Utilizing RSS Feeds by Web Request and Web Service Models in the .NET Compact Framework

Andrew Dutt, Sarah White, and Zornitza Prodanoff
University of North Florida
4567 Saint Johns Bluff Road
Jacksonville, Florida 32224
{a.dutt, s.white, zprodano}@unf.edu

Abstract - The rising popularity of Mobile Computing creates new methods for accessing Web content. At the same time Really Simple Syndication (RSS) feed is another new trend of constantly updating Web documents that are available on the Internet. The questions arises how to most efficiently access such new type of Web content from mobile devices. This study compares the performance of the .NET Compact Framework when accessing RSS from a direct HTTP request and through a Web Service based approach. Our results show that processing requirements need to be considered into the decision which model is appropriate as the Web Service architecture places much higher processing requirements on an actual remote PC thus alleviating processing times on the mobile device.

Keywords: .Net Compact Framework, HTTP, Web Services, RSS

I. INTRODUCTION

Static Web documents, or pages that contain no CGI scripts or database output or any other means of customized content, are being published in large volume on the Web today. Such non-interactive pages are downloaded by any user as the exact same copy of the sought for document, that is, containing the same content: e.g. embedded images, text, or video and audio objects. A study estimated that there were a total of 167 terabytes of static Web pages in 2002, or about 4.8 billion pages [10]. Today information is published and retrieved on the Web not only in much larger volumes than before, but in a different way, e.g. as blog entries, news headlines, podcasts. Users can access a particular Web site multiple times to check for any updates to it. In the last few years this traditional approach is being used less often since many Web users visit more than one site when they follow news developments or simply research a topic of interest. Once any favorite Web sites are identified by a user, in order to follow up any developments and updates to the content a user has to typically revisit all of them on multiple occasions. This usage pattern can fast become a time consuming and elaborate task. To alleviate the user from such burden, RSS feeds are used today.

A Really Simple Syndication (RSS) feed is a series of XML formatted files for Web page syndication. Typically an RSS feed user retrieves these files containing headlines and short descriptions for each news event and a link to the source of the news – all presented on a single Web page.

The Dell Axim x51 used in this study are typically equipped with CPUs such as the Intel PXA270 520MHz processor and volatile storage of 64 MB of RAM. This is clearly much different then the typical desktop with about 2 GHz dual core processor and 2.0 GB of RAM. This much stricter processor and memory requirement means that heavy PDA processing causes a noticeable pause in the User Interface and the program in which is running in the background. For example, the TechNet’s blog Main Feed http://blogs.technet.com/MainFeed.aspx is 147kB XML.

Preparing this much data for processing and parsing is resource intensive for a mobile processor. The experiments in this study were executed on a Windows Mobile 5.0 platform. The source code used to obtain the time measurements results was based heavily on the Data Points program from John Papa [1]), deploying it to a mobile device, and measuring the times that certain operations required for processing.

II. THE .NET COMPACT FRAMEWORK

Released in 2003, the operating system independent .NET Compact Framework is built for accessing a wide array of functionality at little cost to disk space for the smart phones and PDAs it resides on. The .NET Compact Framework addresses the trend of mobile
device users wanting to access all of the web’s services through their compact devices.

The .NET Compact Framework addresses four main goals of the smart device development community. These goals are: to target multiple platforms using a subset of .NET; to leverage the VS.NET (Visual Studio.NET) technology; to support true emulation; and to support Web Service enabled devices.[3] Though the compact framework is based on the desktop version of the framework, it has many important differences that address the unique nature of devices with resource constraints. Regardless of these differences, the compact and desktop versions of .NET share the same basic architecture.

The .NET Compact Framework is made to be compatible with a variety of operating systems. This is made possible through the use of the Platform Adaptation Layer. The PAL provides an abstraction between the API’s of the host operating system and the .NET Compact Framework’s Requirements. [2] Layered on top of the PAL is the Execution Engine (EE). The EE is the core manager of everything that happens in .NET. Within the EE is the CLR, responsible for managing the execution of code that runs on the .NET framework. This task is accomplished through the use of 2 .DLL files developed for this purpose.

