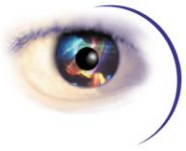


Building Medical Devices or Laboratory Instruments with Remote User Interfaces

by Andrew Dallas, CTO, Full Spectrum Software

Overview

For many years, our clients have been interested in developing instruments that can be monitored or controlled remotely. There are several approaches to implementing this type of service and each of these has its merits and limitations. With the rapid adoption of mobile platforms including the iPad and Android, the number of architectures available has grown. This article is intended to provide the reader with a view into a few of the options in supporting today's technologies.

This paper is not intended to address issues of security or the complexity of validation for controlled medical devices. This paper will not detail how to implement push notifications (such as alarms and alerts) from the device but this capability exists through web sockets, long polling and other technologies.

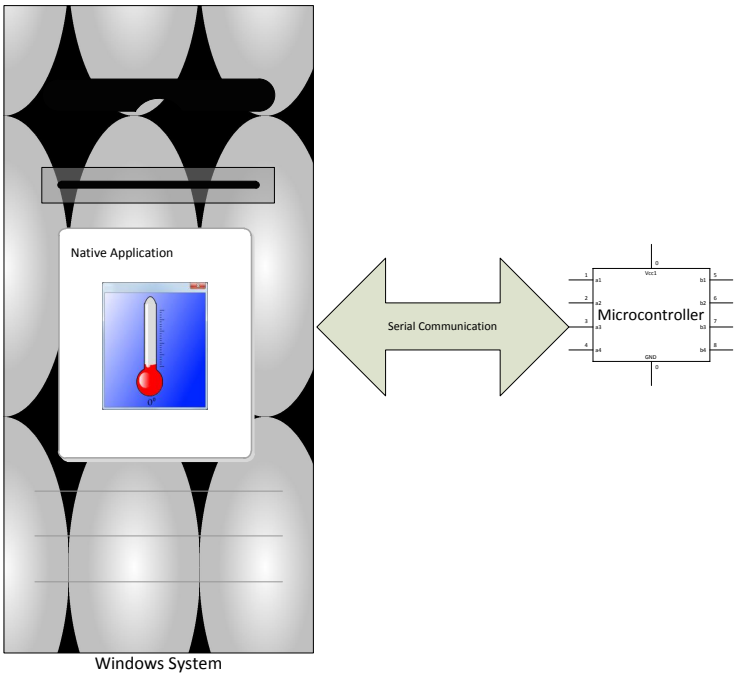
Older Approaches

Historically, many medical and life science instruments were provided remote access purely for maintenance purposes. These remote access interfaces were used to identify faults, monitor consumables and verify that quality control and calibration procedures were being performed. With the capabilities of today's modern operating systems, systems are now often designed to be controlled through remote interfaces as a primary function. This migration has provided users with greatly enhanced capability and it is easy to see that as technologies such as HTML5 are more widely adopted that these capabilities will continue to grow through innovation.

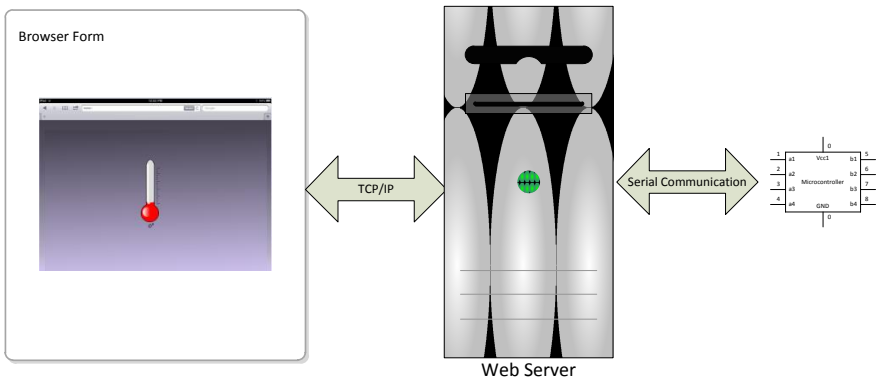
Remote desktop access was an acceptable method of controlling an instrument in the past. Users now expect a more seamless integration between their desktop computer, tablet or other device and the instrument that they wish to control. This usually means that there is either a web interface (through a browser) or a native application (an application compiled for a specific operating system) that connects to the instrument via a network. Both methods are effective but each requires different, if overlapping, skills to implement. Both use web services, a platform independent means by which two programs communicate over the local area network or internet. For the sake of clarity, I will use a simple thermometer application as an example.

