

Connecting Smart Device's on the Internet, Technical Options

Terry H. Ess
T. H. E. Solution LLC

INTRODUCTION

The quote that by 200x there will be more smart devices on the internet than people has become widely accepted. You can already see it if you look at the most integrated manufacturing facilities. Everything from the factory floor up is being connected using “web technologies”. If connecting a smart device can create substantial customer value and you do not do it, your competitors will. So the questions in many cases is not should I do it but how should I do it?

CHARACTERISTICS

Answering the how question is highly dependent on a myriad of factors. Some of the more important are:

1. What is the value to your customer of the functionality enabled by connecting your smart device (an in-depth discussion of this can be found at <http://www.silver-bullet-technology.com>)? This will limit what you can do, by limiting the cost etc., and should be used as a key gauge to what functionality should even be considered.
2. What kind of data transfer volume must be sustained. In the IT world, web applications and services are often scaled to handle tens of thousands of transactions per minute? In general the volume of a smart device will be small, on the order of (10) messages a minute, but if this not the case you need to know.
3. What communication characteristics are needed? For instance, do you need near real time synchronized communication or just periodic asynchronous reports? Should the information flow be initiated by the device or the external world?
4. What is being done in your industry? For instance if your device is a factory floor machine you should consider OPC, especially the new XML data access specification.
5. What level of security is required?
6. Who is going to connect? There is a big difference between connecting a device with proprietary remote applications at the manufacturers facilities versus connecting with any customer's supply chain applications.

Understanding the characteristics of your device given the chosen function(s) can be complex. A first order analysis of a potentially high value function for many devices, remote performance optimization, is summarized in Figure 1:

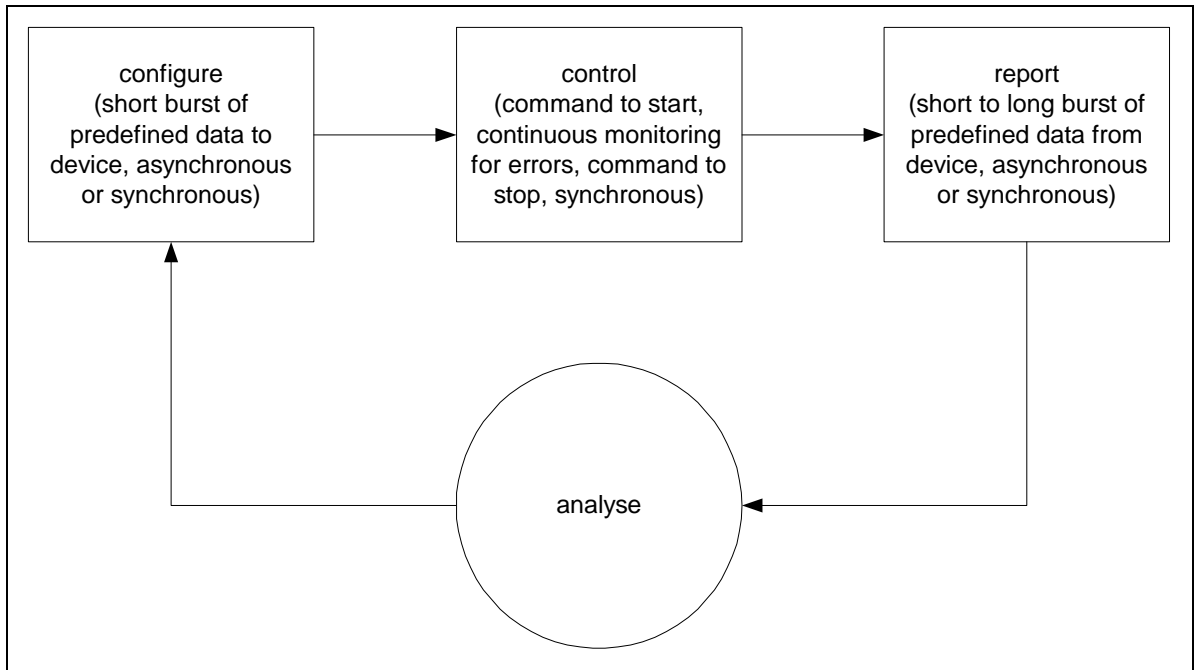


Figure 1, Remote Performance Optimization Flow

As you can see it requires a number of different steps with the remote application connected to the device, each with different characteristics. We can significantly change this application's potential value to customers by making both the configure and report steps synchronous and ending up with the capability to perform near real-time performance optimization.

