CMobile: A Mobile Photo Capture Application for Construction Imaging

Andrew Martin
martinandrewj@gmail.com

Ron Vetter
vetterr@uncw.edu

Department of Computer Science,
University North Carolina Wilmington
Wilmington, North Carolina 28403 USA

Jeff Brown
brownj@uncw.edu
Department of Mathematics and Statistics
University North Carolina Wilmington
Wilmington, North Carolina 28403 USA

Thomas Janicki
janickit@uncw.edu
Dept. of Information Systems & Operations Management
University North Carolina Wilmington
Wilmington, North Carolina 28403 USA

Abstract

In recent years the mobile application space has exploded in popularity, a fact which is reflected in the increasing availability of both free and paid applications on a variety of mobile platforms. In order to take advantage of this ever-growing market, the authors developed a mobile photo capture application, called CMobile, to supplement data gathering for a project/content management system. This paper describes the original design requirements and features of the application, the methodology by which design choices were tracked and implemented, reviews the issues and problems encountered, discusses the resolutions employed and lessons learned, and concludes with a discussion of potential future developments.

Keywords: mobile application development, web services, agile development methodology

1. INTRODUCTION

Construction Imaging (CI), a leader in industry specific content management solutions, has been looking for opportunities to extend its suite of desktop product solutions into the exploding arena of mobile applications and devices (Construction Imaging, 2011).

CI has received numerous requests to enhance its content management model to include photograph management. The types of photographs project managers on site tend to take include images of completed or in-progress construction, safety violations or various impediments to job progress.

Currently, these photos must be extracted from the camera and then uploaded to a computer where they can be manually indexed into CI's content management system either via a desktop or web application. At times the project manager (user) might be out in the field and unable to immediately access a workstation to index the photographs. If the user is using a camera and needs to upload the photos immediately, he/she would have to find a suitable location to index the photos. If the user has a web enabled camera-phone there's an easier solution, but still far from ideal, namely e-mail. Finally, another option is to simply send the images to a third-party to index them.

An optimal solution would be for the user to take photos using their smart phone and automatically upload them to the system with a set of index values populated with data such as the phone's GPS coordinates, a related job number, vendor number and other values. Thus, CI partnered with the authors to develop a solution in the mobile application space, specifically an application designed to take advantage of the features provided by Apple's iPhone 4 (Apple Inc, 2011).

This paper discusses the software development methodology, systems analysis and design, and implementation details of a system called CMobile that allows users to interface with and add photographic content automatically to CI's content management system via his or her iPhone.

The paper describes the specific requirements for the one firm, however the implementation of the analysis and design plus the tool set employed and discussed may be used in other similar photo applications for mobile devices.

2. REQUIREMENTS

Currently, there is only one mobile application on the market that performs the desired customer requirements related to photo management for content management services. Vela Systems provides a product called Vela Mobile that includes an application for the iPad, but the application only interfaces with the proprietary Vela Field Management Suite (Vela Systems, 2011). In discussions with CI's management, it was decided to develop a proprietary product that interfaces only with CI's Content Manager, thereby giving the company complete control over branding and the ability to fully optimize the graphical user interface through seamless integration with their existing product. Management also desired to enhance CI's market position by being able to provide current and potential customers with a mobile application that would assist with data collection in the field.

The following is the original list of requirements for CMobile as identified by CI management and the application's developers (paper authors):

a) Required features:
- CMobile needs to integrate with the phone's camera, allowing the user to take a photograph directly from the application.
- The photograph must be processed (compressed if necessary) with index values associated with it from various static and configurable criteria including but not limited to the phone's number, GPS coordinates and an associated job number, and other configurable values predefined for the content type or selectable from a list of keywords.
- An option to upload a photograph and its accompanying index values via a configurable web service interface to CI's content management system must be provided.
- An option must be provided to allow a photograph and its index values to be emailed directly from the application.
- The application must have the ability to upload photographs that exist on the phone but were not taken from within the mobile application.

- Application access must be restricted by login credentials validated against the web service, however, the username and password may be saved locally for quick login.

b) Optional features:
- The ability to upload/email a batch of photographs with a single set of index values.
- The ability to attach a voice note or other recording.

3. DEVELOPMENT METHODOLOGY

In order to produce a mobile photo management application that achieved the objectives set forth for this project, the system was implemented using the following methodologies and technologies:

- Scrum Agile Development Methodology (Scrum Alliance, 2011).
- An Apple Macintosh computer using the iOS development platform provided by the Xcode Integrated Development Environment (Apple Developer, 2011).
- Objective-C using Interface Builder for the user interface development (Apple Developer, 2011).
- Communication with CI's Content Manager (the backend) with a web service API created using Windows Communication Foundation (WCF) with a JSON (Java Script Object Notation) enabled endpoint (Microsoft WCF, 2011).
- All source code was versioned and managed using Microsoft Team Foundation Server (TFS) via the Team Explorer Everywhere command line tool for OSX (Microsoft, TFS, 2011).
- Development progress was recorded and tracked via Tasks in Microsoft Team Foundation Server's (TFS) development management tools.
- Functionality was tested by Microsoft Test Manager, which fully integrates with TFS (Microsoft, Testing, 2011).

