CSM4 – CONNECT
Field Session Final Report – Jun 24 2010

Field Session Team: Ian Littman & Alex Osecky
Client: Tracy Camp, Kerri Stone & Monica Noring

Abstract

CONNECT is a social networking tool for computer science conferences. In the first revision of the project, dubbed CONNECT-1, connections between users could only be made by roaming volunteers with bar code scanners. The CSM4-CONNECT field session team built an API for real-time connection addition by interacting directly with the CONNECT database, and used this API to create Android smartphone and mobile web applications, giving end users a quick way to make "CONNECTions" themselves, provided the user has a data-enabled device on hand. The API allows future developers to easily tie into the system with clients on additional platforms (e.g., iPhone) without having to build potentially dangerous database integration into the clients themselves.

Introduction

The CONNECT project (Creating Open Networks aNd Expanding Connections with Technology) seeks to augment social networking at conferences by replacing business card swapping with a more dynamic, , extensible, technology-based system. With CONNECT, conference attendees are "CONNECTed" electronically and receive a nightly record of all connections made during the day's conference, including "CONNECTee" contact information. In a future version, users of the system will be able to set goals for meeting other attendees, whether simply quantitative (e.g. connecting with ten other attendees) or tailored to a specific target group (students meeting recruiters or faculty meeting technologists).

At the beginning of the 2010 Field term, CONNECT relied solely on a limited number of roaming volunteers with handheld bar code scanners to make connections. This factor bottlenecks connection-making and limits the amount of information that can be associated on-site with a connection, thus diminishing the usefulness and flexibility of the system. An alternate method for making CONNECTions was clearly needed, one that allows users to quickly make connections themselves while "in the field."

Our team researched five different alternate connection techniques and narrowed down the list to three after a meeting with the client: Short Message Service (SMS, text messaging), a mobile web application and smartphone applications. Our group then created fully functional prototypes of web application and smartphone application (on Android) options, including enhancements such as a real-time people search function. Both systems were tied into a small set of HTTP-based APIs to allow additional client

applications to be added easily, and a document was created instructing other project researchers on how to complete the SMS connection creation gateway.

All applications were documented inline (the Android smartphone application used JavaDoc) and online (via the client's wiki site); tutorial videos were also created to instruct users on operation of the smartphone application. All version control for the application itself was achieved through a client-provided Subversion server.

Report Structure

The remainder of this report is set up as follows:

1. An overview of the requirements taken into account when choosing alternate connection systems, plus a description of the client's technical resources, which provided bounds on the languages and systems our team was able to sue for the project (**Requirements**)
2. A high-level discussion of the functionality our team implemented (**Design**)
3. Implementation details for the API, web and smartphone applications, including issues that we ran into while developing each solution (**Implementation**)
4. A week-by-week account of project progress (**Project Progression**)
5. A description of additional tasks that the client wants completed but were beyond the scope of our session, including reasons we decided not to tackle them (**Future Work**)
6. A conclusion to this report (**Conclusion**)
7. A pro-con listing of the five alternatives proposed for bar code scanner replacement, taken from our first report of the session (**Appendix A: Alternative Pro-Con Table**)
8. A glossary of technical terms used in this report (**Appendix B: Glossary**)
9. References consulted during the project (**Appendix C: References**)

Requirements

We investigated five different potential enhancements for CONNECT that negate the need for a volunteer to be available. The potential solutions were as follows: SMS-only (text messaging), "Hybrid" (SMS + 2D bar code), mobile web site, smartphone application and Poken devices.

Each solution was evaluated according to the following criteria:

1. Amount of effort required by the user per connection (smartphone, hybrid and Poken are low, SMS and mobile website are higher)
2. Degree of modification required to the current CONNECT back end (SMS and hybrid are low, other methods are high)
3. Additional feature potential (SMS, hybrid and Poken are low, mobile web and smartphone are high)
4. Amount of effort required to integrate with conference registration systems (Poken, SMS and mobile web are low to medium, hybrid is high, smartphone is low)

5. Development time and effort required (SMS and hybrid are low, Poken and mobile web are medium, smartphone is high)

In some cases development time for functionality on the smartphone application was less than with the mobile web application due to the availability of appropriate APIs for the task at hand.

The major milestones in the project were as follows:

1. Research alternate connection methods and provide the client with a report explaining pros, cons and implementation details of each method, allowing the client to decide which methods to pursue.
2. Create a mobile web application for creating connections between two or more users, including a per-connection comment.
3. Extend (2) to an Android smartphone application.
4. Add Bump (from Bump Technologies, see References) to the smartphone application as a method of sharing user IDs without needing to enter the IDs directly.
5. Implement find-as-you-type search by name in mobile web and Android applications, preferably with images beside each search result, to ease ID selection.