OPTIONS

It is a given that we are moving device specific data using a reliable but unsecured transport, TCP/IP. But all that TCP/IP provides is the transport of "stuff", any stuff. What we need above this is a means for our software to access TCP/IP and the structuring of how and what is being transported. We can break this into three areas of investigation, TCP/IP access, high level transport protocols and data format protocols.

There are two primary access methods. The most common is the socket interface which provides a simple means to access a software TCP/IP stack. Almost (if not all) platforms of any sophistication provide a socket interface. It can even be found on eight bit micro-controllers. However, this is not the only access method. The TCP/IP stack and the underlying physical connection issues can be off loaded. This is possible using the fairly recent "internet chips" or by front ending your device with another internet capable device. The only typical hard resource requirement on the current device is an RS-232 serial interface.

The primary transport protocol choices are:

1. Proprietary
2. File transfer protocol (FTP) or its "simple" version (TFTP)

3. The world wide web protocol (HTTP)
4. Email protocols (SMTP and POP3)
5. Network management protocol (SNTP)

The principal data format choices are:

1. Proprietary
2. De facto and industry standards (e.g. comma delimited files, ...)
3. Extensible markup language (XML)

Finally there is the combination of high level transport and data format protocols to define the most versatile standard, the web service. Web services are pretty new and the idea of standards is still very much a moving target (though zeroing in quickly). For our purposes web services can be broken into three levels of standardization:

1. Comply with the premise of web service (HTTP 1.0 and XML 1.0).
2. Comply with the WS-I Basic Profile (XML 1.0, XML Schema, HTTP 1.1, SOAP 1.0, WSDL 1.0).
3. Comply with an industry specific definition of a web service (e.g. OPC XML data access 1.0).

COMPARISON

If we wanted to examine every possible solution combination that would become a very large task. Luckily we can take a serious look at only a sub-set of the possible solutions and get a pretty good idea of what is practically available. Specifically the options summarized in the following table were examined in detail including the design, implementation and test of reference code sets:

Description	Application	Top Level Transport	Data Format	Operation
Proprietary content updater	Content updating	Proprietary	Proprietary	Synchronous
Email client	Periodic pre-defined reporting	SMTP, POP3	XML XML Scheme	Asynchronous
Web service client	Content updating	HTTP 1.1	XML XML Scheme	Synchronous
Web service server (light)	Information retrieval	HTTP 1.0	XML	Synchronous,
Web service server (WS-I Basic Profile)	General purpose for control, configure and reporting	HTTP 1.1	XML XML Scheme SOAP WSDL	Synchronous

The proprietary content updater is described in detail in <http://www.the-solution-llc.com/strategy.zip>. The specific reference implementation provides a good example of offloading access to an internet capable device (in this case an off-the-shelf Windows CE device). Since the application level protocol is simple and binary, data exchange is the most efficient by far. The reference implementation on the CE device is fairly small requiring less than 900 lines of C. If the device itself had the capability to connect with the Internet, the implementation on the device would be the same size etc. Since the application level protocol is proprietary, the server side component had to be implemented at the socket interface level. A minimal server required another 900 lines of C++. This does not include the support applications that would be needed to keep track of changes etc.

