the mobile device as well as a database that is accessible via the Internet using web services. The paper will also demonstrate the use of an API that assists the developer with the integration of the app parts in their mobile application. Finally, because of the nature of the framework required for the distribution of app part configurations, we will demonstrate how our API also provides assistance with the process of saving mobile application form data to a local database and a remote database using web services.

The following describes the organization of the rest of the paper. In chapter 2, we will cover background information and describe any concepts that are integral to the paper but not necessarily generally known in the field of computing. In chapter 3, we will discuss related works. In chapter 4, we will describe the motivation for the ClayUI system. In chapter 5, we will cover a general overview of the ClayUI system. In chapter 6, we will cover the administrative web application used for setting up applications as targets for the ClayUI API. In chapter 7, we will cover the general usage of the API and its application in a prototype app. In chapter 8, we will evaluate the benefits of using ClayUI for application development. In chapter 9, we will conclude by describing the contribution of this thesis, summarize any short and long-term impacts and propose future work for this project.

Chapter 2: Background

In this chapter, we will introduce the background information and concepts that are integral to this paper but are not necessarily generally known in the field of computing. We will introduce some of the platforms evaluated for this project, discuss their differences and propose some of the challenges with developing for these different platforms. Because there are numerous mobile platforms currently in use in the mobile market, we will be focusing on the platforms used by Google and Apple. However, many of the concepts covered may be applied to any of the platforms present today.

2.1 Mobile Computing

Mobile computing is a form of computing where the user is able to access and process data into information from dynamic locations. Whereas traditional computing typically uses stationary personal computers with hard wired networks for communication and a constant power source, mobile computing involves technologies that allow for the mobility of the computer such as wireless networking and high capacity/lightweight batteries for a power source.

Mobile computing gives the user the ability to access information wherever he is at the moment as long as he has the required network connectivity. This ability has the potential for more timely information and communication, which is one of the key drivers for the increased of use of smartphones and tablet computers in the enterprise.

2.2 Mobile Computing Form Factors

In 1993 IBM unveiled what could be considered the first smartphone, the Simon [Bellsouth1]. However, according to the Merriam-Webster dictionary, the term smartphone was first introduced four years later in 1997 [Webster12]. The smartphone is a wireless, carrier-based phone that has additional software on it which allows it to perform tasks beyond telephony such as email, texting and Internet browsing. The growth and popularity of smartphones was accented when Apple released the iPhone in 2007 [Apple07]. A year later, Google released its first Android phone [T-Mobile08]. Other manufacturers of smartphones include: Research In Motion (Blackberry), Microsoft, HTC, Samsung, Sony and Nokia.

In 2010, Apple introduced the iPad [Eaton10]. Although Microsoft attempted to introduce their concept of a tablet computer over ten years earlier [Markoff99], the iPad was the first device in this form factor that had the potential to drive a change in the mobile PC market. The tablet computer is a mobile computer with a touch screen and typically lacks a keyboard. The user interfaces for tablet computers include custom software that is optimized for a touch interface. Larger icons to accommodate the size of the fingertip and gesture controls are common with tablets. Since the release of the iPad, the PC market has seen a steady decline in growth for PCs and laptop computers [Nagamine11a], while the iPad and other tablet computers enjoy steady growth.

2.3 Client/Server Computing

The client/server model of computing is a computing model where an application’s tasks are distributed between a client and a server. In mobile computing, the client is the mobile device, which has fewer resources for performing complex, long running tasks and limited data storage capacity for saving data. The server may be a web server that interacts with the client device’s processing requests to retrieve or save records from and to a database. An example of this computing model would be a messaging server’s contact database. The client may add new contacts to her mobile device and save them to the central contact database so that other people may access the contact as in a global address list. To store and search the contacts of every user of this application on the mobile device would be impractical due to the device’s hardware limitations. The client only needs to know how to contact the server to initiate a search. The server would then perform the search, providing the results to the client.

2.4 Cloud Computing

Cloud computing is a computing model that is very similar to client/server model in that the client’s workloads may be offloaded to a more capable device. However, cloud computing differs in that there is more of an abstraction of the specific server that the to which the client sends its requests. There may very well be a grid of hundreds of computers that will either, in turn, process a client’s request, or distribute the requested workload to several of the connected servers in the grid.

Cloud computing has several advantages over traditional client/server computing because compute and storage grids may be colocated in various locations across the globe. Servers that are geographically located closer to the client should respond with less latency than ones that are further away. Having servers colocated in separate locations also provides redundancy if there is a natural disaster or power outage at a location. An additional advantage of cloud computing is the ability to appropriately spread workloads to servers that are sized accordingly for the tasks.

2.5 Apple iOS

Apple introduced their first smartphone called the iPhone in 2007 [Apple07]. Later the same year, Apple introduced the iPod touch, which was similar to the iPhone but lacked the carrier based wireless interface. And in 2010, Apple introduced their tablet device, the iPad. All of these devices run Apple's mobile operating system called iOS. The mobile operating system iOS is based on Apple's OSX operating system, a BSD Unix-like operating system [Apple08]. At its initial release, the only way to develop applications for the iPhone was to develop web based apps. While this was successful, it was limiting because a user could only use the apps when a good connection to the Internet was available

At a town hall meeting in 2008 [Block08], Apple announced that they would release a Software Development Kit (SDK) for iOS so developers could create native applications for the iPhone. This SDK, which is developed in Objective-C, is derived from a subset of the foundation of OS X. The SDK provides user interface elements via the Cocoa Touch framework and allows the developer to create applications that interact with the hardware much like an application would on OS X. The commonality between iOS and OS X development provides a great level of fluidity within Apple's entire ecosystem. For a user who understands using the Mac OS X, an iOS device is also familiar.

2.6 Google Android

Google introduced its first smartphone in 2008, which was manufactured by HTC [T-Mobile08] and its first tablet in 2011 manufactured by Motorola [Savov11]. These devices run Google's mobile operating system called Android, which Google acquired from the company Android, Inc. in 2003 [Markoff07]. Android is a Linux-based mobile operating system, which provides a Java framework for developing applications.

Android's SDK was released prior to the introduction of its first smartphone as part of the Open Handset Alliance [Rubin07]. The SDK, developed in Java, provides all of the necessary components for developing applications on Android devices. The SDK provides user interface elements through interactive widgets and allows the developer to interact with the hardware and other applications through a system of activities and intents.

Because of the open nature of Android, different vendors of smartphone hardware customized the user experience of their Android phones. For instance, Samsung uses their proprietary user interface Touchwiz, while HTC uses their proprietary user interface Sense. These two user interfaces deviate from the standard user interface developed by Google. Because of this, a user of an Android smartphone from one vendor may not be familiar with using an Android smartphone from another vendor. This resulted in criticism of the Android smartphone market identifying a fragmentation and inability to maintain updates to current hardware.

2.7 Design guidelines for iOS and Android

Apple and Google both have a set of guidelines for developing visually pleasing and easy to navigate user interfaces for apps. However, while the typical iOS application maintains a look that is common to applications used in O SX and iOS, Android apps may offer a look that is unique to Android alone. Both platform design guidelines stress the importance of maintaining consistency within the platform to provide a fluid experience between apps. As an example of the differences between the two platforms the Figure 1illustrates commonly used controls: iOS on the left and Android on the right.

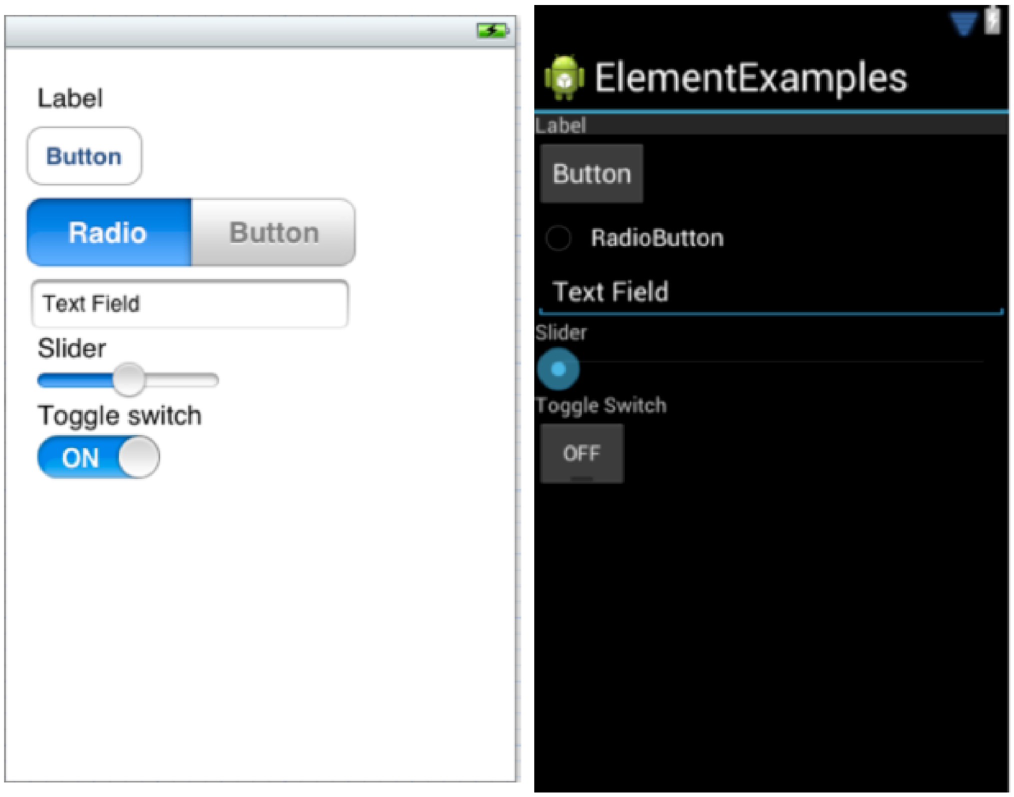


Figure 1 -- A comparison of common controls

As illustrated, these common controls differ in appearance; the iOS controls have a rounded look to them while the Android controls have sharper edges and appear flatter. One could certainly create icons for controls that would achieve the same look on Android as are available on iOS and vice versa. However, doing so would interrupt the flow of the user's experience from the rest of the device.

In addition to having design guideline differences and common control differences, the SDKs for iOS and Android provide controls that are unique to their platform. For instance, a common control utilized in user interface design is the combo box. While both iOS and Android do not provide a control for a traditional combo box, they each provide their own alternative to a combo box. The implementation of these controls is similar, however, their functionality and appearance are very different. Figure 2 illustrates the difference between the iOS and Android respectively.

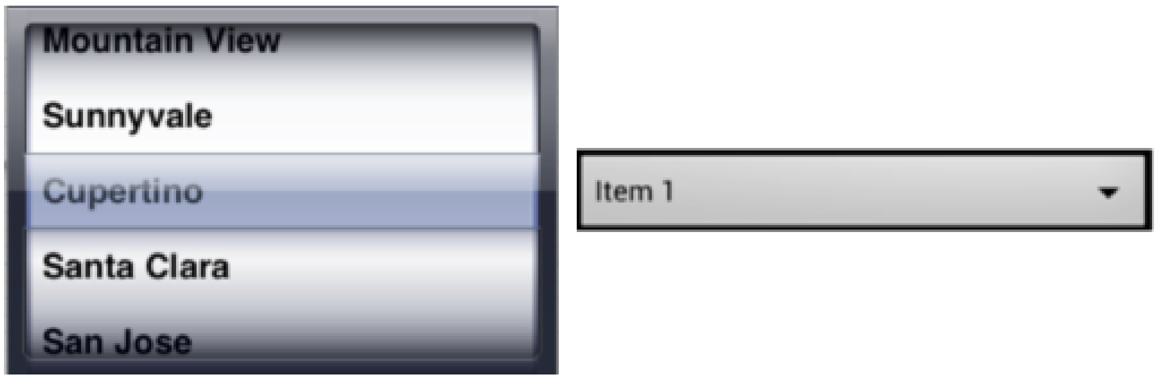


Figure 2 -- Combo box differences

It is easy to see in this example that the iOS Picker View control's spinning wheel look appears to be more of a physical element, while the Android Spinner control resembles the traditional combo box. The iOS Picker View consumes more space than the Android Spinner control.

2.8 Challenges of Supporting Multiple Platforms

When developing for multiple platforms one must consider the differences in platform capabilities as well as the similarities. The developer must also determine if whether it is desirable to maintain a common look and feel for the application across platforms or if the application should conform to a common look and feel within the platform itself. If it is desirable to present a user interface that conforms to the design guidelines of the platforms, the developer will need to understand how to implement the different common controls that are available for each platform.

In addition to user interface elements, the developer may wish to provide the ability to locally store data on the device as well as save the data to a remote location through a web service. Again, the developer needs to understand the way this is implemented on each of the platforms he or she wishes to support.

Chapter 3: Related Work

The field of mobile computing is filled with various devices from a number of hardware vendors. These devices vary in many ways including hardware capabilities, screen sizes, network speeds and types; likewise they operate on several mobile operating system platforms. This situation presents a unique challenge for developers who wish to develop applications that reach a broad audience using these various devices.

There are several approaches that address the issue of supporting multiple mobile platforms and environments. A developer may choose to use a programming language that is rendered at run time such as HTML5 or JavaScript. An application may use a proxy that changes the content that is delivered to the mobile application based on the capabilities of the device. Or, an application may use middleware that provides services based on the context of the device, such as battery level, network availability or location. Our approach is to develop mobile applications in their native programming language and apply a design pattern that uses an API to assist in the creation of user interface elements that conform to the design guidelines of the target platforms. To assist in this, we also introduce a web-based assistive application for configuring the user interface elements.

In this chapter we describe some of the research that attempts to address these challenges and compare the approaches with our framework.

3.1 Code once distribute many

One of the ways that a developer may accomplish developing an application that targets multiple platforms is through the use of technologies that are supported on all of the target platforms. This typically involves using web technologies such as HTML5. One such project that accomplishes this is called Rhomobile [Rhomobile1]. Rhomobile is a suite of development applications that assist the developer in creating HTML5 based applications that are distributable to various platforms.

Our framework is distinct from this project in that we use native programming languages and allow the developer to conform to the design guidelines of each targeted platform. Since a developer uses Rhomobile to develop the application, he would need to design a common look and feel for the application that would not necessarily conform to the design guidelines of each of the target platforms.