Current Systems

CONNECT-1's code base was written primarily in Python, with some shell scripts included for offline batch processing work. The web front end for the project is on one server running Apache with PHP and Python support, pulling data from a MySQL database on another system. The client was flexible with language choice for our additions to the project, and was willing to install Linux packages on their web server if needed to help our team complete our objectives.

The MySQL database for CONNECT includes three tables: one for e-mailings (`email_stamps`), one to store created connections (`encounters`) and one to store user information (`persons`). A "dummy" database was given on which we could test our enhancements. Subversion was given as our version control mechanism, and the Toilers wiki (a Twiki site) was provided as a location where documentation of our product could be placed.

The state of the CONNECT system before our field session took place can be seen on the next page; arrows represent the major direction information flows through the system.

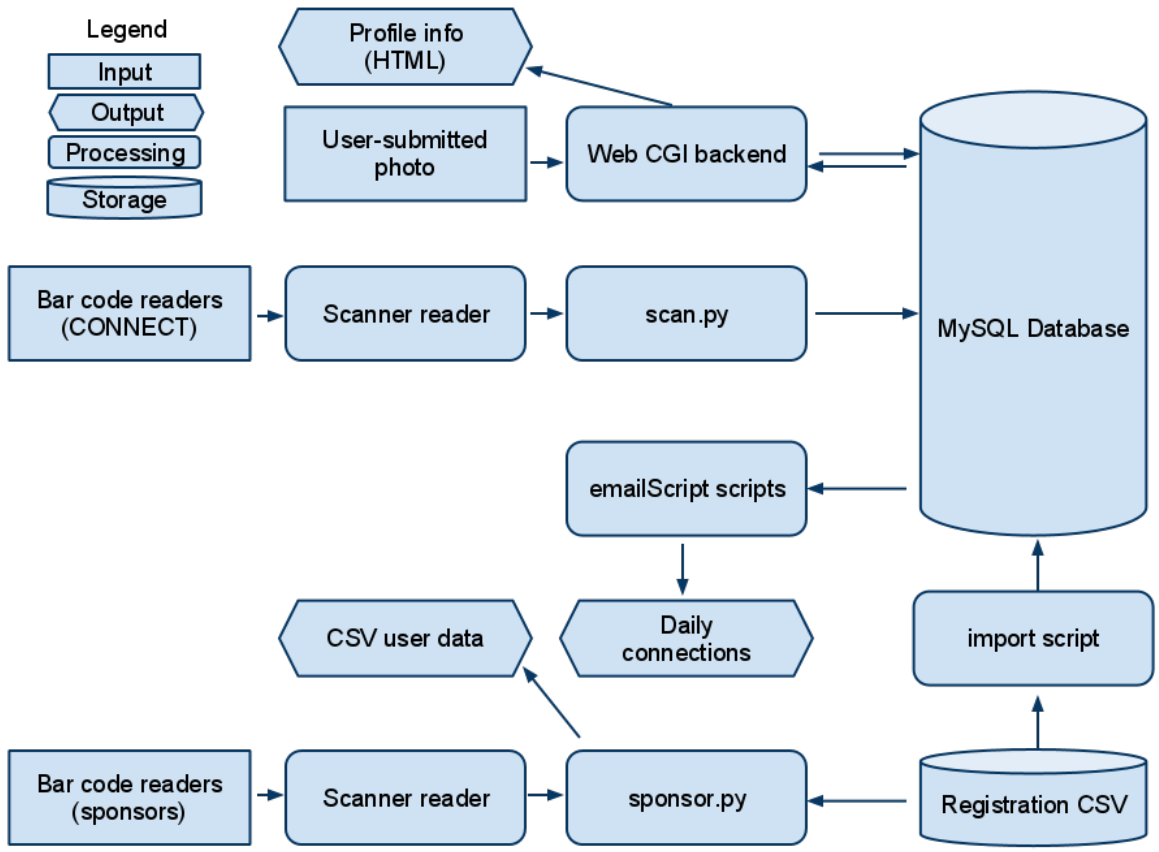

Figure 1 – Pre-field-session architecture diagram

Figure 1 shows that most of the user input for CONNECT-1 was handled by non-real-time Python scripts run by the CONNECT project team, in most cases on a nightly basis. The notable exception to this rule is user profile photo submission, however photo uploads are completely independent from the connection creation apparatus. No APIs were available on which to build connection creation clients, so we were required to interface directly with the database with our own code to fulfill the client's requirements.