The Scrum development methodology was chosen over other potential methods due to the fact that it is the primary methodology currently employed by the developers and CI. The need to quickly adapt to changing requirements that tend to shift the direction of a development project mid-stream requires an agile approach, as opposed to the sequential approach of the waterfall model or other iterative models of development. The Scrum methodology suited our needs best.

The decision to use Xcode with Objective-C and Interface Builder was reached as these technologies are the standard development tools for Apple's iOS environment. The decision to develop for the iPhone itself instead of other mobile platforms, such as Android, Blackberry, etc., was based on the overall marketability of Apple's product at this time, with potential to expand CMobile to other platforms in the future.

4. IMPLEMENTATION

When CMobile launches for the first time it prompts the user to configure application settings (Figure 1). The user must specify a URL that points to an exposed Content Manager Web service (this web service URL will be preloaded into the application upon download from the 'appstore'). Once the address is verified and the connection is established, the user is directed to a Login screen (Figure 2). Login credentials can be saved locally in the settings menu in order to bypass the login screen on subsequent use of the application.

Figure 1: The CMobile Settings Screen

After successful authentication, the user is presented with the home screen (Figure 3). This screen allows the user to take a photo, index or email a photo or image stored on the phone, or edit the application's settings.

Figure 2: The CMobile Login Screen

Figure 3: CMobile's main navigation screen

If the user chooses to take a photo they will be presented with the iPhone's camera, otherwise they can browse for a photo or image. Once the user has specified the image they wish to use they are returned to the home screen where they can choose to index or email the image. If the user opts to index the image they are presented with the index screen (Figure 4).

Figure 4: Indexing an image

The index screen allows a user to index values associated with an image. Fields configured for GPS coordinates or heading are pre-populated from the phone if that data is available. Once the user has completed the input, the image can be uploaded. When the upload has completed the user is informed of its success and redirected to the home screen.

The process is similar if a user decides to index a photo already stored on the phone, except that the user is presented with the iPhone's camera roll where he/she can choose the images to index (Figure 5).

If the user decides to email an image instead of uploading it to the web service, they can follow the same steps, but this time they will be presented with the email screen (Figure 6). Here they can specify recipients, a subject and a message to send along with the attached image.

One thing to note is that once an image is indexed and uploaded to CI Content Manager via the WCF (Windows Communication Foundation) service it is no longer CMobile's concern. It may be saved, placed in a workflow, routed, trigger notifications, etc. CMobile's primary purpose is to create and present the data, not perform any other actions on it.

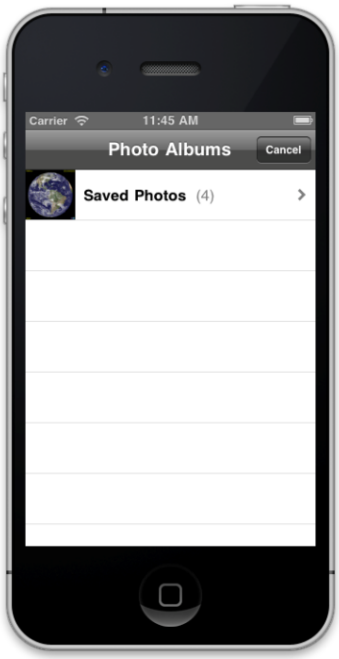

Figure 5: Browsing for an image

For version 1.0 of CMobile we have decided not to incorporate the ability to upload video or voice recordings. We also do not allow the user to upload batches of images under one set of index values.

Service Method Calls

We have implemented a web service API using WCF technology that performs some basic tasks for our web products, such as retrieving data source information, authenticating and authorizing user credentials, and importing and saving documents. We use this service to communicate with CI Content Manager.

The following is a list of preexisting service methods that were required for the application:

- **GetDataSources() -** A JSON enabled WebGet method that returns a list of available data sources to the user (JSON, 2011).

Figure 6: Indexing a photo

- **Login(string uname, string pass) -** A JSON enabled WebGet method that authenticates the user based on supplied username and password and begins the user session.

- **GetLayouts() -** A JSON enabled WebGet method that returns the layout of the content type. Includes configuration information about user settings and preferences and available fields and their configurations (name, data type, default value, etc.).

- **Logout() -** A JSON enabled WebGet method that ends the current user session.

- **IsLoggedIn() -** A JSON enabled WebGet method that determines if the user is still logged in and that the session has not expired. Required for instances where the user leaves the application running for an extended period of time. This method is typically called before making any other service calls that require the user session to be active.

The following is a list of new service methods that were implemented for the application:

- **GetNewDocument() -** A JSON enabled WebGet method that returns a stub of a Document object. In CMobile's case this object contains all the information required to import a photo: its data and index values. This particular method was added so that the JSON representation of the Document object would not have to be constructed programmatically from scratch in the application.

