Vibrational Spectroscopy 72 (2014) 37–43

Contents lists available at ScienceDirect

Vibrational Spectroscopy

journal homepage: www.elsevier.com/locate/vibspec

Developing a mobile app for remote access to and data analysis of spectra☆

Matthew J. Evans[a], Graeme Clemens[b], Christopher Casey[a,*], Matthew J. Baker[b,**]

[a] School of Computing, Engineering and Physical Sciences, University of Central Lancashire, Preston PR1 2HE, UK
[b] Centre for Materials Science, Division of Chemistry, School of Forensic and Investigative Sciences, University of Central Lancashire, Preston PR1 2HE, UK

ARTICLE INFO

Article history:
Received 20 December 2013
Received in revised form 13 February 2014
Accepted 15 February 2014
Available online 23 February 2014

Keywords:
Mobile app
Spectroscopy
Spectra
Pre-processing

ABSTRACT

Vibrational spectroscopy is a non-destructive analytical method that can be used to analyse a wide range of materials. A vibrational spectrum contains information on the chemical structure of the sample being analysed, which can be recorded rapidly. With hand held mobile device technology being considered as a relatively mature market, there is an excellent opportunity to combine vibrational spectroscopy with mobile devices for in situ analysis of samples. There are still instances where analytical instruments require being linked to desktop PC's/laptops for instrument control and data manipulation. However, mobile devices are becoming increasingly more powerful thus, enabling data manipulation on devices via cloud based technology. With desktop PC's and laptops often having a larger environment footprint than the instrumental spectrometer itself, this therefore highlights the potential for mobile spectroscopy devices. This paper reports the first development of an app (SpectralAnalyser) to enable the use of mobile devices to access and manipulate spectra and describes the different approaches and implementation issues considered during the development of apps to display spectra on Android and iOS platforms.

1. Introduction

Vibrational spectroscopy, such as Raman and Infrared (IR) spectroscopy are proven analytical methods, which have been used to analyse a wide range of materials. With only simple sample preparation required, a spectrum, which contains information on the chemical composition of the material being analysed, can be recorded rapidly. As well as this, the instrumentation needed to record a vibrational spectrum is simple to operate and relatively cheap i.e. no expensive reagents needed. Because of these advantages, vibrational spectroscopy has been applied to many real world problems in many varied areas, such as biomedical [1,2], defence [3,4], forensics [5], astrochemistry [6], for bio-signatures in the martian environment [7].

For instance, in the biomedical sphere, vibrational spectroscopy has been shown to be a valued technique in the analysis of serum [8], capable of discriminating high grade brain tumours, low-grade brain tumours and normal patients from 1 μl of human serum to sensitivities and specificities of 93.8 and 96.5% respectively [9], classifying ovarian cancer to an accuracy of 96.7% using plasma samples [10]. Raman spectroscopy has recently been shown to be capable of discriminating metastatic brain cancer, high grade brain cancer and normal cancer from tissue samples using a small wavenumber range of 600–800 cm^{-1} to a sensitivity and specificity of 100% and 94.44% for high grade cancer, 96.55% and 100% for metastatic brain and 85.71% and 100% for normal brain tissue spectra [11]. In addition, FTIR has proven to be an excellent analytical technique to quickly determine heavy water concentration in pressurised heavy water reactor [12] and FTIR and Raman are capable of remote sensing of Chemical Warfare Agents (CWAs) [13].

A relatively mature market in hand held instruments and configurable technology provide an excellent opportunity for enabling mobile in situ analysis of samples using vibrational spectroscopy, however in situations as described above an ability to place the spectrometer in a different location to the feedback system would be of great advantage. For instance, a miniature spectrometer with cloud based data feedback could be placed in a hazardous location to monitor air quality thus, providing real time results back to scientists outside the hazardous area, or a spectrometer could be placed in a clinical theatre without the need for an operator inside therefore, minimising the risk of infection. Currently this is not possible.

Developments in the mobile/miniature spectrometer technology and in situ use of spectroscopy require appropriate

☆ Selected paper presented at 7th International Conference on Advanced Vibrational Spectroscopy, Kobe, Japan, August 25–30, 2013.

* Corresponding author. Tel.: +44 01772 893278.
** Corresponding author. Tel.: +44 01772 89 3209.
E-mail addresses: ccasey@uclan.ac.uk (C. Casey), mjbaker@uclan.ac.uk (M.J. Baker).

http://dx.doi.org/10.1016/j.vibspec.2014.02.008
0924-2031/

coinciding developments in software and applications. Some mobile/miniature spectrometers require the spectrometer to be linked to a desktop PC/laptop for instrument control and data acquisition. However, often the desktop PC/laptop has a greater environmental footprint than the spectrometer instrument.