3.2 Dynamic Content

In order to address the challenge of building applications that target platforms with various capabilities and resources, be they network restrictions or hardware resources, some papers demonstrate using a middleware that dynamically changes the content based on these restrictions is effective for improving the overall user experience of mobile applications and web applications.

In their paper, A. Fox et al. [Fox98] proposed that by using a proxy service which is aware of the clients' restrictions, they would be able to improve an application's user experience by changing the content the client is requesting on the fly. For instance, if the client were to request a page full of high-resolution images, the proxy service would be able to re-render these images down to a smaller scale.   This would cut down on the network bandwidth, computational processing and memory requirements of the application.

H. Zhang and W. Ma from Microsoft Research propose in their paper [Zhang04] that a new web content representation document called Scalable Web Document could be used to reformat web content based on the screen size of the device requesting it. The Scalable Web Document would assist the mobile application by adjusting text layout and reformatting large images via a proxy service.

In their paper, Z. Hua et al. [Zhigang06] discuss a project that addresses the screen size constraints of mobile devices named MobiDNA. In this project, web content is broken into blocks and later cached so that a handheld device could more easily display its content. Their project uses a novel method of reducing areas of web pages into thumbnail-like sections that a user can both navigate and zoom in to view detail.

In our framework, we use an internal database to define the content and application elements that are available to the application. The ClayUI API also includes the ability to modify the contents of the internal database based on the results from the web service queries that the framework provides and refreshes the user interface based on this new content. Our framework does not need to address the hardware restrictions of some devices because the developer knows the target devices which will ultimately use the application. These restrictions should be considered as if the developer were building the application for a single device.

3.3 Dynamic Services

Another area of related research is the process of dynamically changing available services based on location, application context, and device types and requests. In the following papers, researchers use the various sensors available on the devices and middleware to assist the device in locating the appropriate services for the tasks the software completes.

In their paper, A. Cole et al. [Cole03] discuss the process of binding middleware to services based on the location of the device. For instance, one service available to the device might provide traffic congestion information for the metropolitan area where the device is located. Once that device moves to a new metropolitan area, a new provider may be available for this service while the original one may not be available. The authors suggest that through the use of middleware, a device could deterministically switch services without the end user knowing.

Another method proposed for dynamically changing the services available to a mobile device is through an applications context.   L. Capra et al. describe a project named CARISMA in which middleware maintains the current context of a running process and changes its behavior based on this context [Capra02]. In this project, L. Capra et al. describe that the middleware could react to situations such as low battery power by reducing graphics resolutions or color depths to conserve power. In their paper, authors A. Murarasu and T. Magedanz [Murarasu09] propose a process of shifting workloads from the local device to a remote device based on the current load of a mobile device. Using this method, an application dynamically utilizes remote services instead of local services without the knowledge of the end user. In his dissertation, P. Grace describes another method of adapting to the devices context by developing a middleware that reacts to context changes and utilizes different frameworks based on the change [Grace04].

In the paper by F. Chien-Liang et al., the authors propose a method of reprogramming wireless sensor network motes based on the recorded information sent to one another [Chien-Liang05]. In their paper, the authors demonstrate the flexibility of reprogramming a sensor network for a new purpose through the example of a fire detection system. They demonstrate that the fire detection system could be reprogrammed to act as a search and rescue system once a fire is detected. This reprogramming is necessary due to the lack of data storage and memory on the motes used in the system.

In our framework, ClayUI, we dynamically build local resources that define local user interface elements. While this resource may remain static through the application's lifecycle, changes to the underlying database structure may be dynamically passed to the configuration of the user interface elements if the developer so chooses.

3.4 Assistive application for code generation

In order to assist professors of the University of Massachusetts Amherst with the distribution of classroom content to mobile devices over the Internet, the RIPPLES group at the University developed the Multimedia Asynchronous Networked Individualized Courseware (MANIC) system [Schapira01a] [Schapiara01b]. This system simplifies the process of posting courseware to a website which is then pushed to mobile devices for offline viewing. This system does not require the user to know any HTML or mobile application development as the system handles this for the user.

In our framework, we provide an administrative website for configuring the parts of the application that are controlled by ClayUI. To configure these application parts, one only needs to have an understanding of the different user interface elements that are used and how to set them up in ClayUI. While a developer with experience of developing for the target platforms is necessary, this is only true for the initial setup. Any changes done to the ClayUI application part will be reflected in the user interface of the application without the requirement of any development experience.

The following table (Figure 3) compares the features of several related named projects and ClayUI.

Feature Description	Project Name			
	ClayUI	Rhomobile	CARISMA	MANIC
Web based assistive application	Yes	No	No	Yes
Device native programming language	Yes	No	Yes	Yes
Device native design principal	Yes	No	Yes	Yes
Support multiple platforms	Yes	Yes	No	No
Runtime application changes	Yes	No	Yes	No
Data storage assistance	Yes	No	No	No

Figure 3 -- Comparison of features

Chapter 4: Motivation

As smartphones and tablets become more popular, software developers struggle to create successful apps that target the various platforms in a timely and cost effective manner. Developers also have to make a conscious decision to either follow the design guidelines for each platform, or develop their own guidelines across platforms. Using a consistent design between different platforms may be beneficial from a development and support standpoint, but it leads to problems where the user loses a sense of flow that is established by the platform vendor. Designing around the platform's design principals helps establish a comfort level for the end user. A design pattern and API that assists developers as they create common controls and their associated methods, provides an opportunity to reduce time when developing applications that conform to platform design principals. It is the purpose of this paper to demonstrate such a design pattern and API to provide evidence of its benefits.

Every year smartphones and tablets become more affordable and more capable. Because of this, these popular devices are more pervasive in our lives. In 2011, the personal computer market saw the smallest growth in recent history [Nagamine12c], while the smartphone and tablet markets produced record growth [Miller11][Nagamine12b]. This trend of purchasing smartphones and tablets over PCs is an indication of the beginning of a shift of the mobile computing form factor from traditional laptop PCs to smartphones and tablets. It is estimated that tablet computer sales grew by more than 181 percent from 2010 to 2011 and may continue to grow an additional 89 percent from 2011 through the end of 2012 [Pettey10]. This is compared to a growth of only 3.8 percent for PCs 2011 and a forecasted 10.9 percent growth in 2012 [Pettey11].

One of the reasons for the popularity of the smartphone and tablet is their ability to run apps that provide much of the same functionality that traditional PCs provide. These devices include web browsers, email clients, newsreaders and many other productivity apps that add usefulness to the device. While many of the apps that are available on various platforms perform the same function, the design strategies may differ in their look and feel. Each vendor invests a considerable amount of time and energy into improving the user experience with its platform. This is even the case within the different hardware vendors for Google's Android platform. For instance, Samsung and HTC developed their own user interfaces named Touchwiz and Sense, respectively. These vendors also develop their own apps to replace the default Google experience apps so they can control the designs of their respective platforms.

Smartphones and tablets are also becoming more prevalent in business enterprises. And where traditionally the IT department dictates which brands and platforms are acceptable in the workplace, the trend is moving towards the practice of employees bringing their own devices to work [Reed11]. This practice increases the complexity of developing applications for different platforms that conform to their design guidelines by increasing the heterogeneity of platforms supported by the developers.

Tools that are available to developers to assist with completing tasks such as setting up common controls in user interfaces, setting up database connections, and setting up communications with web services increases productivity and reduces the development lifecycle time. Here we consider a few scenarios to demonstrate the benefits of the ClayUI framework.

Scenario 1

A research firm wants to develop an application for mobile devices to help its researchers track answers to questions during an interview. These multiple choice questionnaires may change based on the subject of the interviewee. The firm would also like to be able track the results of these surveys in a database for further analysis. It is assumed that the interviewer will not always have a persistent network connection for accessing surveys, so a web-based application could not be considered.

Our framework, ClayUI, would assist the developers with setting up the user interface by simplifying the process of populating repetitive user interface controls for presenting the questions in the interview. ClayUI would also set up the data adapters and backend database used for storing results from the surveys. Finally, because the firm wants to track the results of the surveys for further reporting, ClayUI would set up the necessary helper objects for persisting the results to a web service that would write the data to a backend database used for reporting.

Scenario 2

A business that allows its sales force to use their personal smartphones and tablets for business use wants to extend the sales lead application built in-house to the sales personnel’s smartphones and tablets. The types of devices that the sales force uses is a mix of iPhone smart phones, iPad tablets, Android phones and tablets and Windows Mobile phones. The IT department of this company understands that the users of these devices chose them based on their preference for the way the device's operating system functions and looks. The IT department desires to provide a new application that conforms to the guidelines of their respective devices.

ClayUI would assist the developers with setting up the user interface and local backend database for storing contact information. As an optional benefit, the developers could use ClayUI to store contact information in a central database by saving the data through a web service.

Scenario 3

An ecommerce site wants to provide a mobile app for its customers. As a customer courtesy and because the company knows its customers use a variety of smartphones and tablets, the team decides to develop apps that conform to the design guidelines of the various platforms that its customers use. Much of the user interfaces for these will use common controls and the data for the mobile apps is stored on web servers where retrieval should be completed using web services. Additionally, the app should allow the user to store her shopping cart on her local device for later retrieval.

ClayUI would assist the developers by setting up the user interface controls that are common between the different platforms conforming to their respective design guidelines. ClayUI would also assist in providing the necessary objects for storing shopping cart data to the local database on the mobile device as well as the objects for retrieving items that are available for purchase from the web service.

For all of the above scenarios, our research indicates that a developer can use a framework and API to assist in the processes of developing an application that targets multiple mobile platforms. This process reduces the overall development time by configuring common user interface controls, configuring connections to a local database and configuring connections to web services for retrieving and saving data. This process also alleviates some of the burden of re-configuring common items on different platforms where certain technologies in use may be different. For instance, Android and iOS use SQLite as their database backend where Windows Mobile uses Microsoft SQLCE. And finally, this process reduces the total cost of support for the applications through the use of APIs, which may be independently tested for bugs.

Chapter 5: Overview

ClayUI is a framework for assisting developers of mobile applications that target multiple platforms with differing design guidelines. The framework is comprised of five major parts: the application administration web application, a server database backend, web services, a local database on the client, and the client APIs. Figure 4 represents the overall design of the ClayUI framework and how each component interacts with each of the others.

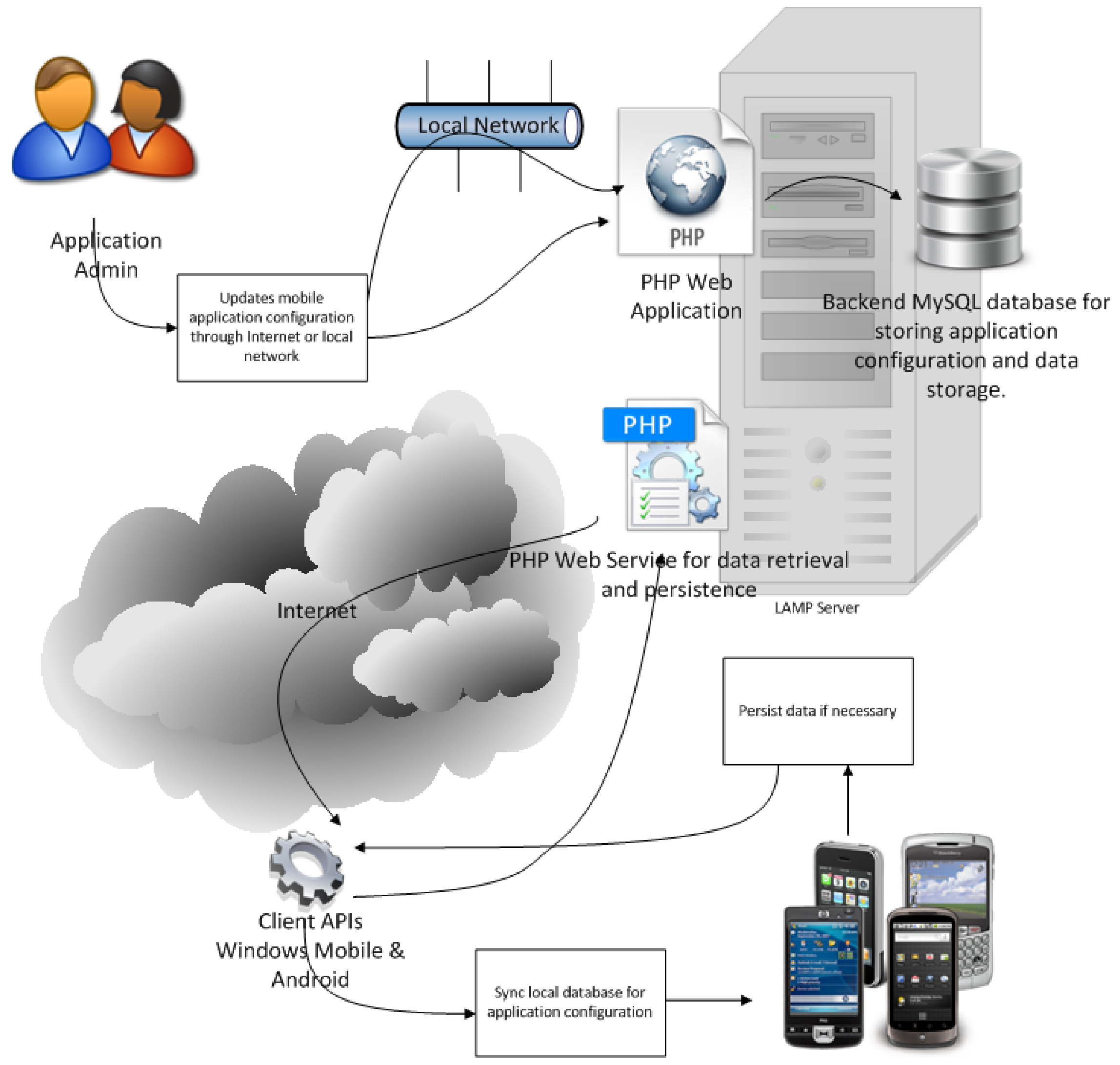

Application Admin
Local Network
PHP
PHP Web Application
Backend MySQL database for storing application configuration and data storage.
Updates mobile application configuration through Internet or local network
PHP
PHP Web Service for data retrieval and persistence
LAMP Server
Internet
Persist data if necessary
Client APIs Windows Mobile & Android
Sync local database for application configuration

Figure 4 -- ClayUI Overview

In the following chapter, we will give an overview of each of the major components used in ClayUI.