- **GetNewDocumentWithCount(int fieldCount, int fileCount) -** A JSON enabled WebGet method that returns a stub of a Document object. It is similar to the GetNewDocument() method but also returns the supplied count of stubs of Fields and Files properties on the Document object.

- **ImportDocumentWithImageStreamAsInvoke(Document document) -** A JSON enabled WebInvoke method that returns a document object that describes the photo to be imported into the content management system. This document contains the index field values and the image's data stream.

Figure 7 (in Appendix A) is a representation of the process by which CMobile communicates with the WCF Service API in order to submit content to the CI content management system. The seven service methods that CMobile calls are depicted under the category in which they operate, with the service methods listed in italics. Note that only the service methods required for the mobile application are listed. Figures 8 and 9 (in Appendix A) show a graphical view of the classes and their relationship to one another. Finally, Figures 10 and 11 (in Appendix A) show screen shots of the CMobile web and desktop interfaces.

5. DEVELOPMENT CHALLENGES

Fully implementing memory management was a major programming challenge encountered during implementation. Issues included understanding when an object needed to be retained as not to lose a reference to it later, when an object needed to be released in order to prevent memory leaks, and when an object should be ignored because another object currently required access to it.

Fortunately, Xcode provides both a code analyzer and a real-time leak detection tool. The analyzer can be run on the source code and offers information on locations where memory leaks are bound to occur, areas where leaks may occur, and sections where it was unnecessary to release objects. While the analyzer provides a lot of useful suggestions for handling memory, it can't always account for the flow of the application, so employing a leak detection tool assists with debugging in the Xcode simulator. It supplies real-time information about current memory allocations, including all introduced leaks.

One of the simplest tasks in CMobile's development was the incorporation of the iPhone's camera and email interface. Apple has made accessing these features simple and the integration seamless.

The most time consuming part of the development process was implementing the necessary service calls to the CI WCF service. After a few unsuccessful attempts at using various toolkits designed to communicate with non-RESTful services, the best approach was to enable certain service methods to return JSON formatted dictionaries and use the iOS JSON Framework to consume and convert these objects. Doing so made accessing data from the service as simple as constructing a RESTful URL and waiting for a response.

Even using the JSON Framework, we ran into a few problems trying to pass image data to the service. The first issue was that the service did not accept very large query strings. We updated the size the service would accept and were then able to successfully upload images. The second issue occurred when we realized that images over a certain size (about 1.3 megabytes) would exceed even the maximum allowable query string size. As a result we had to find another method to import the images. The solution was to enable the service methods to accept JSON web invoke calls. This allowed us to configure an HTTP POST message with the image data in the body of the message, bypassing the query string size limits. We were then able to import images of a much larger resolution and quality.

The implementation of the settings menu was completed by using InAppSettingsKit, an open

source solution by Edovia (2010). This toolkit allows for the easy inclusion of in-app settings or a duplication of the iPhone's application settings in your application.

All software development was focused on creating the application to run on the iPhone, however some of the decisions made along the way, such as the one to use the JSON format for communication, make the project more easily adaptable to other platforms. As discussed, the application communicates with Content Manager via an exposed WCF web service which acts as an API to all the functionality of the content management system itself. Once CMobile has passed its photo and index information to the web service, the content management system is free to perform any number of functions with the photo based on the data associated with it.

In order to verify that the requirements for the completion of this project were met we created test suites in Microsoft Test Manager. Each test suite is associated with a given story or task and has associated test cases that test the entire range of functionality implemented by the story's tasks.

Figure 12 (in Appendix A) displays a test designed to assess CMobile's ability to index a photo taken from the iPhone's camera, from login to success. Each test is comprised of multiple steps, each with their own expected outcome. As the test progresses, each step can be marked as having passed or failed. The test itself only passes if all of its steps have passed and then the code was considered complete, having met its requirements.

6. SUMMARY AND LESSONS LEARNED

CMobile was conceived and commissioned, after reviewing requests for enhancement from both current and potential customers, in order to establish CI as an enterprise content management provider in the mobile space. This allowed CI to expand its suite of products to what most consider the platform of the future in both enterprise content management and computing as a whole.

To further this goal we were charged with creating a simple yet powerful mobile photo capture application for the iPhone. To complete the task we leveraged the power of newer platforms and technologies and integrated them with more recognizable tools and systems in a completely seamless and unified fashion while maintaining both extensibility and adaptability on both sides.

The simplicity of the JSON protocol, when compared to standard XML formatted protocols, makes it ideal for use in situations that require the consumption of data passed over protocols like HTTP. Unlike XML, JSON does not require knowledge of a document type definition that the recipient understands, making its payload smaller and easier to parse, quicker to transmit (due to its smaller size), and generally easier to construct. JSON requires only simple evaluation of the text of the serialized string using a corresponding method in the given language.