The New Architectures

Let me begin by describing the core of the architecture, the host computer to microcontroller communications. Most complex devices these days have at their core a microcontroller. This small processor communicates with sensors and other hardware devices to perform the base functionality of the device. Microcontrollers may be very "smart" and control nearly all the device's function or they can be fairly "dumb" and rely on a host application to control its activities. Regardless of this ability, the microcontroller will communicate with a host application by serial (RS232) ethernet or other physical or wireless medium. The host application provides the user interface to the operator.

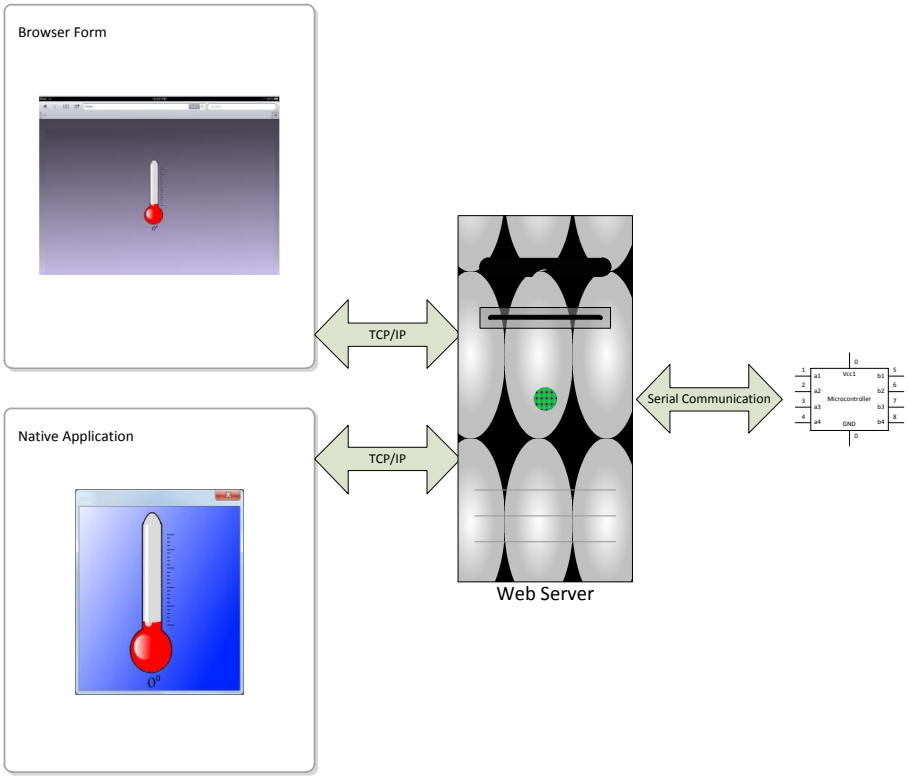


Similarly, when a web server is introduced into the architecture, it takes the place of the host application and it can communicate directly with the microcontroller via the same RS232, ethernet or other means. The web server, however, provides either a user interface or web services to processes running on the same or more importantly, on other, remote computers.



Design Divergence between Web and Native Interfaces

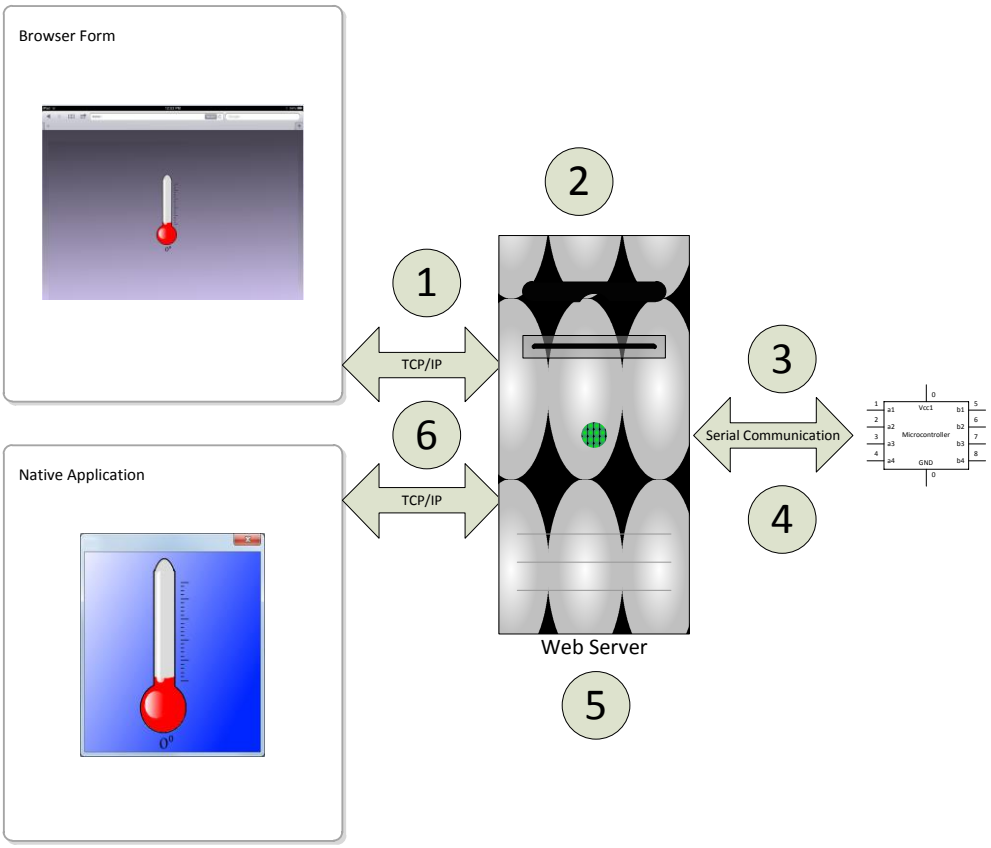
Here's where things start to diverge for native applications and web interfaces. A web interface is implemented such that the web server also hosts the web pages. A browser requests these pages from the web server and displays them to the user. A native application, that can run remotely, will display its own user interface screens and query the web server for the content. This is done through a technology called web services.