Design

Overview

Our product can be divided up into three components: the API, the mobile web client and the Android smartphone application. The API interacts with the back end MySQL database (specifically the `persons` and `encounters` tables), and the web and android clients interact with the API. This format allows for easy extensibility and abstracts direct database interaction away from any user-visible code. We created two APIs, one for creating connections (connectapi.php) and the other for getting a list of names and IDs for a find-as-you-type style search scenario (getNameList.php).

Both the mobile web application and the Android application allow connection creation, including a user-defined comment, and both allow the user to enter IDs by searching for users by name rather than typing an eight-digit number (the current attendee identification scenario) manually. These functions use connectapi.php and getNamelist.php, respectively. The application also saves a user's own ID on device so it does not need to be re-entered on every connection, and optionally includes Bump, which allows two smartphone client users to gat each other's IDs by physically bumping their phones together.

For our mobile web application, we designed two web pages, one with find-as-you-type search functionality and one without. We used JavaScript to resize the CONNECT logo banner to fit device screens, depending on the device's screen resolution (desktop or mobile) and orientation (portrait or landscape). Because low-end mobile browsers do not support JavaScript, we made sure that basic functionality could still be accomplished with JavaScript disabled.

For a visual representation of how the system works, see http://youtu.be/MfH6dmpOBRU for a screencast of the Android client's functionality, and http://youtu.be/eD5LBRyvN3Q for a demonstration of find-as-you-type on the web application.

Figure 2 is a revised architecture diagram detailing the changes that were made by our team:

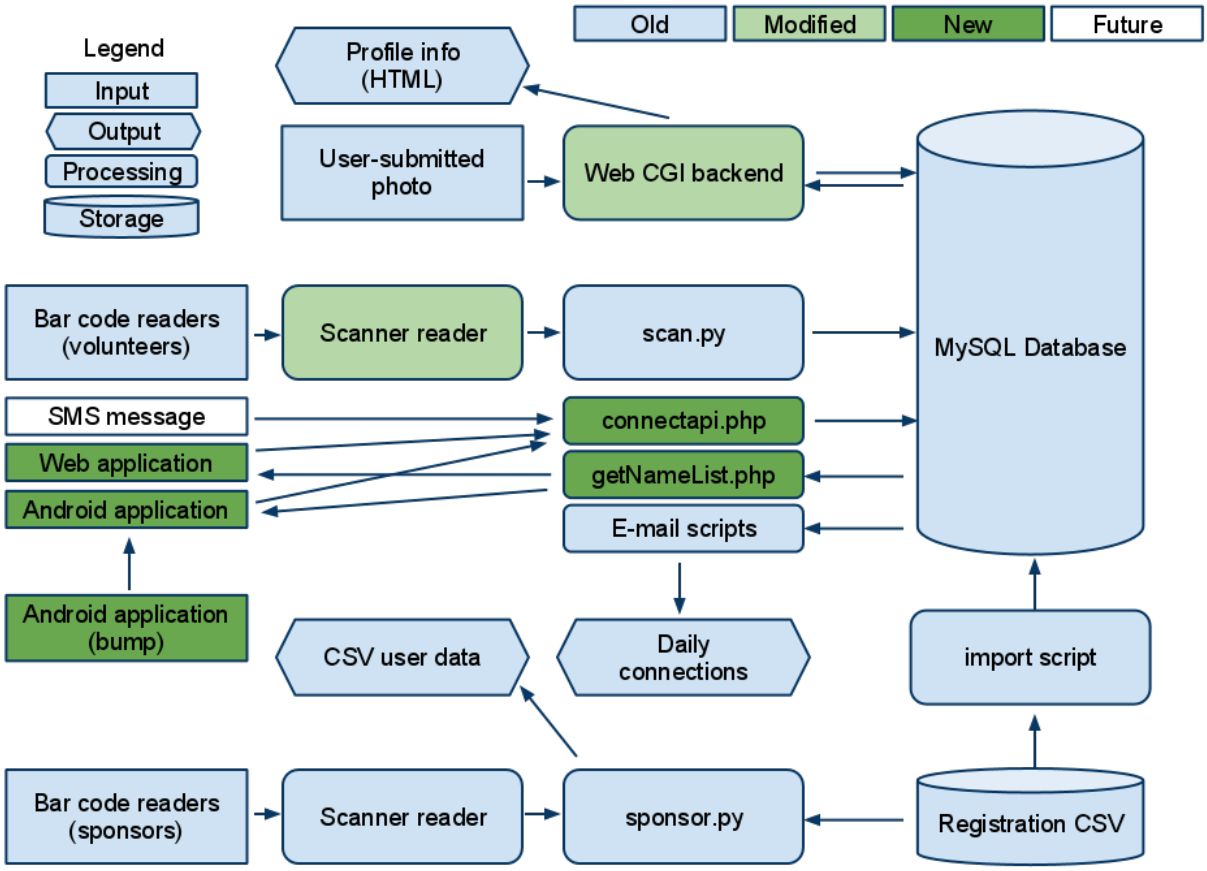

Figure 2 – Post field session architecture/data flow diagram

API Detail

More information on the below API calls can be found on the Toilers wiki article ConnectProjectAPI.