The ease of taking non-RESTful, .NET-based WCF service methods and adapting them to send and receive JSON messages is an extremely useful and powerful lesson learned from this project. By doing little more than adding a service attribute, one can transform a service designed to work with specific .NET client architecture, such as C# applications, services and assemblies, or browser-embedded Silverlight applications, into a near universally accessible API.

Another important lesson learned was the need to understand the quirks and intricacies of Objective-C. Having to learn the uniqueness of the language's syntax and the methods by which memory is managed and the tools available to avoid or correct these issues allows developers to more effectively construct and deploy Objective-C applications on an iPhone.

The final key takeaway for all developers is the ease of integrating GPS with photos, the simplicity with which the iPhone allows said integrations, and the ease of using those features to expedite and simplify the process of transferring that information via a web service to a content management system.

While we are confident that we accomplished what we set out to do in CMobile Version 1.0, the product is far from finished as Version 1.1 is already underway. Ideas for additional features are abundant including, but not limited, to the following:

a) Reintroduction of batch image processing.
b) The ability to view images and documents in the user's work list and take action on them

(approve, reject, review, annotate, attach notes, etc).

c) The ability to view a mapping of all images containing GPS and heading index data in a given area.

This is just a small list of many possible opportunities to expand this project in the future. This first iteration of CMobile will provide a solid platform from which to move forward and realize the full potential of Construction Imaging solutions in the mobile space.

7. REFERENCES

Apple Developer (2011). Xcode - Developer Tools Technology Overview. Last retrieved 2/20/11. <http://developer.apple.com/technologies/tools/xcode.html>.

Apple Inc. (2011) Apple - iPhone 4 - Video calls, multitasking, HD video, and more. Last retrieved 7/9/2011. <http://www.apple.com/iphone/>.

Construction Imaging (2011). Construction Imaging Content Management Software & Systems Integration. Last retrieved 7/9/2011. <http://www.construction-imaging.com/>.

Edovia Inc. (2010) Edovia - Insanely Great Software for Mac and iOS. Last retrieved 7/9/2011. <http://www.edovia.com/>.

JSON. (2011), Last retrieved 7/9/2011. <http://www.json.org/>.

Microsoft. Team Foundation Server (2011). Last retrieved 7/9/2011. <http://msdn.microsoft.com/en-us/library/ms181238(v=vs.80).aspx>.

Microsoft. What Is Windows Communication Foundation. (2011). Last retrieved 7/9/2011. <http://msdn.microsoft.com/en-us/library/ms731082.aspx>.

Microsoft. What's New For Testing. (2011). Last retrieved 7/9/2011. <http://msdn.microsoft.com/en-us/library/bb385901.aspx>.

Scrum Alliance (2011). What is Scrum? Last retrieved 2/20/2011. <http://www.scrumalliance.org/learn_about_scrum>.

Vela Systems (2011). Fela Mobile iPad and Email iPad App for Construction Field Management. Last retrieved 3/3/2011. <http://www.velasystems.com/ipad-for-construction-vela-mobile/>.

Appendix A (Figures)