For dynamic web pages, pages that update without the user interacting with the page, a page will make a request to the server to update its information. When the server

receives this request, in our case to get the latest temperature reading, it communicates to the device to retrieve the information and then send this on as a response to the request

For a native application, the app makes the same request to the web server for the temperature and it updates its user interface to show the result. Both of these can use web services to support the request.



The Call Chain

1. The browser or native application or browser requests data from the server via web sockets.
2. The server receives the request and optionally refuses due to a fault or busy state.
3. The server sends a request via a serial communications protocol.
4. The microcontroller responds with its current temperature reading.
5. The server performs any checksum, calibration or scaling operation of the value.
6. The server responds to the native application or browser with the new temperature reading via web sockets.

Deciding on an Architecture

When one builds a native application for the iPad or iPhone, the available technologies are somewhat limited. Generally speaking, developers use objective C and Apple's tools to develop the app. The app then needs to be deployed either through the app store or following the enterprise deployment process. OSX doesn't have the restrictions of the deployment model but does use objective C.

When one builds for Android, the language of choice is Java. Deployment is simpler but there are many versions of Android to consider and many devices today are shipping with fairly old versions of the Android OS.

A native MS Windows application will usually be built in C#.NET using winforms or WPF technologies. This is conveniently similar to the back-end programming technology used by Microsoft Internet Information Server.

A native Linux deployment or cross-platform deployment will frequently use C++ and QT or Java. Installation on linux platforms can be more complex than other platforms but C++ engineers and Java engineers are widely available.

When one builds for a web interface, the deployment is directly on the server and is relatively simple. Updates for a system require only a redeployment on that system rather than to all devices that might be clients. The technology for implementation may be limited to HTML and C# or may include flash, HTML5, java or a host of other technologies.

How to choose a technology

Regardless of the technology, remote capabilities should use web services for communication. Proprietary TCP/IP protocols are problematic in the IT infrastructure of some hospitals and laboratories.

When selecting a technology, consideration should be made as to whether many users should have access to the system simultaneously or not. If simultaneous access is permitted, should all instances be synchronized so they see the same user interface as each other. Timing of events should be considered in the technology selection. That is, are there time critical notifications that must be displayed to the user or is it acceptable for the user to see the notifications a few seconds later? If hazard analysis indicates that there are safety concerns, are the concerns mitigated?

From a business perspective, one should consider the technical skills needed to maintain the system. Will it be possible to find an individual or individuals with all the skills necessary to maintain the system or will a large, diverse and possible expensive staff be required?

Summary

My team and I have been building these types of systems for many years. Technology has naturally allowed us to expand the capabilities of these systems and provide exceptional usability along with remote access. Do not be deterred by the false assumption that remotely controlled systems are slow and cannot handle the rigors of high-throughput or algorithmically intense processing. These systems are being successfully developed and deployed today and will help you to provide competitive advantages when implemented following good design practices.

About Full Spectrum Software

Full Spectrum Software is an ISO 13485 certified, 15 year old consulting firm specializing in the development of embedded and applications software for the medical, life sciences and scientific industries. The company has delivered over 100 commercial products to market.

About the Author

Andrew Dallas, the firm's CTO, is widely considered one of the leading authorities on best practices in FDA and ISO 13485 controlled software projects Andrew serves on the Editorial Advisory Board of Medical Device and Diagnostic Industry magazine and he has published extensively in major trade and peer reviewed technical publications.

Contact Cindy Larkin, ClientServices@FullSpectrumSoftware.com
(508) 620-6400 • 225 Turnpike Road • Southborough, MA 01772 •
www.FullSpectrumSoftware.com