5.1 Administrative site

The administrative website uses open source components to deliver an interface to the user via its web browser. The application is developed using a combination of HTML5, CSS3 and PHP, and built on an open source LAMP (Linux, Apache, MySQL, PHP) foundation to provide a simplified interface that is dynamic and responsive. Using the administrative site, a developer can rapidly configure the parts of the application he wants controlled by ClayUI with a basic understanding of user interface elements.

5.2 Database backend

The storage behind the administrative website uses the open source database engine MySQL. This database houses all of the configuration information for the application parts that a developer creates as well as any of the data that her applications generate if she so chooses. Since the data is stored using a database engine that includes connectors for multiple platforms, developers can further extend the functionality of the database backend for reporting or for other applications.

5.3 Web Services

ClayUI uses several PHP web services that produce JSON (JavaScript Object Notation) objects in order to pass data between the administrative database backend and the client API. This method provides a language independent data descriptor that is lightweight and easy to read.

5.4 Local Database

ClayUI stores the application configuration in a local database of the mobile device. Depending on the target device, the database engines used are SQLite for iOS and Android devices, and the SQL Server Compact Edition for Windows Mobile devices. These database engines are lightweight database engines that provide more capabilities than storing configuration and data in flat files.

5.5 Client APIs

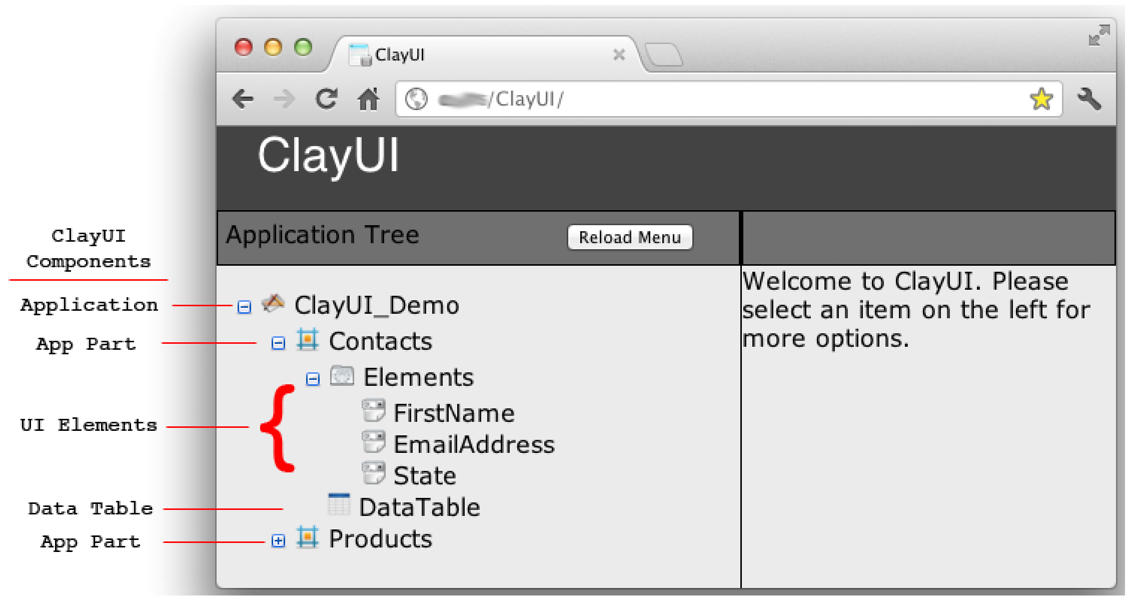
From the point of view of the client application, the API is the heart of the framework. The ClayUI API is structured in such a way that the developer only needs to instantiate a few objects and execute their methods to control the layout of the application parts, and save data locally or to the web. All of this is accomplished without the requirement of the expertise for implementing this on the various platforms she is targeting for her application.

Over the next couple of chapters we will go into further details of the aforementioned components of ClayUI.

Chapter 6: Administrative Interface for Adaptable User Interfaces

One of the goals of the ClayUI framework is to ease the burden of setting up and modifying portions of an application that use common user interface elements. Another goal was to provide this ability on all of the development platforms, including: Microsoft Windows, Apple Macintosh, Linux and others. In order to accomplish these goals, the ClayUI administration site is built using web technologies that make the tool available via a modern web browser. These technologies: Linux, Apache, MySQL, and PHP, commonly referred to as a LAMP stack, are now considered an industry standard method for delivering dynamic and data-driven web applications.

The application administration website is laid out with a menu on the left side which represents an application tree. An application tree is a graphical representation of the components that ClayUI maintains. The structure of the administration side of ClayUI is broken into five major components: Applications, App Parts, Elements, Data Tables and Web Services. Figure 5 illustrates the overall user interface of the ClayUI administration application and the components with which an administrator interacts.



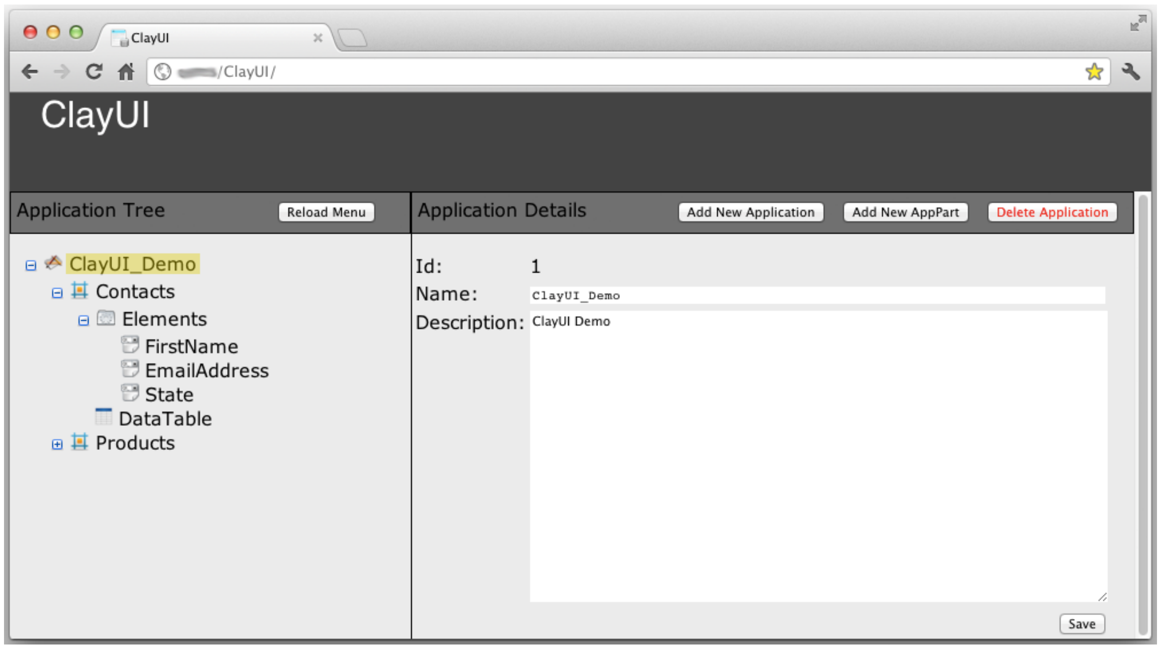
Welcome to ClayUI. Please select an item on the left for more options.

Figure 5 - ClayUI Administration Application

Clicking on any of the components in the application tree displays the detail screen to the right of the application Tree. The detail screens are used to define configuration values that are used in the app parts of the application. In the following sections, we will cover each component of the application administration interface and describe their role in the framework.

6.1 Application

ClayUI has the ability to control the configuration of multiple applications. Each application is represented as the root element of the ClayUI application tree. Once the application is created in ClayUI, additional details such as the application's name and a description may be added to help document the application. Figure 6 illustrates the application detail screen.



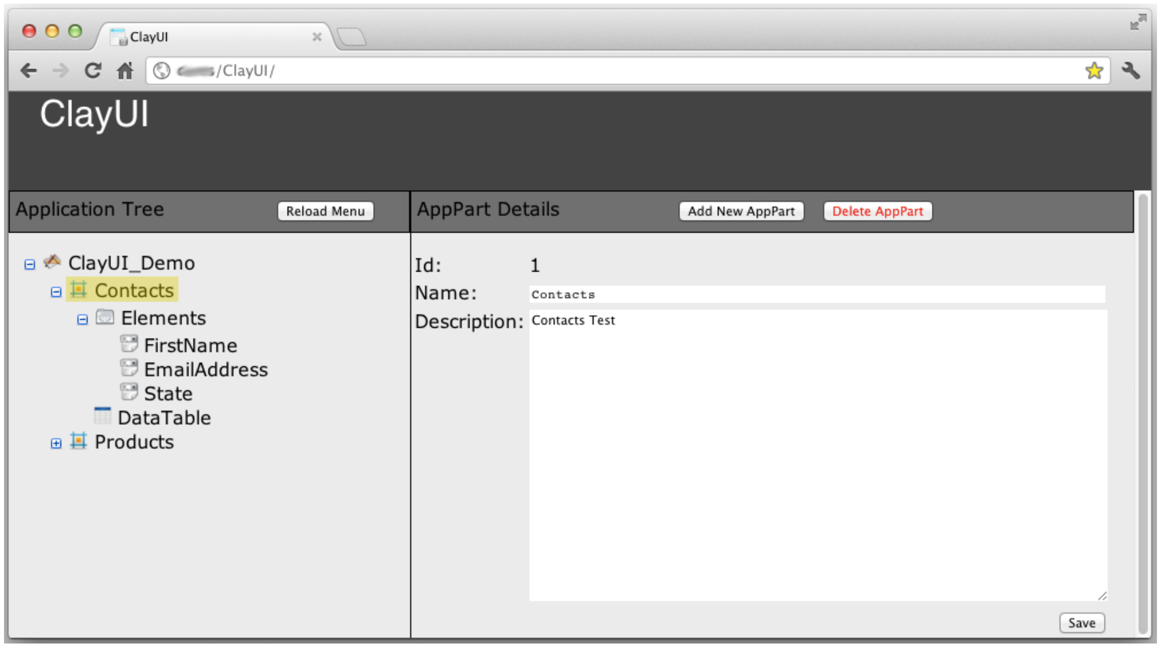
Id: 1
Name: ClayUI_Demo
Description: ClayUI Demo

Figure 6 - Application Details

Figure 6 demonstrates buttons for adding a new application, adding a new app part, or deleting the application. The name value is used to identify the application on the application tree and does not have to be unique as the application ID value identifies the application in the backend database. The ID value is also used when the developer is assigning the application ID later in the ClayUI API.

6.2 App Part

The next component down the application tree in ClayUI is the app part. An app part is the section of the mobile application the developer would like to control with ClayUI. These app parts are flexible in that the developer may code around the app parts, controlling some elements outside of ClayUI. Some examples of this would be tab controls for separating functionality, buttons that add functionality, or completely new forms within the application. Figure 7 illustrates the app part detail screen.



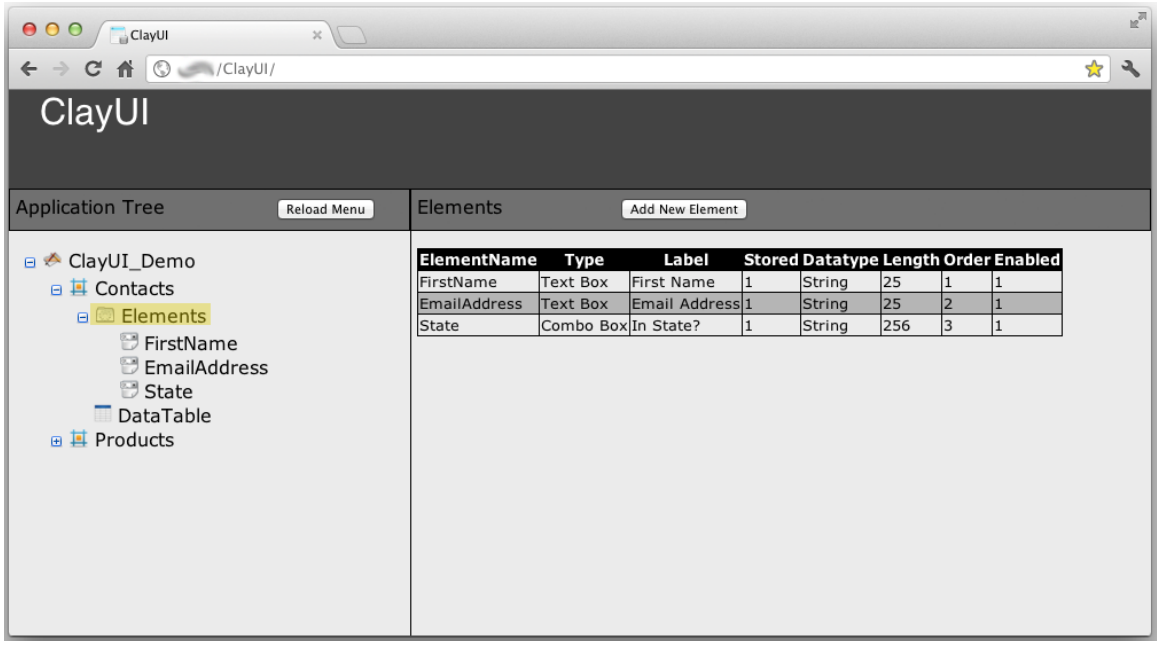
Id: 1
Name: Contacts
Description: Contacts Test

Figure 7 - App Part Details

The above figure demonstrates buttons for adding additional app parts to the application or deleting app parts from the application. The app part name value is used to identify the app part on the application tree. This value does not need to be unique as the app part ID value is used to uniquely identify the app part in the backend database. The app part ID value is also used when the developer is assigning the app part ID later in the Clay UI API. There is no limit to the number of app parts that a developer is able to add to his application.

6.3 Elements

The next level in the application tree is the elements level. At this level the developer will define the user interface elements she wishes to have in her app part. These elements are common user interface controls used in mobile applications. In the administrative application we use a generic name for the controls because the ClayUI does not know the platforms the developer will use for the final application. Figure 8 illustrates the table of elements currently configure for the app part displayed when selecting the elements item in the application tree.