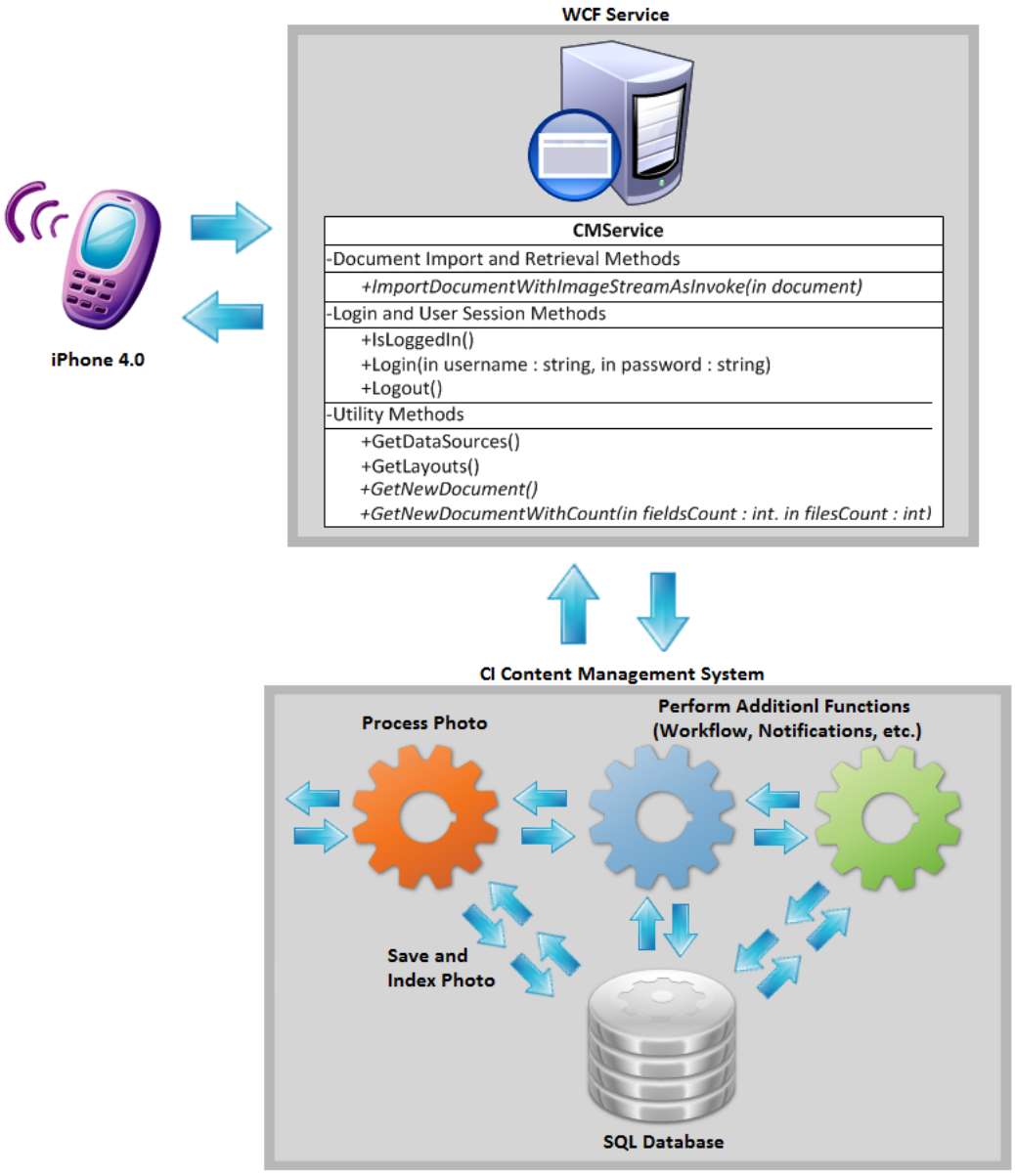

Figure 7: A graphical representation of CMobile and its integration with CI Content Manager