The Email client provides the means for a “center” to specify the pre-defined report which a device will mail periodically. Both the requests and the reports are sent in the body of the E-mail in XML format. Both also include security measures to circumvent attacks. The device reference implementation using a socket interface required less than 600 lines of C++ while the implementation using offloading to an internet chip, ConnectOne’s iChip, required less than 800 lines [Note: The overall size of the iChip implementation would be substantially smaller since the software TCP/IP stack etc. is not needed]. The performance of both implementations were roughly the same. Two desktop level (not enterprise) applications for a center were also implemented. These provided the means to easily compose and send report requests and to retrieve, parse and store device reports in a database. Both were implemented in Visual Basic as Email client programs (the Email server was provided by the ISP). Together they required less than 1000 lines of VB.

The web service client is described in detail in <http://www.the-solution-llc.com/updateservice.pdf>. It provides the means for a device to intelligently update its content (including software). The updating procedure is significantly more sophisticated than that used in the proprietary content updater though it achieves the same basic objective. The procedure uses the exchange of defined (using XML Scheme) XML documents and then a series of HTTP gets to update the device. The reference implementation includes some security measures but is not completely adequate in this area. The device reference implementation using a socket interface and a minimal sequential XML parser (Note: the actual implementation used the Xerces open source XML parser which is large) should require less than 800 lines of C. Two server side reference implementations were undertaken, one using J2EE and one .NET. Each took less than 500 lines of code (Java and C# respectively) to implement the update web service.

The light web service server is described in detail in <http://www.the-solution-llc.com/webservice.zip>. It provides the means for client programs (including web pages, spreadsheets etc.) to easily retrieve information from a device using HTTP gets with “freehand” XML documents as the response. The device side server reference implementation took less than 500 lines of C++. The client side scripts (VB script and VBA) required less than 100 lines.

The basic profile compliant web service server is described in detail in <http://www.the-solution-llc.com/embeddablewebservice.pdf>. It provides a general purpose “command and control” interface through a WSDL defined interface. The device side server reference implementation using a socket interface tool less than 1000 lines of C++. The internet chip version, ConnectOne’s iChip, required less than 1400 lines [Note: The overall size of the iChip implementation would be substantially smaller since the software TCP/IP stack etc. is not needed]. Unlike the Email case, the performance of the internet chip version was substantially less than that of the socket interface version. In general the internet chip should be restricted to five or less simultaneous connections. Where this option really shines is on the client side. Two client applications were built to test the server, one using C# and .NET the other Java and Sun’s Java Web Services Developer Pack 1.3. In both cases access to the web service was automatically generated from the WSDL and the client program simply called stubs.

None of the solutions examined used a De factor or industry standard data format. Why? Because with the possible exception of government EDI standards, they are all expected to disappear under the integration power of web services. This is not to suggest that the use of XML standards comes without a price. For example, the amount of data (in bytes) exchanged to “read” a device’s short parameter would be approximately (10) bytes with the proprietary binary encoding option, (130) bytes in “freeform” XML and (900) bytes with a basic profile compliant web service. The only reason this inefficiency is tolerated (or even ignored) is because bandwidth and processing capabilities have grown so large and web service support so widespread.

Until the advent of the internet chip, the use of the offloading option was too expensive for most applications (though it did provide a good development path to get from legacy to next generation devices). With the internet chip that is no longer true. If your device has an available serial interface and the connectivity enabled applications have value (e.g. performance optimization) then you can probably fit and afford one.

RECOMMENDATIONS

Given the high variance in characteristics it is unrealistic to think that there is one single best solution. All of the solutions investigated can be implemented even on device’s with fairly modest platforms. However, some key points can be made:

1. Unless you absolutely can not afford the inefficiency, XML is the way to go for data format.
2. A basic profile compliant web service server is extremely attractive if you will need to tie in to a customer’s or you own IT organization and have a persistent connection. This option requires that you know what you are doing (both technically and otherwise). If “connecting” is new for your industry approach this carefully. If your device works on the factory floor you should seriously consider an OPC XML data access compliant web service.
3. If synchronous operation is not required for the connection enabled applications you will support, the Email client is the safest and easiest path. It is especially

attractive in circumstances where there is no persistent connection and you are enhancing a legacy device using an dial out version of an internet chip set. It can be implemented, even on a large scale, without any need for extensive enterprise infrastructure using any of the widespread access ISPs.