ElementName	Type	Label	Stored	Datatype	Length	Order	Enabled
FirstName	Text Box	First Name	1	String	25	1	1
EmailAddress	Text Box	Email Address	1	String	25	2	1
State	Combo Box	In State?	1	String	256	3	1

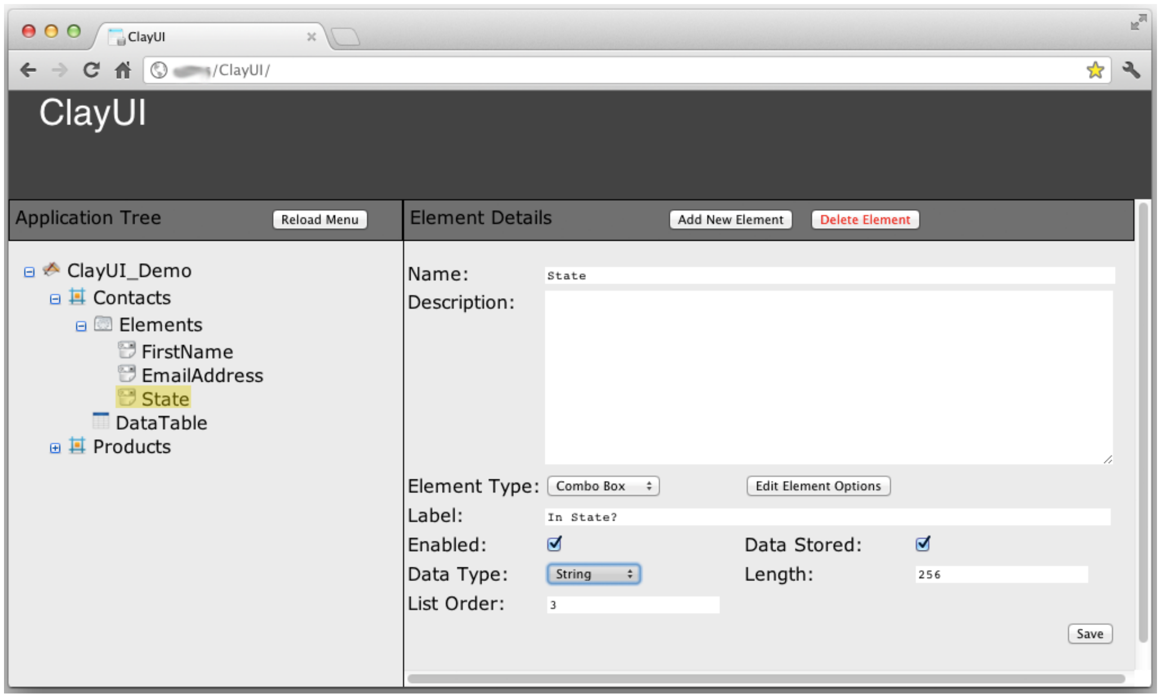
Figure 8 - Elements Overview

Each element added to the app part has the following details that a developer can use to configure the layout of the app part in his application:

- Name – The name of the element is used to represent the element in the application tree as well as to assign a control name on the Windows Mobile devices. This value must be unique to each app part.
- Description – This value is not required, but may be used to further describe the purpose of the element.
- Element Type – This value determines the type of user interface control the ClayUI API will place in the mobile application. Available user interface element options are:
    - Text Box – An element that allows the user to free form type text
    - Label – An element that statically displays text
    - Combo Box – An element that gives a list of items to choose from in a drop down list
    - Radio Button – An element that displays a grouping of items to choose from with selection boxes
    - Check Box – An element that gives the user the ability to either select or de-select the item using a single selection box
- Label – Text that will display along with the user interface element. If label is the selected element type, this is the only text that is displayed
- Enabled – Determines whether or not the ClayUI API will display the element in the mobile application. This option is useful for creating new versions of an application where the developer wishes to retain historical data from a previous version of an application

- Data Stored – Determines whether or not this element will have an associated data field in the underlying data table used to store user generated data from the mobile application
- Data Type – The data type maps to a MySQL data type used in the data table. Available data types are:
    - String – a variable length data type for storing text
    - Integer – a signed or unsigned exact numeric
    - Decimal – a signed or unsigned approximate numeric with a default precision of 18 and scale of 4
    - Date – short date data type such as ‘12/12/2012’
    - DateTime – date time stored such as ‘12/12/2012 00:00:00’
- Length – If the selected data type is string, the length field is used to determine the maximum number of characters stored in the field
- List Order – The order in which the elements will appear in the user interface. If all of the list order values are the same, the default order is in the order that the elements are added to the app part in the application administration site

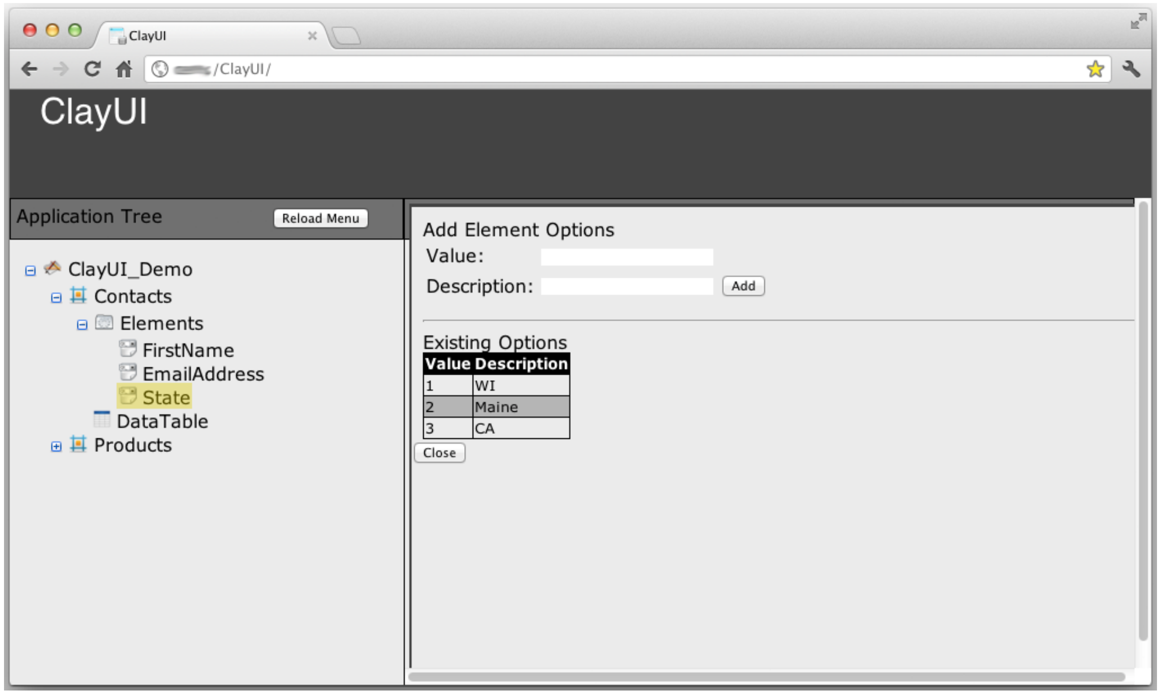
Figure 9 illustrates the form used to configure the element.



Name: State
Description:
Element Type: Combo Box
Label: In State?
Enabled: ✓
Data Stored: ✓
Data Type: String
Length: 256
List Order: 3

Figure 9 - Element Details

If the developer chooses to use the element types combo box or radio button, he also has the ability to define the options that the user will have from which to choose. This is accomplished by selecting the Edit Element Options button in the element details form. The form that is displayed gives the user the ability to add a value and description to the option. The value field is used in the data table for efficiency of sorting and joins. The description field is used in the user interface to define the options available for the combo box or radio button group. After the developer fills out the two fields and selects the Add button, the form will refresh with the new values. Figure 10 illustrates this form.

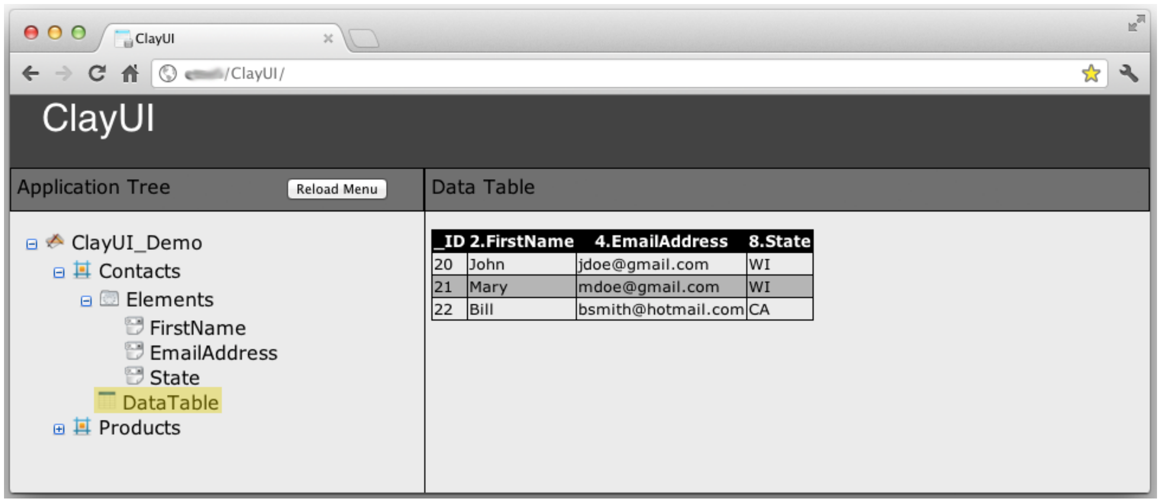


Value	Description
1	WI
2	Maine
3	CA

Figure 10 - Element Options

6.4 Data Tables

The final component of the application tree is the Data Table. The data table detail form gives the developer a look into what records are already stored in the underlying data table for this app part. As new records are added, this view provides a convenient way for the administrator to see what records are being written to the database via the web services ClayUI provides. Figure 11 illustrates this view.



_ID	2.FirstName	4.EmailAddress	8.State
20	John	jdoe@gmail.com	WI
21	Mary	mdoe@gmail.com	WI
22	Bill	bsmith@hotmail.com	CA

Figure 11 - Data Table view

6.5 Web Services

In addition to an interface for creating applications and their respective app parts, the ClayUI application administration site provides PHP web services; these web services provide the facility of retrieving the application configuration and accepting records from the ClayUI API for storage in the data table of the app part.

All of the functions of the web services are written using a RESTful architecture. Using this method, all calls to web services use standard HTTP GET and POST methods. REST, which stands for Representational State Transfer, is supported by almost all HTTP clients on mobile devices and is slowly becoming the dominant interface method for building web services [Rodriguez08]. The architecture defines that the web services should use data descriptors such as XML or JSON. We chose to use JSON because it is supported on more mobile devices than XML. However, the REST architecture suggests that both JSON and XML could be used based on the request of the client.

All of the functions of the web services are written into the ClayUI API so that the developer need not know how to access them. However, for the purposes of this paper, the following services are defined and available:

- GetAppParts – Returns a JSON object of all of the app parts for the specified application ID
- GetElements – Returns a JSON object of all of the elements for the specified application ID and app part ID
- GetElementOptions -Returns a JSON object of all of the element options for the specified application ID and app part ID
- GetDataTableSchema – Returns a JSON object of the schema information that the ClayUI API uses to define a data table in the local database for storing app part data
- PutTableData – Accepts a JSON object of data table records to send to the ClayUI database

The calls to these web services use stored procedures in the ClayUI MySQL database to minimize the possibility of a SQL injection. This also further simplifies the routines that the PHP web services call to retrieve or accept the JSON objects used in ClayUI.

Chapter 7: ClayUI API Details

One of the goals of ClayUI is to demonstrate a design pattern that is useful on several mobile operating system platforms using the native programming language. This is to ensure that the developer has the flexibility to provide native applications that conform to the platform's design guidelines while simplifying the process of implementing segments of the application that are controlled by ClayUI. To this end, it is necessary to provide an API that is similarly structured for each of the platforms so that a developer should easily understand the method of implementing the API in her application. Figure 12 gives an overview of the structure of the ClayUI API and the relationship of its components with other UI elements in a mobile application:

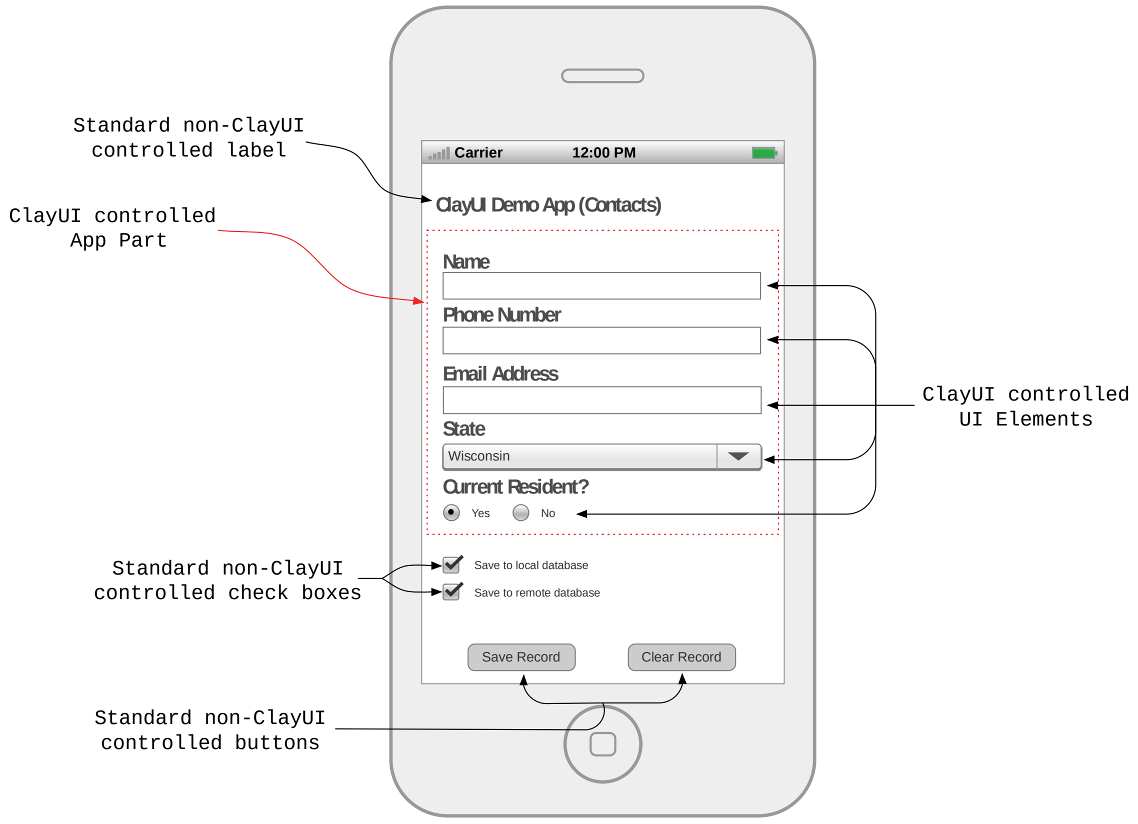

Standard non-ClayUI controlled label
ClayUI controlled App Part
Standard non-ClayUI controlled check boxes
Standard non-ClayUI controlled buttons
ClayUI controlled UI Elements
Carrier
12:00 PM
ClayUI Demo App (Contacts)
Name
Phone Number
Email Address
State
Wisconsin
Current Resident?
Yes
No
Save to local database
Save to remote database
Save Record
Clear Record

Figure 12 - ClayUI Components

The ClayUI API provides all of the facilities necessary for connecting to the ClayUI web services and local database. It also provides all of the necessary methods for creating a structure for laying out user interface elements that ClayUI will control. In the following sections we will explain each of the major areas of the ClayUI API and provide appropriate use cases for each area.