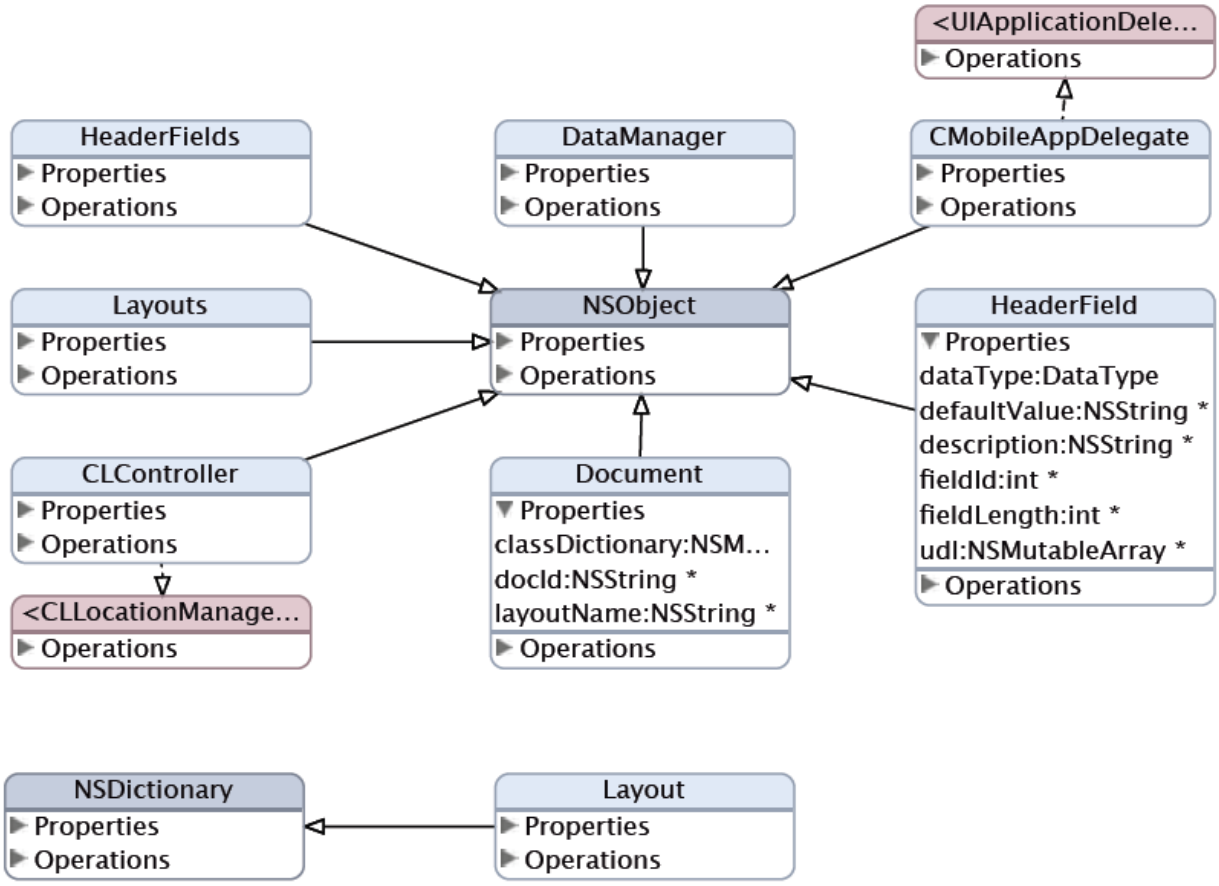

Figure 8: Class Model from Xcode

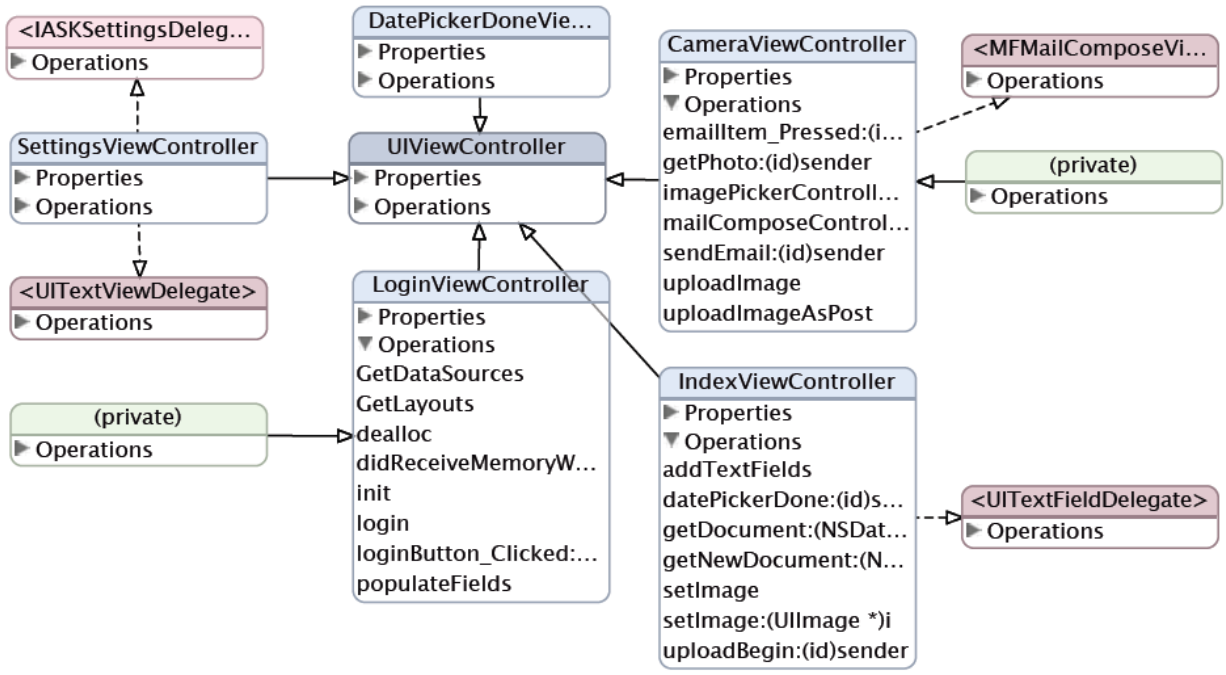

Figure 9: Class Model from Xcode