connectapi

Description: Makes connections
Parameters: id_1 (user's conference ID), id_2 (other attendees' IDs, separated by commas or whitespace), comment (adds context to connection)
Returns: Structure containing user's name, names of successfully connected attendees, connection comment and a list of errors that occurred with the connection, if any
Notes: In its current configuration, input may cause the API to return an error that is not in structured format, however this should not cause any issues as long as the client assumes that an unstructured response of any sort is an error response. Errors triggering a non-structured response are those relating to MySQL query failure, for example using a person's full name in the id_1 or id_2 fields. That particular error is due to the `persons` database's ID field being set to an integer type rather than a more flexible string type.

Also, duplicate connections (connections with

getNameList

Description: Basis of find-as-you-type, can be used for standard people search as well
Parameters: q (user's query, see notes for format), p (page number for results, default 1), n (number of results per page, default 10)
Returns: Structure containing one page of matching users' names, IDs and profile photo file names, as well as the base path to be appended to the photo file name to get the photo's location and a count for the number of results total for the query, irrespective of page count
Note: Queries can be part/all of a first/last name, or all of a first/last name, then part of a last/first name

Use Cases

Set User's Own ID

Alternative 1 (Mobile Web): User types in ID upon page load. ID is not stored.

Alternative 2 (Smartphone):

1. User ID entry dialog is displayed via one of the following
    a. User opens Android application with cleared cache, or for the first time
    b. User selects Set User ID from the Menu
2. If case 1b, user can cancel dialog via one of the following
    a. User hits Back button on device
    b. User hits Cancel button on dialog
3. User enters text into ID box, hits OK
4. Text in ID box gets saved to program memory and a persistent data store, which is checked at subsequent program launches to pull user's ID

Search for contacts by name (not available on Bump branch of smartphone app)

1. User engages search interface via one of the following:
    a. User selects First Name/Last Name field in the web application
    b. User selects Search from the Menu in the Android application
    c. User hits Search button on device in the Android application
2. User starts typing a query, search interface makes an API call to getNameList, search results (if any) pop up below the search field
3. User selects a query via one of the following
    a. Using navigation buttons and Enter/Select key
    b. Tapping/clicking correct result on-screen
4. Selected user's ID is appended to the Their IDs box

Display Help (smartphone application only)

1. User selects Help from Menu
    a. Using navigation buttons and Enter/Select key
    b. Tapping/clicking correct result on-screen
2. Dialog is displayed with information on application operation
3. User exits dialog via one of the below
    a. User hits Back button on device
    b. User hits OK button on dialog

Get other user's ID via Bump (Bump revision of smartphone application only)

1. Two smartphone client users select "Get ID via Bump" button on application
2. Bump API displays, initializes, displays "Bump phones to connect"
3. Users bump phones together to register an accelerometer event on both devices, repeat process until ID confirmation dialog appears or cancel window by hitting Back device button.
4. Users accept or cancel confirmation. If both users accept, each Bump participant's stored user ID is added to the other user's Their IDs text box.

Make a CONNECTion

Alternative 1: Smartphone application

1. User enters text into "Their IDs" box, enters text into "Comments" box if desired
2. User hits "Make a CONNECTion" button
3. Client makes API call to connectapi, parses the result or lack thereof and displays a dialog…
    a. Complete success: dialog displays success message, lists full names of users connected from "Their IDs" box
    b. Partial success: dialog displays success message, lists full names of users successfully connected from "Their IDs" box, notes that some errors were thrown, shows a "More Info…" button that, when clicked, gives users a list of all the errors thrown.
    c. Complete failure: dialog displays failure message, shows a "More info…" button that, when clicked, gives users a list of all the errors thrown.
    d. No response/invalid response from server: dialog displays failure message, shows a "More info…" button that, when clicked, shows an error to the user explaining that an invalid response, or no response at all, was received, and asks the user to retry the connection when a more solid data signal is available.
4. User selects OK button in the dialog or hits Back button on device to exit the dialog

Alternative 2: Web application

1. User (maybe) enters text into “My ID”, “Their IDs” and “Comments” boxes
2. User hits CONNECT button
3. Client makes API call to connectapi, parses result or lack thereof, displays a web page as follows:
    a. Complete success: page displays success message, names of everyone connected (including user), connection comment
    b. Partial success: page displays success message, lists names of users successfully connected (including user) and connection comment, as well as a list of errors thrown by the connection attempt
    c. Complete failure/invalid/no response from API: page displays failure message, lists errors thrown by the connection attempt

NOTE: Potential errors mainly consist of invalid/missing user IDs for either “Your ID” or “Their IDs.” Duplicate connections are quietly discarded by the API and are not regarded as error conditions.

