Video conferencing system for Telerehabilitation
EG2604 UROP Manuscript

Yaadhav Raaj[1] and Arthur Tay[2]

Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119615

ABSTRACT

The main portion of this project was a major FYP headed by Prof. Yen Yi Sheng and Prof. Arthur Tay, and was called "Telerehabilitation." It was on the development of a commercially usable electronic rehabilitation system; that was to aid physiotherapists in measuring body movements accurately in various physiotherapeutic exercises. It also involved the development of a software/user-interface that could be used by both patient and therapist. This software involved both the development of an app and a server backend.

An example would be where a therapist has to measure how far a patient was able to bend is forearm. With sensors attached to the patient's forearm, the system would be able to track how far the arm has been bent, and would remotely send this information back to a server backend where the therapist is able to view. The sensors composed of both AVR and AMD based microcontrollers, and Xbee and WiFi wireless protocols.

PROBLEM

Before joining, a basic hardware implementation had already been set-up. And will be discussed in detail further on. Before joining, there already existed several user interfaces. There was one written for an Android phone and another java-based application running on Windows. However, most of these were crude and displayed raw data, such as angles sent from the sensors serially. There was no proper UI implementation/solution that could easily be used by Therapists and Patients, abstracting the unnecessary data. A simple mobile-based user-interface needed to be developed.

METHOD

Software

Before joining this project, the team was still undecided on which platform to choose for their flagship app. I managed to convince the team to choose iOS as their first platform of choice due to the following reasons:

Developing in iOS guaranteed that the application would work seamlessly across all Apple devices. Developing in Android would be cheaper, but would have resulted in fragmentation if the app were to be deployed across multiple Android devices. HTML5 would have ensure cross-platform compatibility, but did not provided hardware access to the Camera as of iOS5 and could not open web sockets as easily as on a native platform.

---

[1] Student

[2] Senior Lecturer

Furthermore iOS is still seen as a professional grade device, especially in the medical field. It may have been faster to start developing on Android since several members were familiar with Java, but developing on iOS is still considered niche, and forcing the team in the iOS development direction would ensure an iOS option for this system. An Android system could always easily be developed later since iOS developers are difficult to find. App deployment and management is done easily with iOS. Administrators could control which iOS device could have access to the app with various certificates and provisioning profiles. Furthermore, the App could easily be deployed on the AppStore if it becomes commercially viable

HTML5 code can easily be accessed via a browser, and Android APK Packages can easily be decompiled to view the source code. This is because both platforms essentially run on a JIT (Just in time) compilation method, meaning the source code can be accessed. Facetime, Berkeley UDP and TCP sockets and the fact that I could provide assistance in the provisioning profile and setting up of the development environment helped as well.

Hardware

The hardware system composed of a head node, made of an mbed based 32-bit ARM microcontroller, WiFly RN-171 802.11n wireless, and Xbee 802.15.4 wireless modules. The sensors consisted of Arduino Fio chips, with an 8-bit AVR microcontroller, attachable Xbee module and Sparkfun ADXL335 accelerometer module. The diagram on the right represents the communication diagram between hardware. All communication occurred serially.

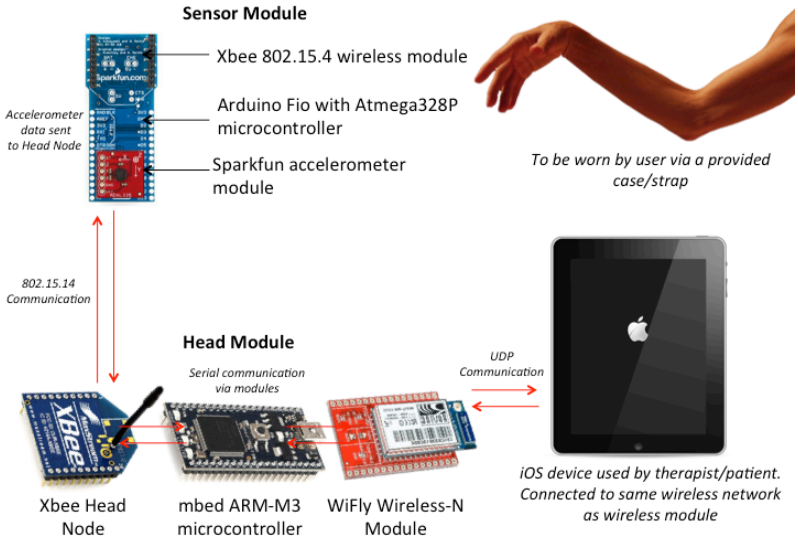

Figure 1

As seen from Figure 1, the system consists of multiple sensor modules, and a single primary head module. This head module would be addressed via the iOS app from the iPad. It is assumed that the iPad and the WiFly chip are on the same network. Although the hardware set-up was more or less built, I completely understood it's internal workings and circuit connection. This ensured that the person who built the circuit set-up, Benjamin Hon[3] would not be the only one familiar with the hardware side.