7.1 ClayUI App Base

The ClayUI App Base class is the foundation for all of the facilities of ClayUI. Thus, this class is one of the required classes a developer will need to instantiate in his application. The app base class is responsible for setting up a new local database and creating all of the tables that hold the structure for the app parts configured for the application. The app base class instantiates the utility classes for the app part, element and element option classes of the application. These utility classes contain methods the base class calls to synchronize the local database with the ClayUI remote database by sending requests to the ClayUI web services. The utility classes also contain methods for saving app part data to the local database and for saving local database records to the ClayUI remote database through the ClayUI web service. Figures 13 and 14 illustrate the use case for the app base class and the conceptual class diagram for the hierarchy of the app base class. Both of these diagrams are derived from the Android implementation, which uses Java. However, the diagrams for Windows Mobile and iOS would be similar.

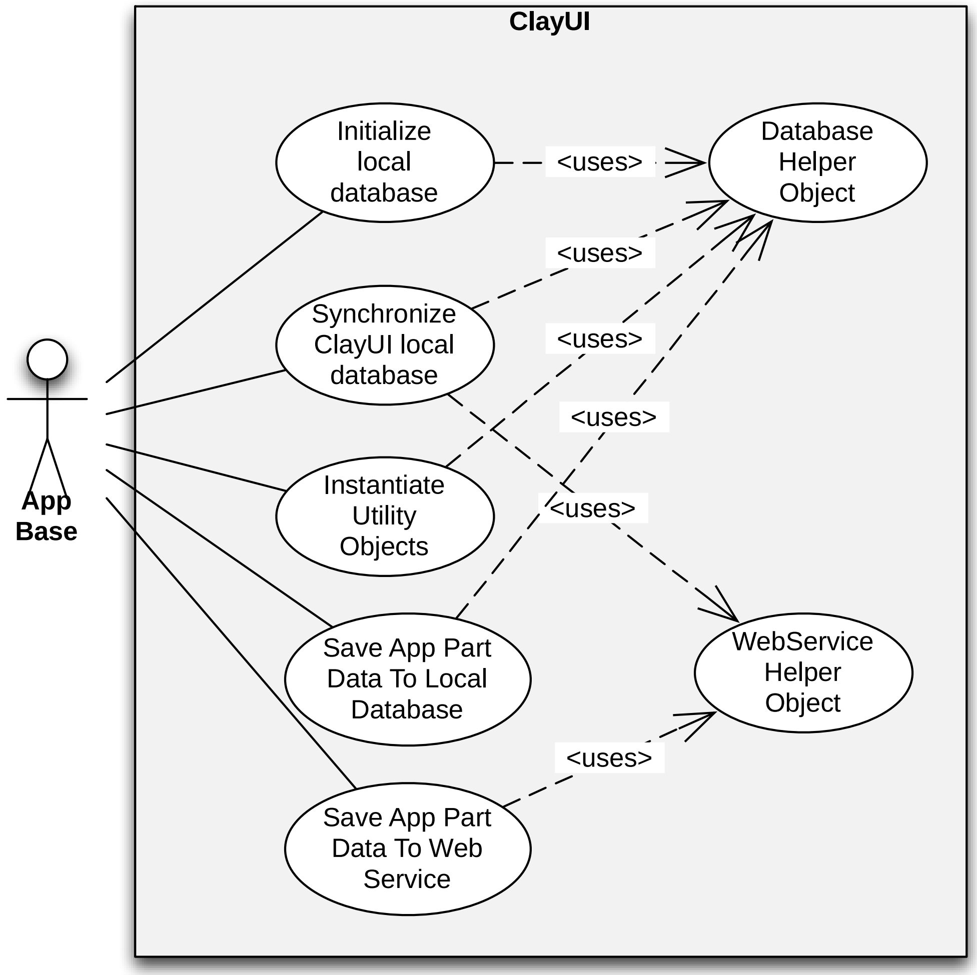

ClayUI
Initialize local database
Synchronize ClayUI local database
Instantiate Utility Objects
Save App Part Data To Local Database
Save App Part Data To Web Service
Database Helper Object
WebService Helper Object
<uses>
App Base

Figure 13 - App Base Use Case

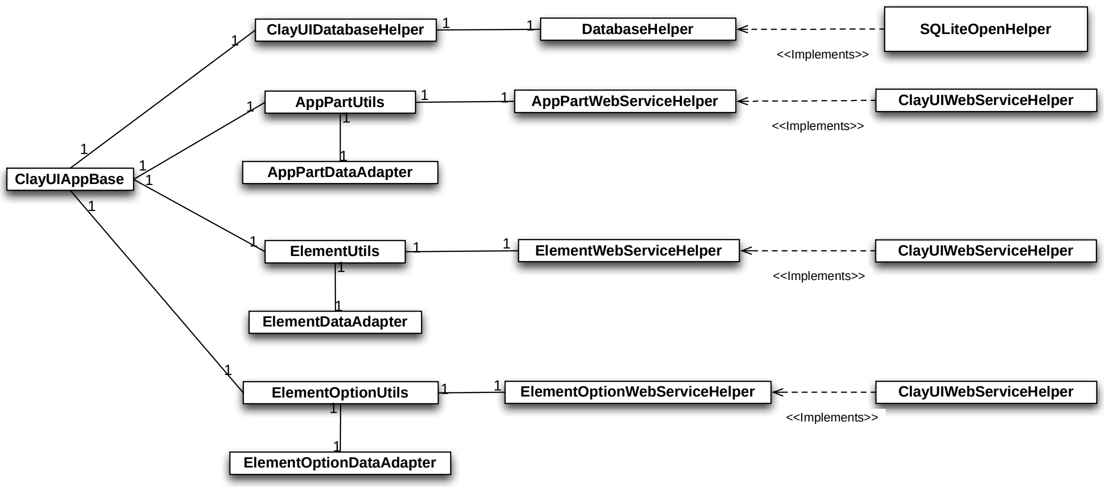


Figure 14 - ClayUI App Base Conceptual Class Diagram

7.2 App Parts and App Part Utils

The app part class is significant because it represents the portion of the user interface that is controlled by ClayUI. Each app part defined in the ClayUI administration application is mapped to an app part class in the mobile application. Thus, it is necessary to instantiate an app part class for each of the app parts defined.

The app part class is responsible for fetching the user interface elements from the local database and adding the elements to the user interface. The app part class also defines the appropriate methods for refreshing the user interface layout if the element definition of an app part changes.

Figures 15 and 16 illustrate the use case for the app part class as well as the conceptual class diagram.

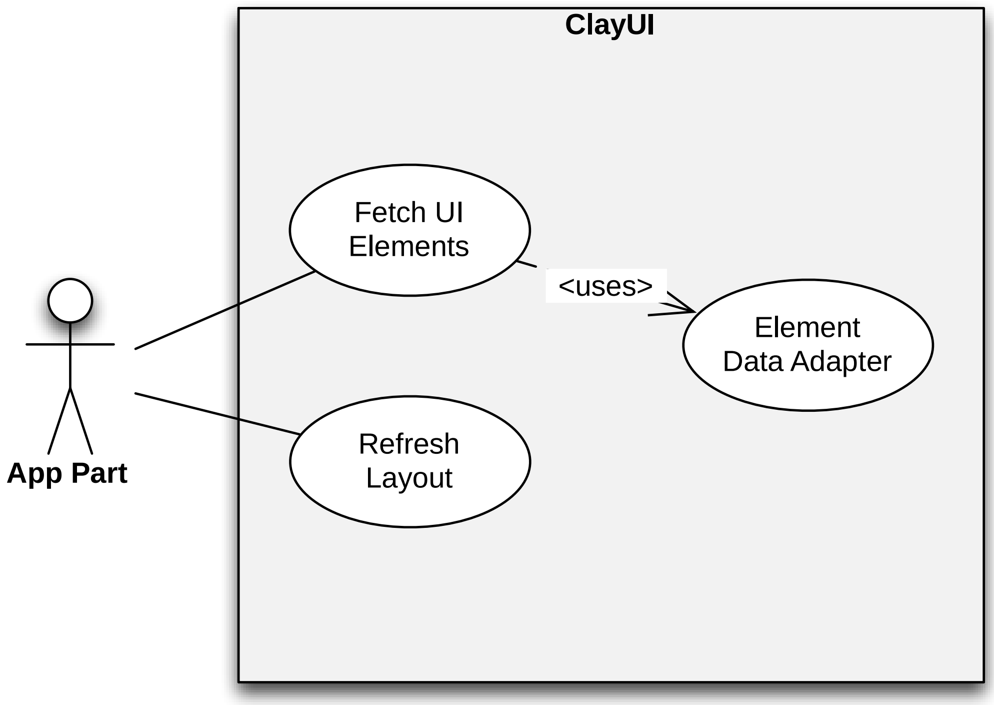


Figure 15 - App Part Use Case

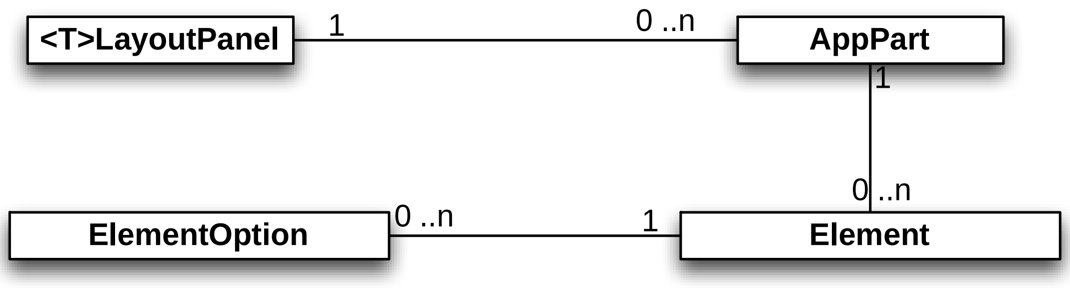


Figure 16 -- App Part Conceptual Class Diagram

It should be noted in the conceptual class diagram, a class of the type layout panel references the app part class. This type is a generic representation of a panel class within the application. For instance, using the Android platform, the layout panel could be a LinearLayout class. The layout panel is necessary as it is the class that handles the population of user interface elements in the mobile application.

Related to the app part class is the app part utils class. The purpose of this class is to simplify the implementation of the app base class when making calls to the database helper and web service helper objects. The methods within the app part utils class are publicly available; however, it is not necessary to call them directly.

7.3 Element and Element Utils

The element class is significant in that it represents an individual user interface component that is associated with an app part. The element class is responsible for fetching the element options that are necessary for the element if the user interface component type for the element is a combo box or an option group of radio buttons. Figure 17 illustrates the use case for the element class.

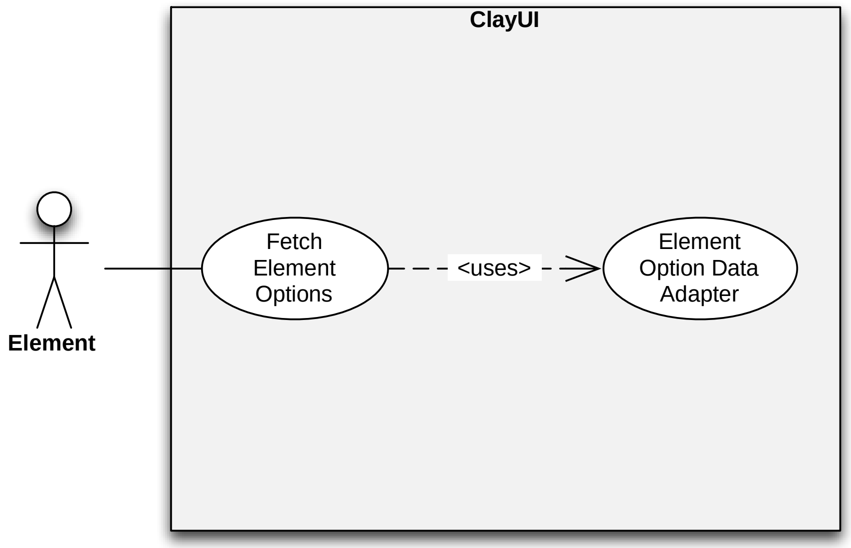


Figure 17 -- Element Use Case

Related to the element class is the element utils class. The purpose of this class is to simplify the implementation of the app base class when making calls to the database helper and web service helper objects. The methods within the element utils class are publicly available; however, it is not necessary to call them directly.

7.4 Implementation

One of the goals of the ClayUI API is to provide a simplified method for implementing a dynamic and adaptable user interface that performs common tasks such as data entry and information representation. As was mentioned above, we have accomplished this through the use of utility classes and a base class. The minimum requirement for the ClayUI API is to implement an instance of the ClayUI Base class and the ClayUI App Part class. In this section we will give an example implementation for use on the Android platform. The fundamentals of this implementation are applicable on other platforms; however, appropriate adaptation for the target platform is necessary.