Implementation

Languages and Formats

We decided to use PHP as the back-end language for our API due to its simplicity and compatibility with all major web server systems. For front-end web work, we used JavaScript and HTML due to our rather low-end requirements for the system. For both our API and our mobile web application, we built our application from the ground up due to its simplicity; complex frameworks would have obfuscated rather than assisted with the application and possibly made it slower to run.

The API’s input is a simple HTTP GET request for ease of testing in both client and API implementation. API responses are blocks of JSON due to their compactness and readily available Java and PHP tools for parsing JSON objects and arrays. Due to JSON’s compact size, even such complex and relatively data-heavy options as find-as-you-type search result population occur in near-real-time, even over CDMA 3G in our tests. In fact, due to debugging code in the Android emulator, find-as-you-type search was found to be faster “in real life” than in-emulator, with near-instant population for a query. For more information on our testing methodology, see the **Testing** heading later in this section.

For the smartphone client, we developed in Java (Android’s native language is Java) and made significant use of built-in Java and Android APIs for HTTP communication, URL building for API queries, persistence and find-as-you-type search. Bump’s API is implemented as a Java JAR file, plus a number of additional non-code resources (images, animations, sounds).

Development Environment and Problems Encountered

Our IDE of choice was Eclipse, with the PyDev extension installed for working on legacy Python code from CONNECT’s previous iteration. For PHP, sometimes a text editor, like TextWrangler, was used. We decided to run the web application from a DreamHost shared hosting account for easier file modification and accessibility to the outside world. The amount of work required to transition APIs and applications to the actual CONNECT server is minimal.

One snag our team ran into was that the Bump API for Android currently crashes when called if an XML file is in the Android res/xml directory for a program. The problem with this in our case is that Android's search API requires programs using it to have an XML file in that specific folder. The problem took awhile to track down, however as it turned out we were not the first ones to experience this difficulty. See http://groups.google.com/group/bump-api-android/browse_thread/thread/3caa1f58019b73eb for more details on this issue.