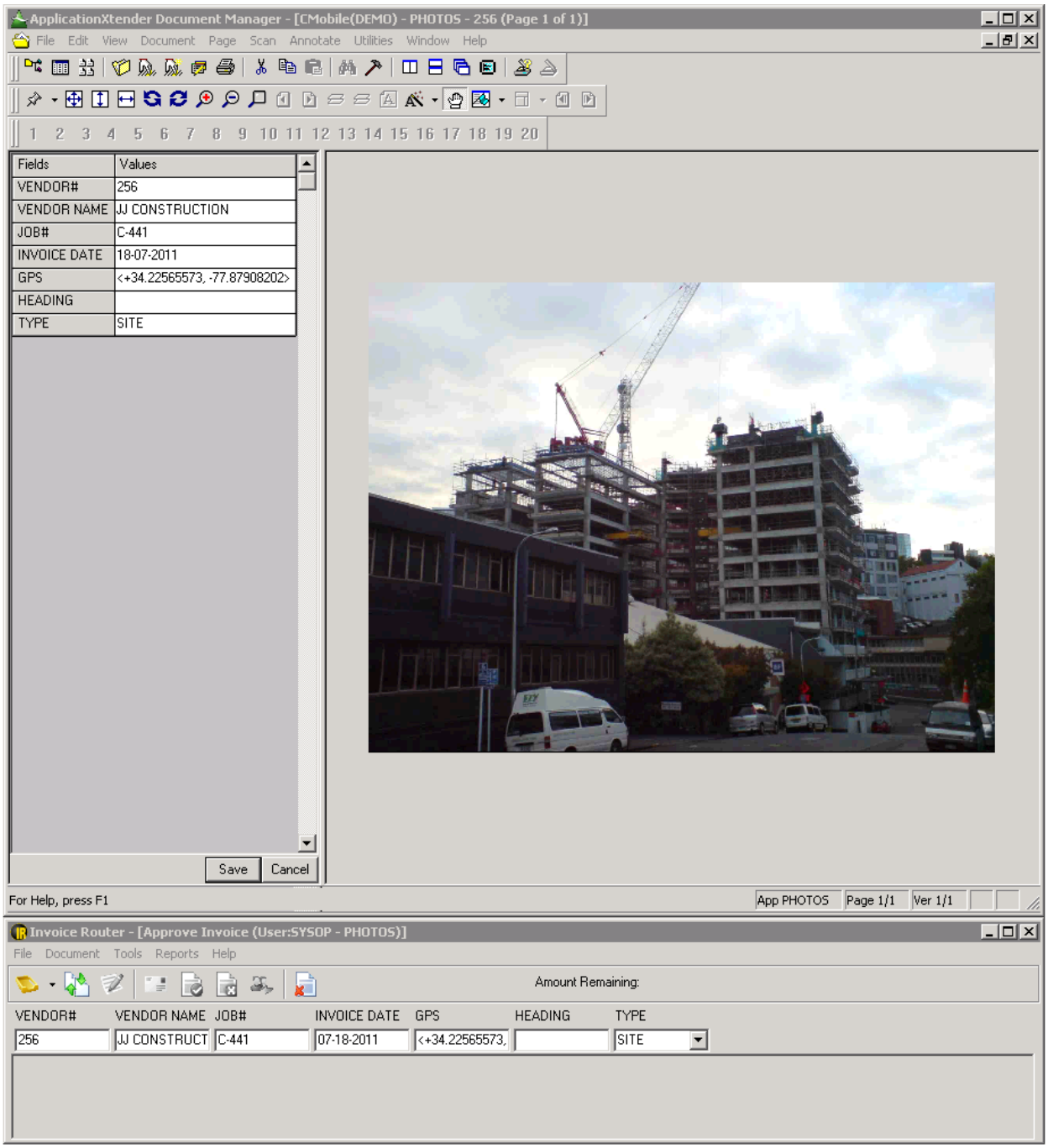

Figure 10: CI Content Manager Desktop Interface

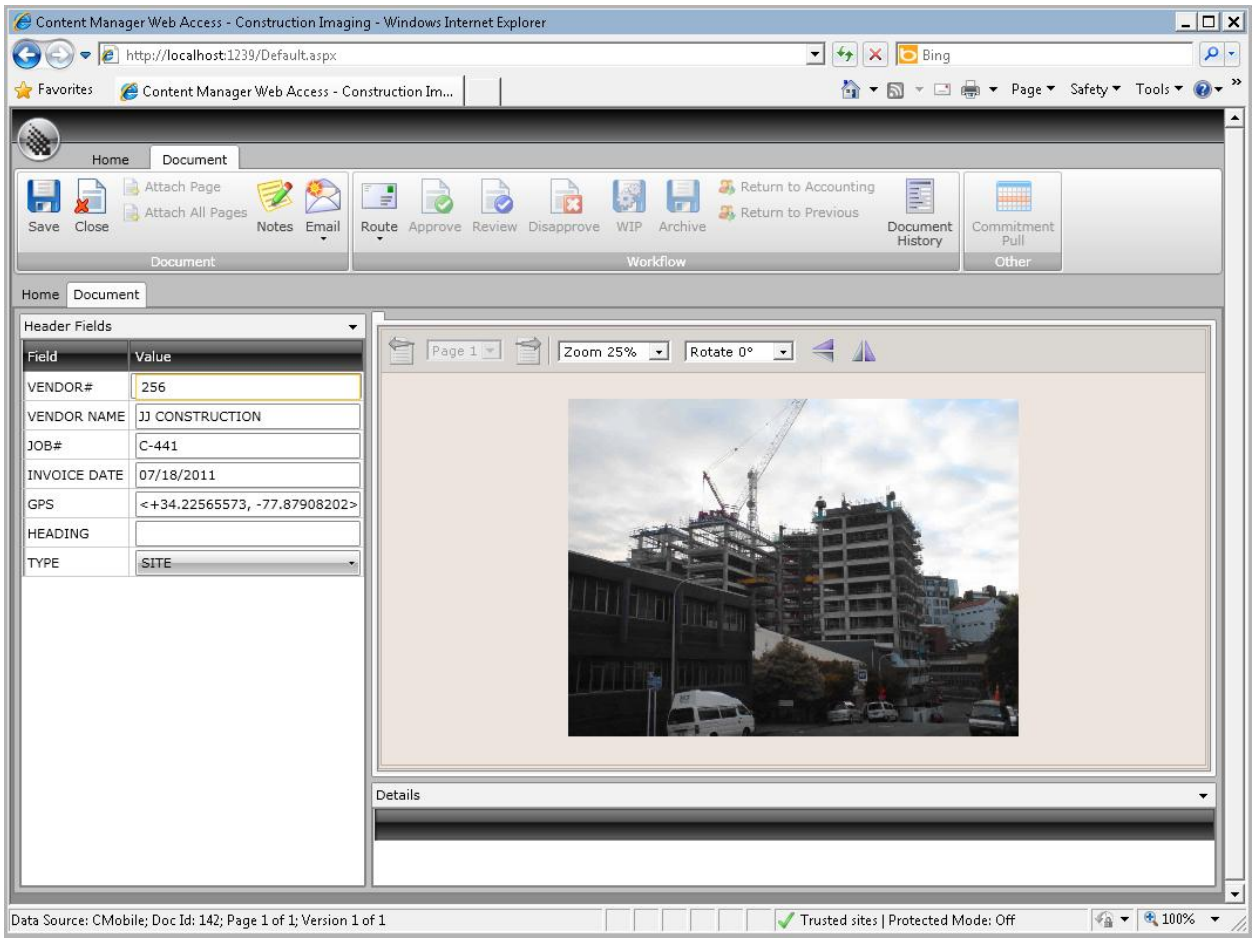