The implementation on Android is as follows:

1. Define and instantiate a LinearLayout class.
2. Instantiate a ClayUI App Base class by passing the application ID from the ClayUI Application Administration website and the base URI for the ClayUI Application Administration website to the constructor.
3. Instantiate a new ClayUI App Part class by calling the getAppPart method from the App Base class, which returns an instance of the App Part, class. This method requires that the app part ID from the ClayUI Application Administration website is passed to the constructor.
4. Call the fetchElements method from the ClayUI App Part class by passing the method a Context object.
5. Call the refreshLayout method from the ClayUI AppPart class by passing the method the LinearLayout class defined previously and a Context class.
6. Override the onResume method of the Application Activity and add the ClayUI App Base openDB method to open the SQLite database when the activity starts.
7. Override the onPause method of the Application Activity and add the ClayUI App Base closeDB method to close the SQLite database when the activity is closed or pauses.

At this point the application should be functional; however, not very useful. The developer needs to provide a function for implementing the synchronization methods of ClayUI which saves form data to the local database and uses a the web service to send the local database records to the ClayUI remote database. The ClayUI API provides the necessary methods for these functions; however, it is up to the developer to design an appropriate implementation.

In the following example, we chose to implement the synchronization function of ClayUI by adding a menu to the mobile application. The following steps illustrate this implementation.

1. Override the onCreateOptionsMenu method of the default Android Activity and add menu options for the function of syncing the app part schema with the ClayUI web service, saving local database records to the ClayUI remote database, and saving the app part data from the form to the local database as follows:

CODE: ---

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
      menu.add(Menu.NONE, 1, Menu.NONE, R.string.syncSchema);
      menu.add(Menu.NONE, 2, Menu.NONE, R.string.syncData);
      menu.add(Menu.NONE, 3, Menu.NONE, R.string.saveAppPart);
      return true;
    }
```

END CODE: ---

2. Add a method that will handle the function of calling the ClayUI API method for synchronizing the local database with the ClayUI web service. This method fetches the database records that control the UI layout in the mobile application. The implementation of this method is as follows:

CODE: ---

```
    private void syncSchema() {
        // sync with web service
        appBase.syncLayoutStructure();
        // fetch new elements from local database
        appPart.fetchElements(this);
        // refresh UI layout
        appPart.refreshLayout(appPartLayout, this);
        Toast.makeText(getApplicationContext(), "Layout updated",
            Toast.LENGTH_SHORT).show();
    }
```

END CODE: --

3. Add a method that will handle calling the ClayUI API method for saving the app part data from the form to the local database. The implementation of this method is as follows:

CODE: ---

```
    private void saveAppPart() {
            appBase.saveAppPartDataLocal(appPart, appPartLayout, this);
     }
```

END CODE: --

4. Add a method that will handle calling the ClayUI API method for sending the local database records to the ClayUI web service to insert in the remote database. The implementation of this method is as follows:

CODE: ---