Android Details

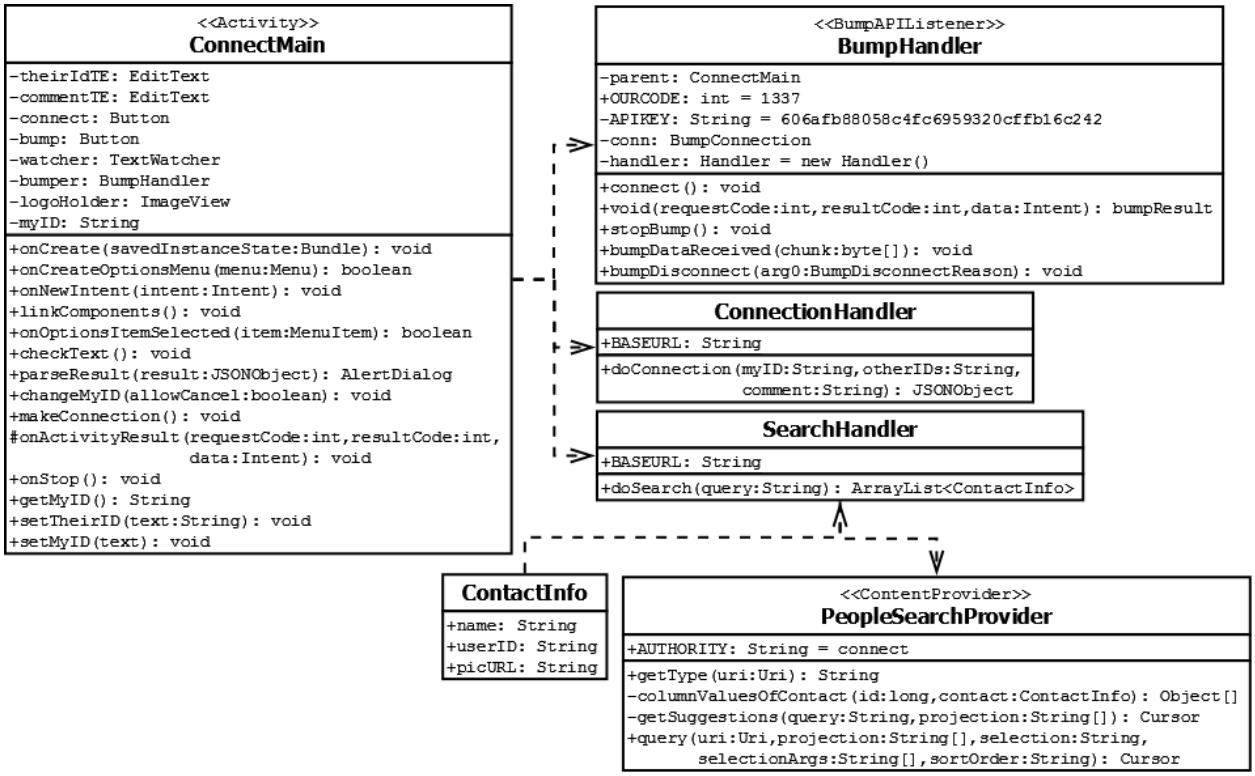

Figure 3 - UML of Android application

Our Android application ended up being divided between five Java classes in all. Our main (ConnectAndroid.java) file contained the majority of program code, however both search and connection API calls were given their own classes: `ConnectionHandler.java` and `SearchHandler.java`, respectively. Android requires a ContentProvider implementation to deliver properly formatted suggestions to its search facility for find-as-you-type; this was provided in `PeopleSearchProvider.java`. Our customized Bump API listener implementation can be found in `BumpHandler.java` in the Bump branch of the Android project code. More information on the application can be found at the Toilers wiki page entitled ConnectAndroidClient. All major methods and classes are documented inline using JavaDoc.

Non-core Enhancements

Some enhancements were made to the CONNECT system that did not directly involve the APIs and/or mobile web/Android clients. These enhancements are noted above with the lighter shade of green and are described below:

Web CGI backend

In order to implement find-as-you-type search with photos inline with reasonable network performance, we needed thumbnail-sized images for each user's profile picture. For flexibility, we wanted both 96x96 and 48x48 pixel thumbnails, in case an application on a high-resolution smartphone (like the new Apple iPhone or some higher-end Android handsets) requested a thumbnail.

The logical place for thumbnail creation was within the CONNECT photo upload script, so we added a few system calls to ImageMagick within the upload Python script to create the necessary thumbnails. The ImageMagick package was originally not installed on the client's web server, however they had no problem adding it upon our request.

Scanner reader

The current scanner based system for making connections will not be retired immediately, so it is in the client's interest to make sure the scanner driver application works as smoothly as possible. Unfortunately, due to low scanner hardware quality, sometimes a scanner's clock second value would get stuck on 255 (or -1 if the second value is a signed 8-bit integer), or record a scan with a second value above 60 otherwise. Either solution would throw an exception in the Python-based driver application.

To correct this issue, we added code in the driver to change a 255-second reading to a zero-second reading before the number was parsed as a date. For other 60-plus values in the seconds variable, we subtracted 60 from the seconds variable and added 1 to the minutes variable in the scan time stamp until the seconds value was in the expected range for Python's date function (between 0 and 59 seconds, inclusive).

Testing

We tested our mobile web application's basic functionality on a variety of desktop and mobile browsers, mostly via emulators but in some cases by using an actual device. The platforms we tested on were Openwave (used in older non-smartphones), Opera Mini, iPhone, Android and Chrome on the desktop. On Android we tested using different screen resolutions to ensure that the web site displayed correctly on all of them. More information about our testing methodology can be found in our initial report.

For the Android application, we used the Android emulator provided with the platform's SDK to assure compatibility with multiple screen resolutions and operating systems. We also tested on hardware with the Motorola Droid and the HTC Evo 4G, which at the time were running Android 2.1update1 as their operating system.

Project Progression

Below is a summary of how the project progressed over the five weeks of work time during field session:

Week 1 and before – Research

1. Read CONNECT proposal to get additional background on the project
2. Researched QR codes, OCR software, Poken, bar code reading libraries for initial report
3. Discussed smartphone/mobile web options with Adam Jack (part of CONNECT team)
4. Wrote initial options report
5. Constructed initial presentation (presented in week 2)

Week 2 – Web UI

1. Wrote proof-of-concept script to convert Google Voice formatted SMSes to bar code reader output format
2. Set up development environment for Android, iPhone, Python
3. Created initial connection creation mobile website, learning PHP/MySQL syntax as needed
4. Reworked mobile website to separate control logic from view logic, creating first API
5. Researched Bump API (for smartphone application)
6. Created design presentation (for week 3)

Week 3 – Android application

1. Wrote design report, modified design presentation
2. Modified API
3. Wrote initial Android application, including Bump functionality
4. Added some project documentation to Toilers wiki