Figure 11: CI Content Manager Web Browser Interface

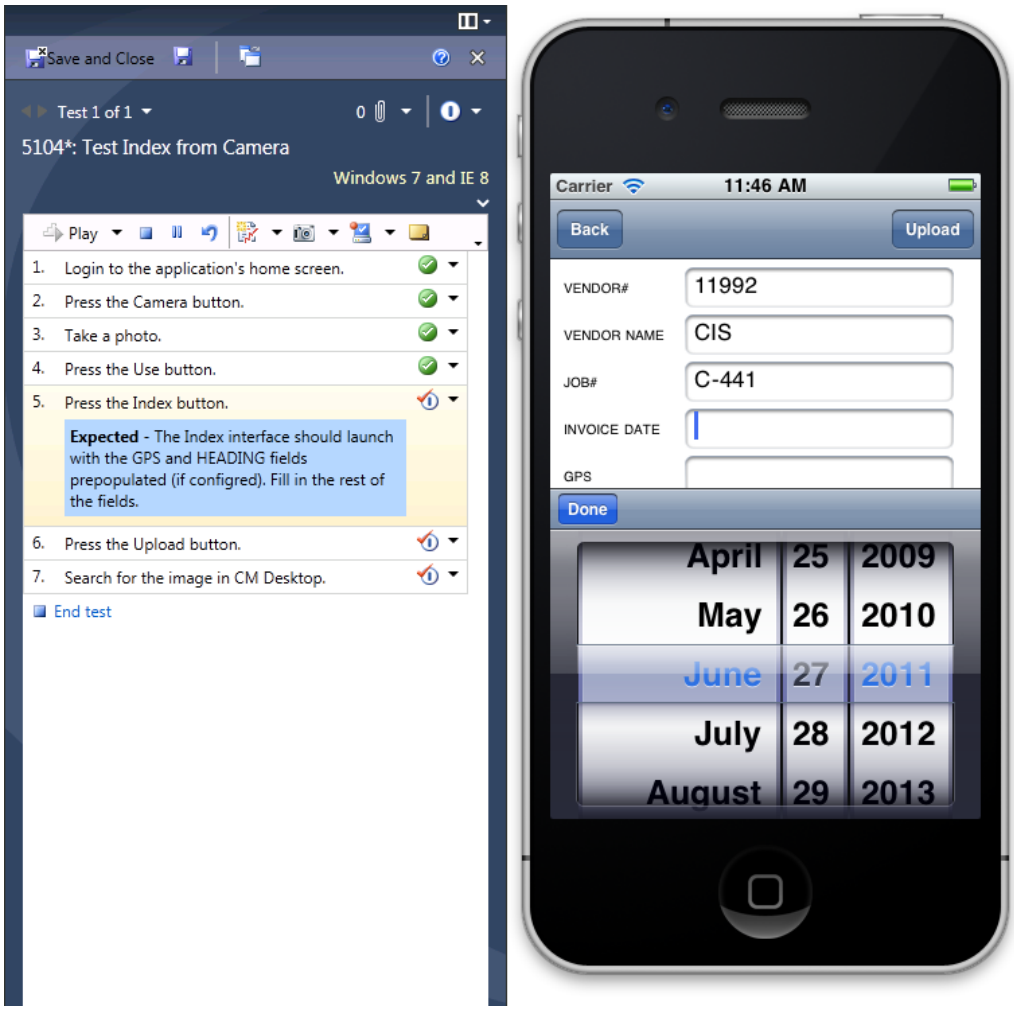

Figure 12: An execution of Test Case 5104