```
    private void syncData() {
        appBase.saveAppPartDataWeb(appPart, this);
    }
END CODE: ------------------------------------------------------
```

5. Add the ability to call our methods to the menu items by overriding the onOptionsItemSelected method like the following:

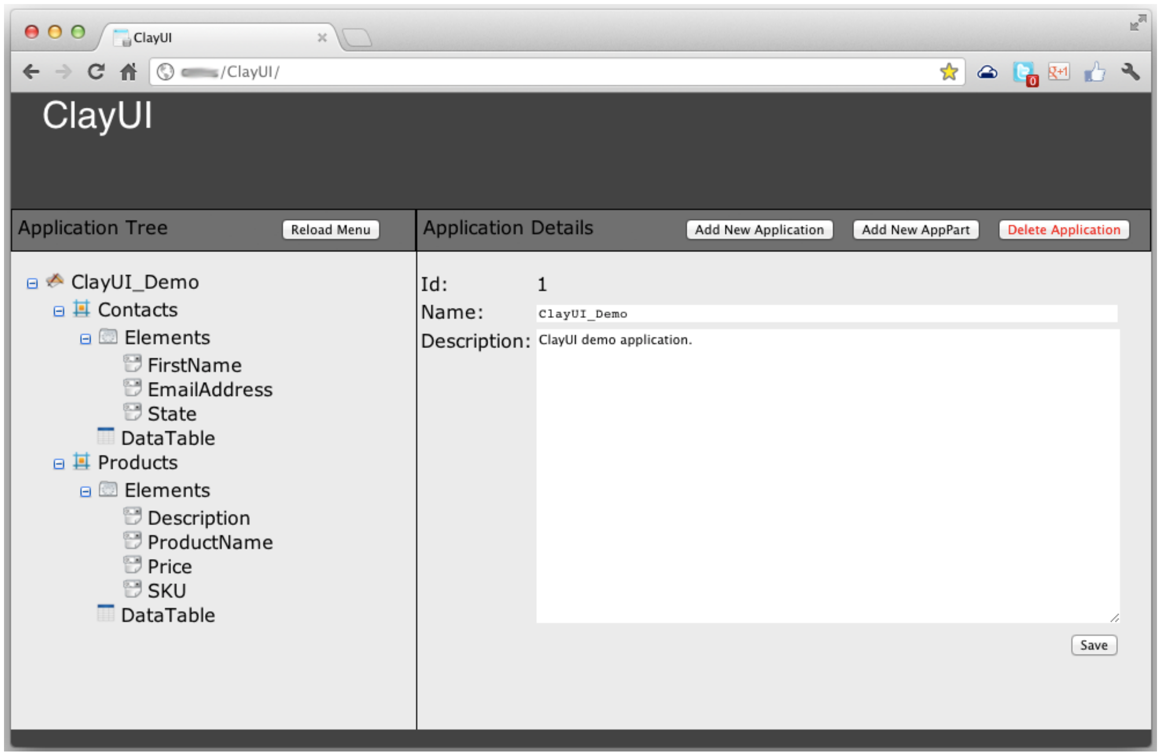
```
CODE: ----------------------------------------------------------
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // check menu ID
        switch (item.getItemId()) {
            case 1: this.syncSchema();
             break;
            case 2: this.syncData();
             break;
            case 3: this.saveAppPart();
             break;
        }
        return false;
    }
END CODE: ------------------------------------------------------
```

Chapter 8: Evaluation

In order to evaluate the usability of the ClayUI API, we developed a prototype app that demonstrates the capabilities of the ClayUI API. In the following sections, we will describe the prototype app and compare the implementation of the app on the Android and Windows Mobile platforms, which will demonstrate the flexibility of applying different design principals with the ClayUI app parts. Finally, we will provide some details on how the ClayUI API may simplify the development process for certain application functionality.

8.1 ClayUI Prototype

The mobile application we created to demonstrate the ClayUI API contains two ClayUI app parts. The first app part is called Contacts, and its purpose is to allow a user to enter a few pieces of information related to some contacts he wishes to track in his application. The second app part is called Products, and its purpose is to allow a user to enter a few pieces of information related to some products that he wishes to track in his application. From a broad overview, a use for such an application could be to enter purchased goods, and the sales representatives responsible for these goods. In Figure 18, we illustrate the ClayUI application tree that demonstrates this.



- ClayUI_Demo
  - Contacts
    - Elements
      - FirstName
      - EmailAddress
      - State
    - DataTable
  - Products
    - Elements
      - Description
      - ProductName
      - Price
      - SKU
    - DataTable

Id: 1
Name: ClayUI_Demo
Description: ClayUI demo application.

Figure 18 -- ClayUI prototype Application Tree

When we first configured the ClayUI prototype, we configured all of the user interface element types as basic text boxes. We did this in order to demonstrate the process of synchronizing the ClayUI configuration database with the remote ClayUI database and letting the ClayUI API refresh the layout. In Figure 19 we illustrate the resulting application implemented in Android on the left and Windows Mobile 6 on the right.

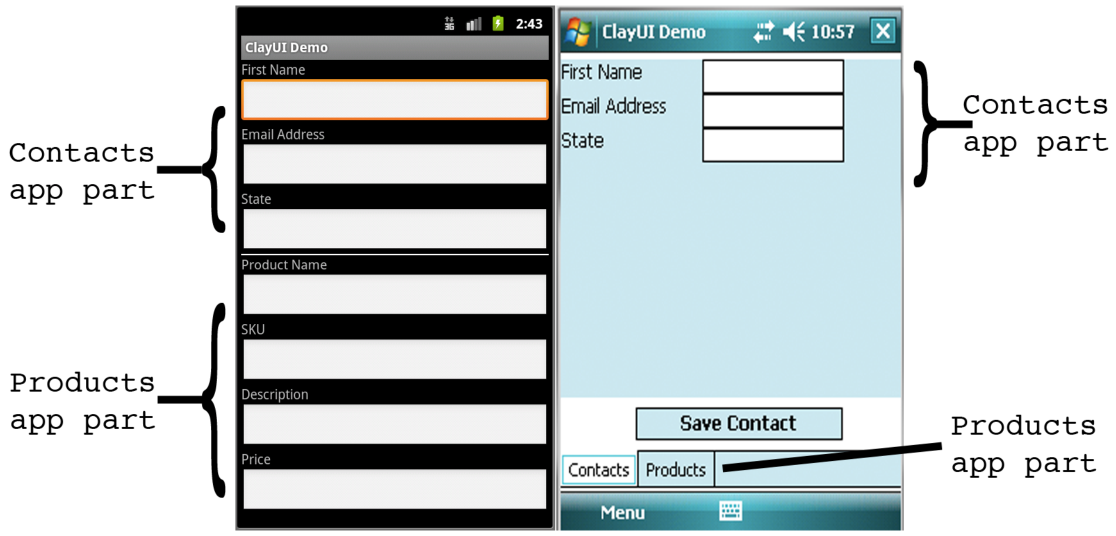


Figure 19 -- ClayUI Prototype (Android and Windows Mobile implementations)

In the example shown, we separated the two app parts with a horizontal bar to differentiate between the two on the Android implementation. Whereas in the Windows Mobile implementation, we used tab controls to separate the app parts. In the Android implementation prototype app, the only user interface elements that required manual configuration were the placement of the FlowLayout views and the horizontal separation bar. In the Windows Mobile implementation prototype app, the only user interface elements that required manual configuration were the FlowLayout panels on, the tab controls, and the buttons. The ClayUI API creates the remaining user interface elements at runtime.

To demonstrate the ability of ClayUI to modify the user interface at runtime, we changed some of the user interface types in the administration application. We changed the state and description fields to combo boxes and the price field to a radio group.

After making a call to the ClayUI API's method to synchronize the local ClayUI configuration database with the remote database, our application makes a call to the ClayUI API's refresh layout method. These functions are executed via a menu selection in the Android implementation and the Windows Mobile implementation. The resulting user interface changes illustrated in the Figure 20 were accomplished without any code changes from the developer.

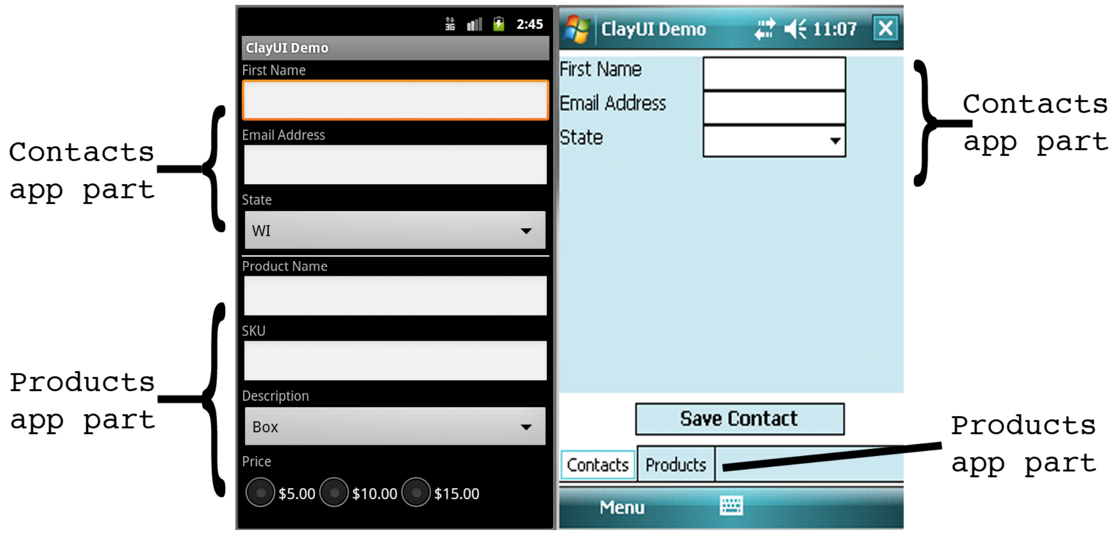
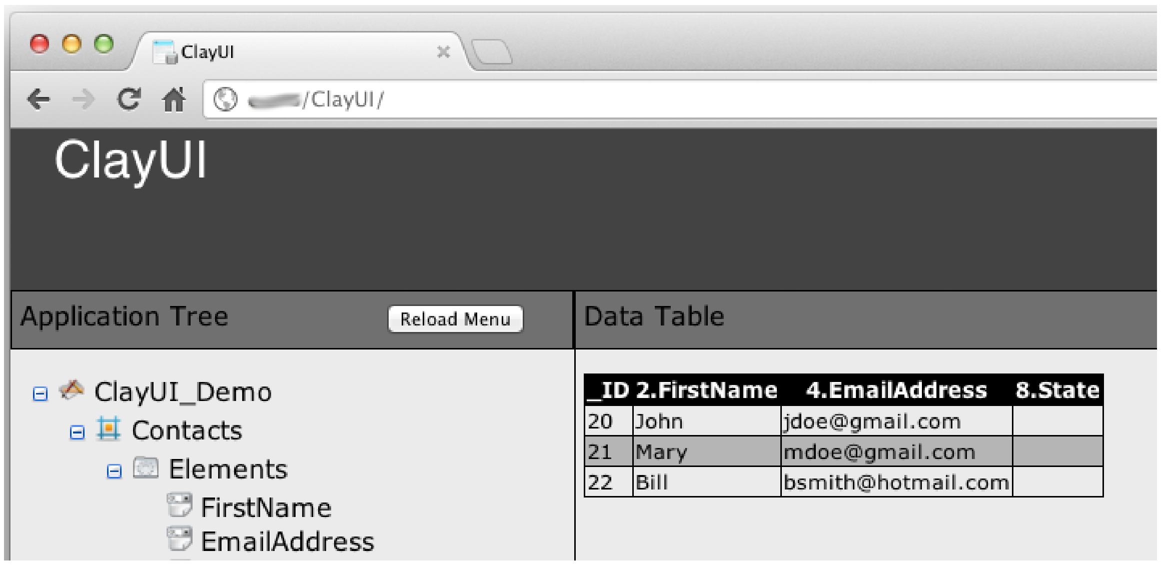


Figure 20 -- Resulting user interface

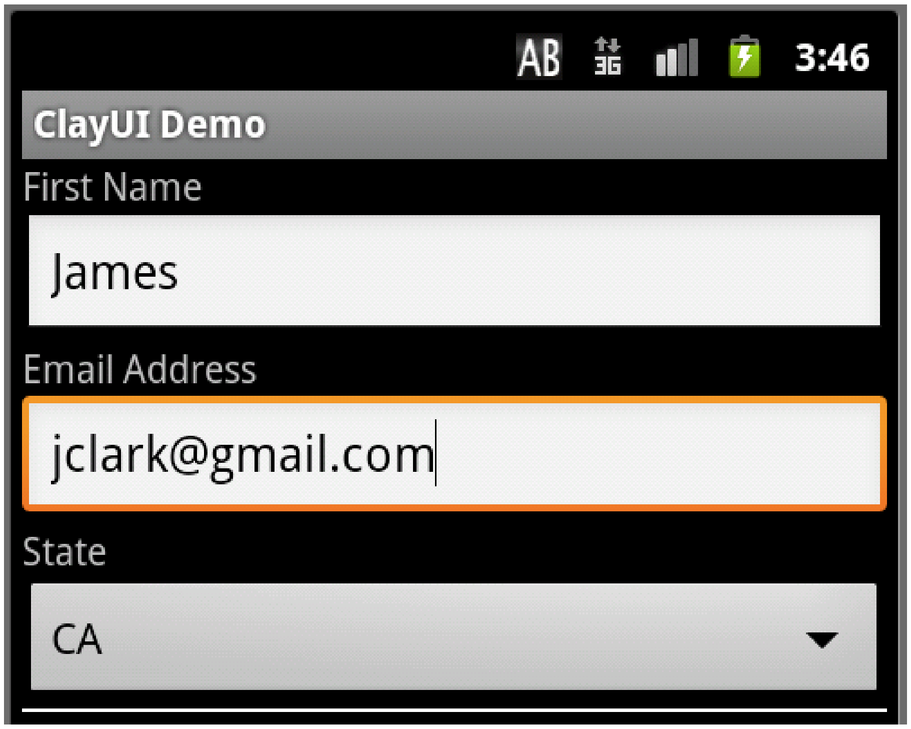
Another feature of the ClayUI API is the ability to save records to the local database as well as push the records to a remote database using a web service. To demonstrate this using the Android implementation, we will enter a new contact record, save it to the local database and then post it to the web service. Figure 21 shows that in our remote data table, we have three contact records.



_ID	2.FirstName	4.EmailAddress	8.State
20	John	jdoe@gmail.com	
21	Mary	mdoe@gmail.com	
22	Bill	bsmith@hotmail.com	

Figure 21 -- Contacts data table before record addition

Within the ClayUI prototype app on Android, we enter some contact information as illustrated in Figure 22.



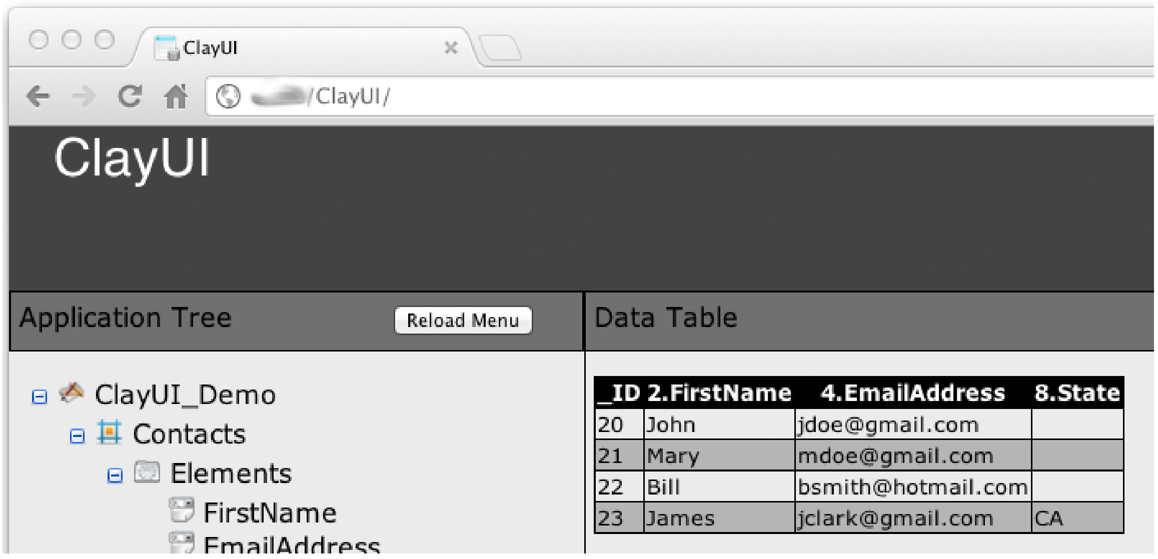
First Name: James

Email Address: jclark@gmail.com

State: CA

Figure 22 -- Entering contact information

Once we have finished entering our contact information, we need to save the form data to the local database and subsequently push the data to the ClayUI web service. In our Android application, this is accomplished using two menu functions that call the ClayUI API methods to carry out the tasks. Once the records are successfully written to the remote database, we can verify the successful operation by refreshing the ClayUI administration application data table page. This result is illustrated in the figure 23.



_ID	2.FirstName	4.EmailAddress	8.State
20	John	jdoe@gmail.com	
21	Mary	mdoe@gmail.com	
22	Bill	bsmith@hotmail.com	
23	James	jclark@gmail.com	CA

Figure 23 -- Successful web service post result

8.2 Ease of implementation

In the previous section we demonstrated the implementation of the ClayUI API on the Android platform using a prototype app. In this app we provided an example form which contained user interface elements and we provided facilities to save the form data to the local database and subsequently post the local database records to a web service so that it is stored in a remote database. In this section we will demonstrate the benefits of the ClayUI API by comparing our prototype application to an example application that accomplishes the same function without using the ClayUI API.

For comparison, we duplicated the ClayUI prototype application without the use of the ClayUI API. This mobile app is illustrated in Figure 24.

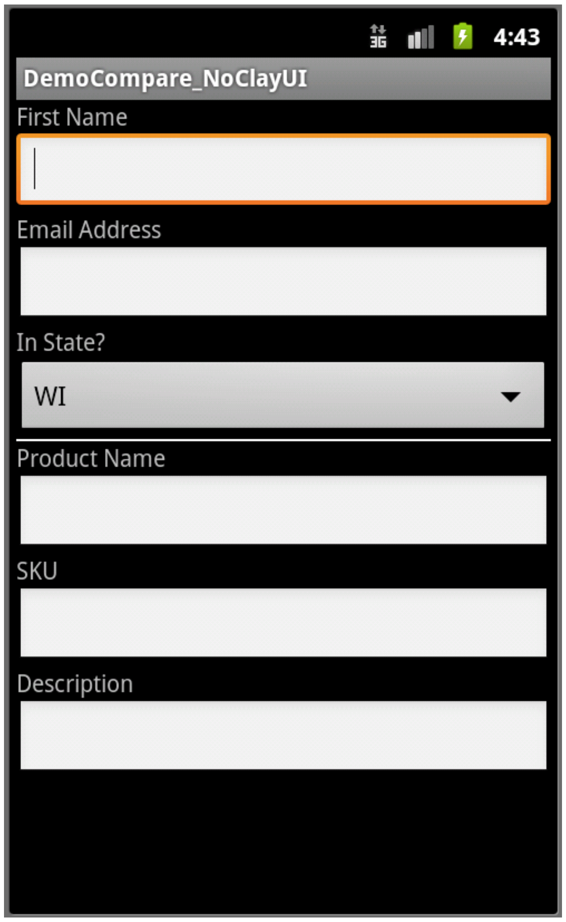


Figure 24 - Native mobile app for comparison

In order to accomplish this, we needed to manually create all of the user interface elements and develop the methods for creating the local database, saving to the form data to the local database and posting the local database records to the web service. This ultimately required significantly more development time for this simple mobile app.

Figure 25 provides a comparison of the development of these two similar mobile apps on the Android platform.

Comparison	Native Implementation	ClayUI API
Approximate time to develop	2 hours	30 minutes
Total # of classes	7	1
Total lines of code	618	145
Knowledge of process for writing to local database and web service	Yes	No

Figure 25 - Native app vs. ClayUI API app

The target audience of the ClayUI web application is application administrators that may or may not have experience developing mobile applications while the audience of the API is mobile application developers. To get feedback from these target audiences, a demonstration of the ClayUI web application and API was given to a group of four application developers and two application administrators. After the demonstration, all participants were given a survey to assess their reactions to ClayUI. Both groups were asked to answer the following questions regarding the ClayUI web application:

1. What is your overall rating of the ClayUI Administrative web application?
2. How easy did you find the navigation of the user interface?
3. How would you rate the ease of setting up a new application?
4. How would you rate the ease of configuring user interface elements in existing App Parts?

5. How would you rate the need to expand Claus's support to include more data types and user interface element types?

All questions were evaluated on a scale of one to five. One was considered the least favorable while five was considered the most. Figure 26 illustrates the result of these questions, the number provided represents the mean of all scores:

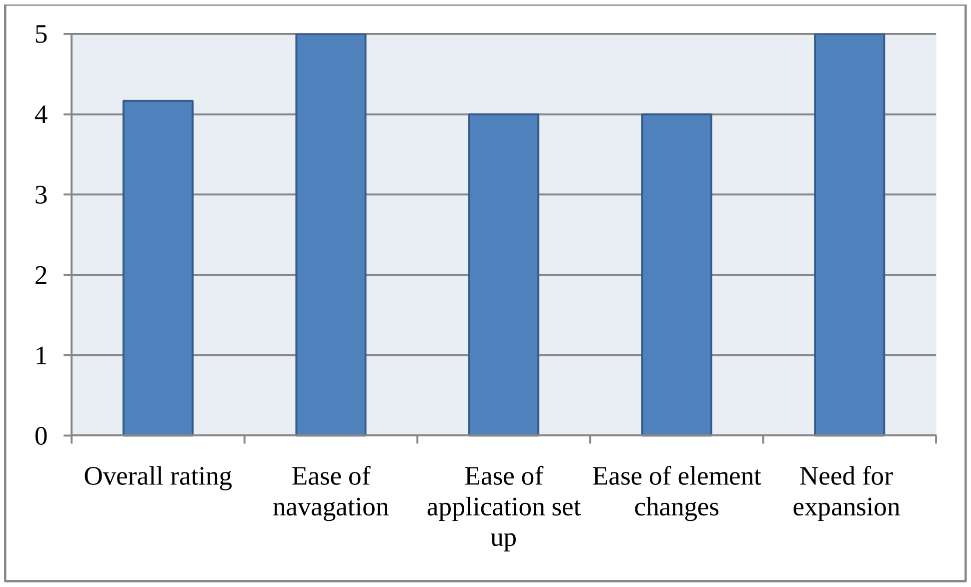


Figure 26 -- ClayUI Administration Web Application Survey Results

The application development group was also asked the following questions regarding the ClayUI API:

1. What is your overall rating of the ClayUI API?
2. How easy did you find the implementation of the API?
3. How would you rate the flexibility of using the API in the design of a mobile application?
4. How would you rate the usefulness of this API?
5. How would you rate the need to expand the ClayUI API to support more data types, user interface element types and mobile platforms?

All questions were evaluated on a scale of one to five. One was considered the least favorable while five was considered the most. Figure 27 illustrates the result of these questions; the number provided represents the mean of all scores:

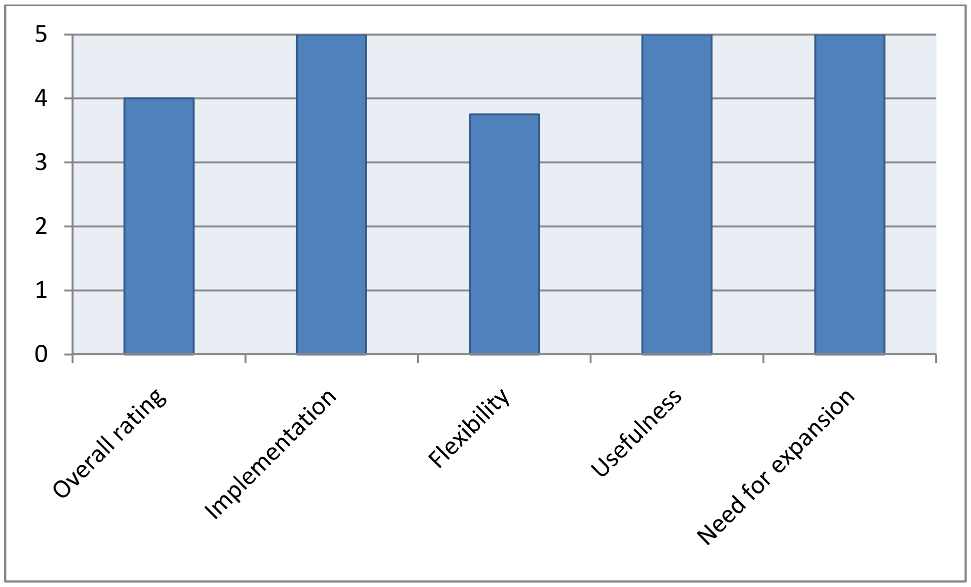


Figure 27 -- ClayUI API Survey Results

Chapter 9: Conclusion and Future Work

In this thesis, we have demonstrated a design pattern, framework and API that assists with the development of mobile apps that adhere to their respective design guidelines. In the next sections we will summarize the design and approach we used in this paper, describe the contribution of this thesis, evaluate the short and long term impact of this thesis and propose the direction for future work.

9.1 Summary

ClayUI is a framework and API that is designed to assist developers with the process of building mobile apps that target multiple mobile operating system platforms. We have given evidence that the mobile computing environment is growing in the number of users as well as the number of mobile operating platforms. In addition to this, the number of different mobile platforms support different native programming languages and design guidelines. ClayUI assists developers who design mobile apps that target different mobile platforms by allowing the developer to adhere to the design guidelines of the platform. In addition to this, our framework and API assist with the process of creating a local database for saving user generated data and posting this data to a database enabled web service where additional applications may access it.

The distinction of our solution is that it aids a developer with the construction of segments of an overall mobile application while providing flexibility in the design of the application so that the developer may apply the design guidelines of the platform.   Our solution provides an API that is written in the native programming language of the target platform. This allows the developer to use the appropriate Software Development Kits and any features and benefits that may be specific to the target platforms. With our solution, there is no reason to cut out a feature because it may not be supported on another platform.

9.2 Contribution of this thesis

1. Working API for flexible and cross platform user interface design: as the mobile computing industry continues to grow in divergence with respect to the mobile operating systems used, it will become increasingly difficult to create mobile applications that target these systems and continue to provide an user experience that is fluid and expected with respect to the rest of the platform. Using an API and design pattern for common user interface elements, a developer has the flexibility to apply platform specific design principals.
2. Working API for a common system of storing and retrieving application data: storing data in a local database as well as utilizing web services for posting and retrieving data is a common task in mobile application development. When a developer creates an application on multiple platforms, there is a chance the platforms use different technologies for these tasks. An API that provides this functionality for a developer reduces the need for a developer to learn the different methods of accomplishing these tasks.
3. Applying distributed computing design patterns to mobile application design: one of the benefits of mobile computing is that it helps to keep us connected and allows us to access data on the move. The application of distributed computing

design patterns in the design of mobile applications helps to provide more robust systems that have the capacity to deliver more value to the consumer. Using these principals, a mobile app becomes a part of an overall system instead of an island of information.

9.3 Short and long-term impact