Week 4 – Testing and search addition

1. Activated Verizon Motorola Droid to allow Bump testing (Ian has an Android phone)
2. Tested Bump functionality, fixed bugs
3. Added significant additional documentation to Twiki
4. Added API for people search supporting partial (find as you type) queries and suggestions
5. Started research on iPhone application creation, then was told by client to not go down that route for the moment
6. Attended three client meetings to solidify tasks for weeks 4 and 5
7. Started find-as-you-type mobile web application (JavaScript based)

Week 5 – Search and documentation

1. Continued enhancement of find-as-you-type capability on mobile web application
2. Modified find-as-you-type API to add more search options
3. Added find-as-you-type functionality to Android application
4. Debugged Bump incompatibility with find-as-you-type, branched code (current state is one Bump revision, one search-capable revision)
5. Added documentation, tutorial videos to Twiki

Future Work

The CONNECT project is by no means complete, and the following are immediate areas of concern for moving on with what we were able to create:

Security & Confirmation

Currently, anyone with knowledge of the CONNECT API's location and request/response format can create connections or obtain a list of all conference attendees in the CONNECT database, regardless of whether they have opted in to the CONNECT program. The opt-out issue can be easily solved by a few changes in the getNameList API, however further work on security would require a client and/or user authentication system, which due to time constraints was beyond the scope of the field session project.

Additionally, due to the ease by which a user can now make a connection without the other party's consent, a "pending" state needs to be established for connections created via the API and a web interface must be made available to a user, after login, to confirm or reject pending connections. This was discussed with the client, however such a system's usefulness is predicated on a per-user login system, which has not been implemented yet.

Scalability

Our team did not have the ability to test our applications on a "live" database due to privacy concerns, yet the most likely performance bottleneck for our application is the speed at which results for our complex API-based MySQL queries can be evaluated and returned. Before releasing the API and associated applications to a real-world conference environment, the system should be stress-tested to ensure that performance is still acceptable when the associated database tables have thousands of records in each.

SMS Integration

We were able to outline the steps needed to get an SMS gateway to CONNECT working. The system would use Google Voice to translate between text messages and e-mails. Then Python or shell scripting would be utilized to parse through the resultant e-mails and insert the appropriate connections into the appropriate CONNECT database table. However the client instructed us to focus on other items, letting other students work on that facet of the project, so that task has not yet been completed.

More End User Functionality

The client would like users to be able to view and edit their CONNECT profiles via a web interface and possibly a mobile UI as well. They also want to be able to implement goals and social networking games that a user can manage. Ideally, APIs will be created for each task and applications built atop them, however the vast majority of user-facing functionality requires a robust authentication system first.

Bump + Search Smartphone Integration

Currently the Bump library is incompatible with Android's search API, however this may change as Bump matures. Once compatibility is achieved, the Bump and Search branches of the Android application need to be merged so both ID entry methods are allowed in one application.

Miscellaneous Additions

The Android application does not yet include user photos along with search results. This functionality needs to be implemented and performance tested. Implementation time should be low, as the API is already passing image locations and Android's search suggestion API already includes provision for thumbnail-sized images accompanying each search result.

An iPhone application was proposed to mimic the functionality of the current Android application, however the consensus was that development of the application would take significantly longer for the same functionality as had been implemented in Android. The iPhone application has thus been postponed, to be coded at a later date. According to current surveys the iPhone has a significantly larger application market share than Android, hence the need to develop for the platform in the future, despite its narrower carrier selection in the United States.

More potential expansions of the CONNECT project are available at http://toilers.mines.edu/twiki/pub/Toilers/ConnectProject/CONNECT_tasks.doc (login required).

Conclusion

We believe that we did a large amount of useful work for the client, delivering a working product that suited their needs. Some user interface elements may need to be honed, and additional functionality could certainly be added, however we believe that all of the client's requirements have been met.

Other than significant exposure to PHP, Android's Java implementation and AJAX-style programming, our team learned that Google Docs is a handy tool for collaboration on reports and presentations, and that more complex, API-rich systems such as Android are incredibly helpful at implementing advanced functionality (like find-as-you-type) versus lower-level systems like JavaScript. Our foray into Android was helped significantly by modifying tutorial projects rather than trying to create our application from scratch, though we at times saw some idiosyncrasies of such a new system, including broken libraries and documentation that did not adequately explain some of the API options available in the operating system.

For further information on this project, consult the Toilers wiki under ConnectProject, ConnectProjectAPI, ConnectWebUI and ConnectAndroidClient entries, or contact ilittman@mines.edu directly.