A mobile applet (app) capable of accessing/manipulating data would prove extremely useful and the applications wide and varied. In this study we describe the development of apps for the interrogation and manipulation of vibrational spectra using Android and iOS platforms with cloud-based technology and investigate the different development approaches required and the implementation issues arising with each platform.

2. Methods and tools

2.1. Development approach

Small prototype apps were developed to evaluate the graphics and user interface libraries. Once the libraries had been selected, an iterative incremental approach was used. Requirements for each iteration, a fixed period or "timebox" of two weeks, were defined and prioritised before each iteration as MoSCoW lists based on client feedback. MoSCoW is a "prioritisation technique used in business analysis and software development to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement" [14]. The client classifies the tasks for an iteration as "Must have", "Should have" "Could have", "Would like, but won't be implemented". The developer indicates the expected time each task should take. To be used effectively, timeboxing requires at most 60% of the requirements to be of type "Must have", so the client and developer can be confident that they will be achieved within the timebox.

The iterative lifecycle ensured testing was continuous. A benefit of the Android platform is the heterogeneity of its target devices. However, this still requires rigorous device testing to ensure a consistent user experience. During development, the application was tested primarily on a HTC One X handset, and also on a Nexus 7 tablet and a Kindle Fire tablet. The Android Device Emulator was used during development. However, user interface testing on the emulator is limited as touch screen gestures cannot be truly simulated using a mouse. The iOS app was tested on an iPhone and an iPad as well as the iOS emulator.

2.2. A cross-platform application using Qt

Qt is a cross-platform development framework based on C++ [15]. An initial prototype was written using Necessitas 4.8.2, a version of Qt which aims to support Android implementations [16] and the QCustomPlot graphics library [17]. However, although the implementation ran on an Android-based Nexus 7 tablet, it was not possible to implement the gesture-based interaction expected of a mobile application and development switched to O.S.-specific environments.

2.3. Android application

2.3.1. Development environment and graphics library

The development used the Eclipse IDE, which is the most common environment for the Android Software Development Kit (SDK) [18] and provides facilities for writing, testing and debugging Android applications, this includes integration with the Android emulator and Android devices. Android applications use the Java programming language, compiled for the Dalvik virtual machine.

Several graphics libraries, GraphView [19], AFreeChart [20], and AChartEngine [21] were evaluated through prototypes. The demonstration using GraphView was sluggish on a Nexus 7 tablet, but both AFreeChart and AChartEngine provided appropriate facilities and performed adequately on the Nexus 7, a Kindle Fire, and an Android phone.

2.3.2. User interface

The Android application uses an Action bar across the top of the screen. Since commonly used actions such as "Load" and "Subtract" are always visible, the user can quickly control the application. However, because of the Action bar, the application is only supported on devices that have Android 3.0 and above. Less common actions such as logging into Dropbox or peak selection are accessed through the "action overflow" button of the Action bar, so not to overwhelm the user.

Touch, pinch and pan gestures allow the user to select and manipulate spectra. However, when labelling peaks, the user must use the zoom toolbar to vary the number of annotations displayed.

2.3.3. Program structure

The Android application framework is based around "Activities", "a single focused thing that the user can do" [22]. The code within an Activity is initially executed through callback methods activated at appropriate stages of the Activity Lifecycle, e.g. when the activity is created, started, paused, resumed or destroyed.

The application has a single activity, which creates the action bar buttons and the view of the graph and attaches appropriate event handlers to respond to input events.

Interaction with Dropbox (see Section 2.5.1) uses an `AsyncTask` object, allowing the user interface thread to initiate and display the results of a background operation, while maintaining responsiveness to the user while the background task is running. The developer provides a `doInBackGround` method that is executed by a background thread and a `doPostExecute` method that is executed on the user interface thread after the background task has completed.

2.4. iOS application

2.4.1. Development environment and graphics library

The program was written in Objective C using the XCode IDE, which provides facilities for writing, testing and debugging iOS applications, including integration with the iOS emulator and IPhone/IPad devices. The key algorithms for processing spectra were translated from Java. The Core-plot graphics library [23] provided the necessary facilities and appropriate performance.

2.4.2. User interface

An Action bar approach was emulated using icons, most of which have two actions depending on whether they are pressed or held down, this saves screen space and simplifies the interface. Whereas menus in the Android application appear in a Dialog box, the iOS application has a separate view, implemented by ECSlidingViewController [24] which slides in from the left. This view is populated either with the available Cloud and local files or with the spectra that can be selected (e.g. to annotate peaks).

The iOS application also allows pinch and pan gestures. However, it does not require an extra toolbar to increase the number of annotations. Instead, it automatically decides which peaks to label depending on the zoom level.

2.4.3. Program structure

Asynchronous interaction with Dropbox (see Section 2.5.1) that allows the user interface to remain responsive is provided automatically by the DropBox iOS API. Methods such as `[DBRestClient loadFile:intoPath:]` are asynchronous and return immediately rather than when the requested action is completed. The developer implements event handers for