The growth of mobile devices in our daily lives is staggering. Manufacturers of these devices continue to develop their own mobile operating platforms and developers continue to develop mobile apps that enrich our daily activities. As developers attempt to target a larger audience of users using these mobile operating platforms, they will struggle to maintain applications that fit seamlessly into their target platforms. It is our belief that the framework and APIs that we developed add value to the overall effort of developing these mobile applications.

In the long term, mobile applications will regularly access information through web services and cloud services to provide functionality not yet attempted. The framework we developed helps to alleviate the learning curve for utilizing these services by providing simple to use methods that are implemented using similar procedures on all platforms. The developer does not need to know the underlying database technology used, or the connectivity methods used for connecting to web services.

Overall, the use of our framework and API could reduce the cost to develop an application that targets multiple platforms. Unit tests performed on the API would reduce the amount of bugs introduced into systems by developers that do not fully understand the proper implementation of the features that the API provides. Additionally, developers could bring their mobile apps to a larger audience of users by targeting more mobile platforms while maintaining the look and feel intended by the design guidelines of the target platform.

9.4 Future work

At the time of this paper the ClayUI only supports a fraction of mobile platforms. ClayUI also only supports a fraction of the layout types and user interface element types. In order for wider adoption, ClayUI should be expanded to support, at least, the most popular mobile platforms. Additionally, ClayUI should be extended to include more of the widely used user interface layouts and controls.

In addition to the core functionality additions that ClayUI needs, no considerations for security was included in this project. The ClayUI Administration Application and web services should be secured using SSL, and the applications that are configured in ClayUI should be secured using a combination of username and passwords or digital certificates.

Future expansion of this project should also include a seamless method of deploying the backend web server, PHP scripts and MySQL database so that potential users of the framework could deploy a private version of the framework. As of this writing, there is only a single server sized appropriately for testing.

BIBLIOGRAPHY

[Albanesius11] C. Albanesius (2011, February 8). *Smartphone Shipments Surpass PCs for First Time. What's Next?* [Online]. Available: http://www.pcmag.com/article2/0,2817,2379665,00.asp

[Android1] (2012, June 13). *What is Android?* [Online]. Available: http://developer.android.com/guide/basics/what-is-android.html

[Apple07] (2007, January 9). *Apple Reinvents the Phone with iPhone* [Online]. Available: http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html

[Apple08] (2008, January 11) *Revolutionary Phone* [Online]. Available: http://web.archive.org/web/20080111051348/http://www.apple.com/iphone/features/index.html

[Apple10] (2010, March 5). *iPad Available in US on April 3* [Online]. Available: http://www.apple.com/pr/library/2010/03/05iPad-Available-in-US-on-April-3.html

[Apple12] (2012). *Develop for iOS: The world's most advanced mobile platform* [Online]. Available: https://developer.apple.com/technologies/ios/

[Bellsouth1] (1993, November 8). *Bellsouth, IBM unveil personal communicator phone* [Online]. Available: http://findarticles.com/p/articles/mi_m3457/is_n43_v11/ai_14297997/?tag=content;col1

[Block08] R. Block (2008, March 6). *Live from Apple's iPhone SDK press conference* [Online]. Available: http://www.engadget.com/2008/03/06/live-from-apples-iphone-press-conference/

[Capra02] L. Capra; E. Wolfgang; G. Blair; P. Grace; P. Mascolo, "Exploiting reflection in mobile computing middleware" *Mobile Computing and Communications Review*, vo. 6, no. 4, pp34 – 34, January 2002.

[Chien-Liang05] F. Chien-Liang; , G.-C. Roman; L. Chenyang; , "Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications," *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on* , vol., no., pp.653-662, 10-10 June 2005 doi: 10.1109/ICDCS.2005.63

[Cole03] A. Cole; S. Duri; J. Munson; J. Murdock; D. Wood, "Adaptive service binding middleware to support mobility," *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on* , vol., no., pp. 369- 374, 19-22 May 2003 doi:10.1109/ICDCSW.2003.1203581

[Eaton10] N. Eaton (2010, January 27). *Apple iPad: Your Impressions?* [Online]. Available: http://blog.seattlepi.com/microsoft/2010/01/27/apple-ipad-your-impressions/

[Fox98] A. Fox, S. Gribble, Y. Chawathe and E. Brewer, “Adapting to networks and client variation using infrastructural proxies: Lessons and perspectives,” *IEEE Pers. Commun.*, August, 1998, pp. 10-19, Aug., 1998.

[Fried11] I. Fried (2011, July 27). *Annual Smartphone Sales Could Reach One Billion by 2016* [Online]. Available: http://allthingsd.com/20110727/annual-smartphone-sales-could-reach-1-billion-by-2016/

[Grace04] P. Grace, “Overcoming middleware heterogeneity in mobile computing applications,” Ph.D. dissertation, Computing Department, Lancaster University, Lancaster, England, 2004.

[Haahr00] M. Haahr; R. Cunningham; V. Cahill , "Towards a generic architecture for mobile object-oriented applications," *Service Portability and Virtual Customer Environments, 2000 IEEE* , vol., no., pp.91-96, 2000 doi: 10.1109/SPVCE.2000.934166

[Haywood11] J. Haywood (2011, July 14). *Media Tablets Have the Opportunity to Transform the Enterprise: IDC Canada Examines Media Tablet Use in Canadian Business* [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prCA22937111

[IDC12] (2012, March 13). *Media Tablet Shipments Outpace Fourth Quarter Targets; Strong Demand for New iPad and Other Forthcoming Products Leads to Increase in 2012 Forecast, According to IDC* [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23371312

[Markoff99] J. Markoff (1999, August 30). *Microsoft Brings In Top Talent To Pursue Old Goal: The Tablet* [Online]. Available: http://www.nytimes.com/1999/08/30/business/microsoft-brings-in-top-talent-to-pursue-old-goal-the-tablet.html?pagewanted=all&src=pm

[Markoff07] J. Markoff (2007, Nov. 4). *I, Robot: The Man Behind the Google Phone* [Online]. Available: http://www.nytimes.com/2007/11/04/technology/04google.html?pagewanted=all

[Miller11] H. Miller (2011, January 18). *Tablet-Computer Sales to Triple This Year, IDC Says* [Online]. Available: http://www.bloomberg.com/news/2011-01-18/tablet-computer-sales-to-triple-to-44-6-million-units-this-year-idc-says.html

[Murarasu09] A.F. Murarasu; T. Magedanz, "Mobile Middleware Solution for Automatic Reconfiguration of Applications," *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on* , vol., no., pp.1049-1055, 27-29 April 2009 doi: 10.1109/ITNG.2009.194 [Nagamine11a] K. Nagamine (2011, January 12). *PC Market Records Modest Gains During Fourth Quarter of 2010, According to IDC* [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS22653511

[Nagamine11b] K. Nagamine (2011, March 29). *IDC Forecasts Worldwide Smartphone Market to Grow by Nearly 50% in 2011* [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS22762811 [Nagamine12a] K. Nagamine (2012, April 11). *PC Market Returns To Positive Growth, Though Gains Remain Small, According to IDC* [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23433412

[Nagamine12b] K. Nagamine (2012, February 6). *Smartphone Market Hits All-Time Quarterly High Due To Seasonal Strength and Wider Variety of Offerings, According to IDC* [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23299912

[Nagamine12c] K. Nagamine (2012, January 11). *PC Market Stumbles on HDD Shortage While U.S. Market Sees Worst Annual Growth Since 2001, According to IDC* [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23261412

[Pettey10] C. Pettey; L. Goasduff (2010, October 15). *Gartner says worldwide media tablet sales on pace to reach 19.5 million units in 2010.* [Online]. Available: http://www.gartner.com/it/page.jsp?id=1452614

[Pettey11] C. Pettey (2011, September 8). *Gartner says PC shipments to slow to3.8 percent growth in 2011; Units to increase 10.9 percent in 2012.* [Online]. Available: http://www.gartner.com/it/page.jsp?id=1786014

[Reed11] B. Reed (2011, December 23). *Enterprise smartphone and tablet incursion to grow in 2012* [Online]. Available: http://www.networkworld.com/news/2011/122311-outlook-smartphone-tablet-254341.html

[Rhomobile1] (2011, September 11). *Smartphone apps made easy* [Online]. Available: http://rhomobile.com

[Rodriguez08] A. Rodriguez (2008, Nov 6). *RESTFUL Web services: The basics.*[Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-restful/

[Rubin07] A. Rubin (2007, November 5). *Where's my Gphone?* [Online]. Available: http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html

[Savov11] V. Savov (2011, Jan. 22). *Motorola Xoom Launching February 17th at Best Buy (update: price at $700).* [Online]. Available: http://www.engadget.com/2011/01/22/motorola-xoom-launching-february-17th-at-best-buy/

[Schapira01a] A. Schapira.; K. De Vries; C. Pedregal-Martin, "MANIC: an open-source system to create and deliver courses over the Internet," *Applications and the Internet Workshops, 2001. Proceedings. 2001 Symposium on* , vol., no., pp.21-26, 2001 doi: 10.1109/SAINTW.2001.998204

[Schapiara01b] A. Schapira.; K. De Vries; C. Pedregal-Martin, "MANIC: an open-source system to create and deliver courses over the Internet," *Applications and the Internet Workshops, 2001. Proceedings. 2001 Symposium on* , vol., no., pp.21-26, 2001 doi: 10.1109/SAINTW.2001.998204

[T-Mobile08] (2008, September 23). *T-Mobile Unveils the T-Mobile G1 – the First Phone Powered by Android* [Online]. Available: **http://tinyurl.com/7wqobga**

[Vaughan-Nichols04] S.J. Vaughan-Nichols, "Wireless middleware: glue for the mobile infrastructure," *Computer* , vol.37, no.5, pp. 18- 20, May 2004 doi: 10.1109/MC.2004.1297229

[Webster12] (2012, June 6). *Smartphone* [Online]. Available: http://www.merriam-webster.com/dictionary/smartphone

[Zhang04] H. Zhang and W. Ma, “Adaptive content delivery on mobile internet across multiple form factors,” *Multimedia Modeling Conference*, 2004. Proceedings, 10th International, vol., no., pp. 8, 5-7 Jan. 2004. doi: 10.1109/MULMM.2004.1264960

[Zhigang06] H. Zhigang; X. Xing; L. Hao; L. Hanqing; M. Wei-Ying, "Design and Performance Studies of an Adaptive Scheme for Serving Dynamic Web Content in a Mobile Computing Environment," *Mobile Computing, IEEE Transactions on* , vol.5, no.12, pp.1650-1662, Dec. 2006. doi: 10.1109/TMC.2006.182

Appendix A

Table of Definitions

Term	Definition
Android	Mobile operating system developed by Google for use with their smartphone and tablet computers
Apache	A widely used open source web server
API	Application Programming Interface
Cloud Computing	System for delivering computing resources as a service
CSS3	Third version of the Cascading Style Sheets document descriptor
Design Pattern	A programatic solution that is reusable in future software designs
Distributed computing	Computing environment where two or more devices share the burden of processing and displaying computational applications
Framework	Structure of services and resusable libraries to implement an application
HTML5	Fifth version of the HTML markup language used for creating dynamic web applications
iOS	Mobile operating system developed by Apple for use with their smartphones and tablet computers
Java	Native programming language used for developing Android applications
JavaScript	Scripting language similar to Java used to add dynamic content to web applications
JSON	JavaScript Object Notation, a text-based data descriptor for data interchange
LAMP	Linux - Apache - MySQL - PHP
Linux	Open general purpose operating system
MySQL	A widely used open source database server engine
Objective-C	Native programming language used for developing iOS applications
Pervasive Computing	Computing using technology which allows for remote access of information and services
PHP	Server-side scripting language used to build dynamic and data-driven web applications
REST	Representational State Transfer, an architecture used for distrubted systems
SDK	Software Development Kit
Sensor Networks	Computing environment built from distributed autonomous sensors that monitor environmental conditions
Smartphone	A mobile computing device that has a telephony radio interface
SQLCE	A database engine based on Microsoft SQL Server optimized for portability
SQLite	An open source database engine optimized for portability

Tablet Computer	A mobile computing device that is optimized for a touch user interface
Web Service	System of retrieving and saving data using standard HTTP GET and POST methods
XML	eXtensible Markup Language, a text based descriptor for data interchange

Appendix B

The source code for this project is available at

https://docs.google.com/folder/d/0BylubvV-Xo2FQWhkdXc2azY1Q1U/edit