Appendix A: Alternative Pro-Con Table

Pro	Con
SMS System	
Compatibility with almost any cell phone	Significant user overhead per connection (1)
Connection-related comments easy to implement (5)	No advanced services (geolocation, profile viewing, etc.) (3)
Can be implemented in basic form with no required changes to CONNECT back end or registration systems (2, 4, 5)	International text messaging is expensive (if only one gateway number is provided and the service is used in a different country)
Can easily be extended for direct e-mail connections (for users with a data plan and no text messaging bundle) (1)	Domestic text messaging is expensive (up to 20¢ per message) if a user does not have the service included in their wireless plan
Rudimentary offline support (SMSes buffered in phone outbox when in low signal areas)	
Note: User effort can be mitigated by shorter ID codes, adding CONNECT phone number to address book. Shorter codes would require back end coordination.	
Note 2: MMS (Multimedia Messaging Service) could be used to transmit bar code images, to be parsed server-side, however potential flaws make this a difficult option to recommend.	
Hybrid System (SMS + 2D Bar Code)	
SMS method can still be used, serves as basis for system (2, 4, 5)	Current system can't read 2D bar codes, QR/DataMatrix readers can't read 1D codes, so two codes per badge required to make system work (2, 4)
Quick and easy to scan a code (1)	Not much effort saved over pure SMS systems (1)
QR/DataMatrix reader software widely available for most mobile phone platforms (5)	Registration systems may not be able to generate necessary codes natively, and systems may not be able to embed information beyond an ID number into a bar code (4)
	Software for scanning code formats common to registration systems (PDF417, Code 39) for mobile phones is comparatively rare (5)
Mobile Web Site	
Can serve as a testbed for any APIs destined for implementation in smartphone applications etc. (5)	Advanced functions are harder to code then with a smartphone application (lack of APIs, lack of standard platform) (5)

Basic functionality quick to set up relative to a smartphone application (5)	Unusable when a data connection is not available
Cross-platform compatibility (user agent detection can be used to customize experience by phone)	Relies on manual data entry for codes (no better than SMS), cannot currently use phone canera (for bar code scanning or name badge OCR) (1)
Can implement extra features (profile views etc.) (3)	Requires users to log in to make connections; session cookie may or may not last for the entirety of a conference, so login may need to be done more than once (1)
Smartphone system (iPhone and Android)	
Can use multiple methods for quick ID entry (bar code scanning, name badge OCR, Bump, search) with APIs for easy implementation (1, 4)	Application must be downloaded by each user wishing to use it (not a requirement on mobile web, SMS) (1)
Can include commenting, location features, profile viewing, alternate connection techniques (people nearby), goal management, etc. (3)	API on CONNECT back end must be created (though can use same API as mobile web site) (2, 5)
Can be used offline (local data store required)	Requires users to log in to make connections (only needs to be done once per device) (1)
Can use code entry as a backup if bar code scanning/OCR fail to work (1)	Significant development overhead (5)
Easy to adapt to a stationary scanner application as proposed in CONNECT 2 (e.g. an Android-based tablet with a front-facing camera) (5)	Must be developed for two platforms to achieve sufficient penetration (iPhone and Android) (5)
Poken	
Very simple and easy to use end user solution (1)	Each device costs about $20
Already has profile management and social networking features, including APIs that CONNECT can use if required (3)	API information is not publicly available (documentation has been requested) (5)
Poken web site provides an interaction timeline	User controls when data is upload (lag time between connection and back end sync may be significantly longer than with current system)
	Can only store 64 contacts before needing to be “dumped” to the Poken server (may be too low a number for large conferences with a successful networking system)
	Must rely on Poken web API to get information from the devices; user IDs are encrypted on-device and transfer protocol is proprietary RF (2)

Appendix B: Glossary

API – Application Programming Interface
Android – Google's smartphone operating system
AJAX – Asynchronous Java And XML (used for the mobile web application find-as-you-type)
IDE – Integrated Development Environment
JSON – JavaScript Object Notation, see http://www.json.org
Poken – A social network focused hardware connection creation device, see http://www.poken.com
OCR – Optical Character Recognition (for reading names from users' name badges)
QR Code – A 2D bar code widely supported on mobile phones
SDK – Software Development Kit
SMS – Short Message Service (aka text messaging)SMS – Short Message Service (aka text messaging)

Appendix C: References

CONNECT project wiki (private site) - http://toilers.mines.edu/twiki/bin/view/Toilers/ConnectProject
w3schools (information on PHP/SQL) – http://www.w3schools.com
Android development website – http://developer.android.com
Bump Technologies (Bump API used in smartphone application) – http://bu.mp
Professional Ajax, 2nd Edition (Zakas, McPeak, Fawcett)