Use Case Scenario (USS)

Multiple apps were to be developed, one for the Patient, and one for the Therapist. Also, a server backend was to be set up as well. The iOS apps were programmed in Objective-C/C++, the server backend in HTML/PHP and the

[3] Master's Student who built hardware

hardware drivers in UNIX C. I was tasked to be involved in the development of the Patient app and server backend. Detailed below is the user case scenario in deciding what components were required for the app.

Patient needs to begin therapy, and opens the application.	Patient needs to be provided with the iOS device. Internet connection required.
Patient/Helper logs in with credentials	Server back end is required where patient credentials and information are stored.
Patient begins a Facetime conversation with Therapist	App automatically calls the respective Therapist.
Patient hits start to begin exercise and list of exercises are displayed	Exercises of the day need to be loaded.
Video preview of exercise is displayed	Videos of exercises being streamed required
Patient hits begin exercises and performs exercise	App communicates with hardware. Hardware drivers required.
iOS device takes a video of exercise being performed.	Video capture controller required
App shows that exercise is complete	Algorithms must make sense of data and tell user exercises are done. Algorithms required.
Video and data are uploaded	Upload controller required.
A graph is shown to the Patient at the end of all exercises	Graph controller is required.

App and Server-side Development

On the server-side, I helped to set-up the WAMP with PHP capability on a server with a fixed IP Address. I also installed modules such as PHP Pear and coded several scripts to help with communication from app to the server.

On the App side, I helped the team set-up the development environment, complete with all code-signing and iPad deployment. I also wrote several major classes to be used in the main program. One of them was the Login controller, which authenticated with the server and received the patient ID, which could then be used to modify the data related to that patient. It allowed login credentials to be safely saved as well, using the standardUserDefaults object, through a hash table.

CamerView: The app required a camera component to handle the recording of a video stream, while data was flowing in from the sensors. It also required the patient to be able to view himself on the camera. This was not possible using the traditional camera/recording API's provided by Apple Documentation. A custom solution was implemented using reference code provided by Apple, and packaged this into an easy to use class that can easily be dragged and dropped into the main storyboard as seen from Figure 2. With a few function calls, video recording could begin asynchronously and could be stopped and saved. Quality control, image size and camera selection could also be controlled. Most importantly, the camera viewfinder could run as a separate window

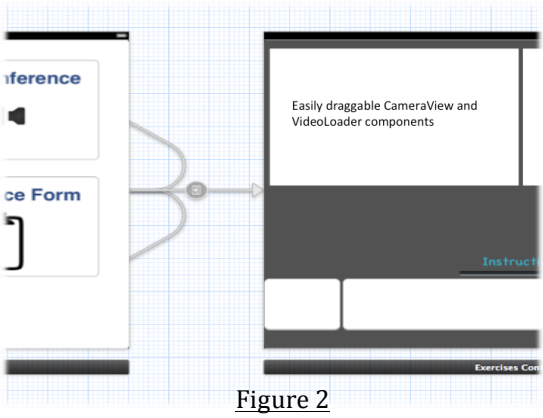

Figure 2

within the app.

QueueFTP: The app also required upload of large video files from the app to the server, where the therapist could then access and view the patient's exercise. Uploading of raw data could be done via simple HTTP Post commands with the included NSURLConnection API. However, in order to upload the video file without corruption, an FTP connection was required to upload the file. The solution's provided by Apple were single threaded methods that would block the main execution of the program when running.

To counter this, I devised a method where the data would be uploaded in packets through a FTP PHP script. This is similar to how files are uploaded in the browser. I then used the ASIHTTP framework to create a class that would allow the input of FTP Parameters such as username/password, and have this object be added to a network queue. This way, the user could easily add large files to the queue and these would be asynchronously uploaded to the server. Also, errors could be handled with ease, depending on the response from the PHP script. Upload progress could also be tracked. Link to site[4]

VideoLoader: The app required a preview video of the exercises to be playing in the background as well. It would start by first checking if the video was stored locally, if not, it would start streaming and downloading the video onto the iOS device. It would then load locally if that particular exercise is selected again. This file can be found in ExerciseController/video.

RESULTS

Although my UROP program had ended before system testing was conducted, my components and the App that I had started building have been implemented into the system. Due to the nature of the iOS development guidelines and UI, the app has been found to be fast, easy to use due to the large touch screen of the iPad. Furthermore, I am fully aware of the internal workings of the client app, server side and hardware drivers written for the app as of completion on Dec 2012.

REFERENCES

"Mbed Handbook." NXP LPC1768. N.p., n.d. Web. 07 Aug. 2012. <*http://mbed.org/handbook/mbed-NXP-LPC1768*>.

Srinivasan, Sriram. *Advanced Perl Programming.* [Sebastopol, CA]: O'Reilly, 1999. Print.

Gay, Warren. *UNIX Programming*. Indianapolis, IN: Sams, 2000. Print.

Mark, Dave, Jack Nutting, and Jeff LaMarche. *Beginning IOS 5 Development: Exploring the IOS SDK*. [Berkeley, Calif.]: Apress, 2011. Print.

[4] https://github.com/soulslicer/QueueFTP