The CLR runs Microsoft Intermediate Language (MSIL). The MSIL is produced by the compiler as a processor- independent language. [2] In this way, the .NET compact framework is ultra-compatible with many of the different processors specially designed for mobile devices. The MSIL is then again converted to the processes native machine code by the just-in-time (JIT) compiler. Adding to the frameworks compatibility across multiple platforms, there is a JIT existing for each processor type supported. From this JIT compilation, any processor supported by .NET can run the .NET applications in the processor’s native language. These processors run the Windows CE operating system and include: StrongARM, MIPS, x86, SH4 and XScale, among others. [2]

An important aspect of the JIT (Just in Time) compiler is that it executes only new lines of code when the program reaches them [2]. This saves loading times in programs, but is an obvious detriment the first time a button is pushed or a form’s component is loaded. Subsequent button pushes or form visits are noticeably faster. This also applies for web service and HTTP requests as well, as the results from the experiments will show.

Figure 1. .NET Compact Framework architecture.

The architecture of the framework up to this point illustrates an important concept. The low-level programming expertise required to program at the hardware level is well shielded by the PAL and the EE. Developers can program at higher levels, allowing concentration in developing what a device can do without spending excessive time worrying about how it does it.

Layered on top of the Execution Engine are extensive Native Support and Class libraries. These libraries support flexible development of applications for the targeted device. The base class libraries are the fundamental building blocks of each application development in the .NET Compact Framework. [2] These libraries provide important functionality such as GUI controls, collections, globalization, and most importantly, XML handling. Using these libraries gives the developer the advantage of efficiently reusing commonly used functionality without having to rewrite it for each new application.

Since the program was built for applying RSS from both a web service and HTTP points of view, the framework it is built upon is required for understanding the processing involved.

In addition to those described, a wide range of capabilities are provided by the Framework. These include rich User Interfaces, Input / Output (File) processing, XML parsing, Web Service consumption and threading [2]. The main component that is utilized by an RSS parser is the DataSet class, which has been optimized in the .NET CF (Compact Framework) 2.0. It is three times faster then CF 1.0 [3]. Access to all of
this functionality is done simply through Visual Studio 2005, and in the case of Windows Mobile 5.0, an additional SDK download.

III. RSS FEEDS

The mobile user has access to many different devices that can connect them to the World Wide Web at any time, from anywhere. Examples of these devices include PDA’s and mobile PC’s, and most often, mobile phones. Though these devices differ in size, technology, and price, they all present the same problem when used as a tool for web browsing. A desktop PC is perfect for viewing web pages, but when we use a mobile device for accessing the web, we need to consider the difference in resolution, screen size, and supported colors and graphics of the device, as well as computational power, memory size, rate of data transfer, and energy consumption. [6] Mobile devices are smaller and often count on processors that are scaled-down to function well within the limited stored energy supply.

We know that users have mobile devices and we know that they want to access Internet content, but how do they achieve this without downloading the classic web page? The concept of Really Simple Syndication (RSS) has been introduced as a solution to these issues.

RSS was originally developed in 1997 by UserLand and was subsequently picked up by Netscape. It has gone through many versions and improvements, and recently is being adopted by any website that wants updated content to be communicated to the mobile user. RSS is a simple approach to solving the obstacles of distributing web content across devices. [7]

RSS Uses a proxy server to adjust and adapt existing web files for use on mobile devices. Developers of large info-centric websites often use special RSS tags to mark up summary information. The RSS proxy server takes an incoming web page and looks for any tagged information. The server will also strip information such as images, video, and animation. The resulting feed is reduced approximately 80% in size and complexity, ideal for a compact device. [6]

IV. HTTP AND WEB SERVICES

The purpose of our experiment was to determine the difference in response time for an RSS feed to a mobile device when retrieved using the HTTP protocol versus Web Services. In order to put this experiment into perspective, we must first examine how the HTTP protocol and the Web Services model work.

HTTP is an Application Layer level protocol that is used to transfer data across the World Wide Web. The initial HTTP transaction starts with a request being sent from a client computer (in this case, a mobile device) to the server requesting data that resides there. The server returns the data using the TCP/IP protocol. The data being sent through an RSS feed depends on the information that has been marked up with the RSS family of XML tags. HTTP is essentially a request and response system, and this transaction will carry on in the same manner until the HTTP connection is terminated.

On the other hand, a Web Service itself is a software program, a module, delivered over a network using XML. The software components are wrapped in specific protocols for transfer, and have the ability to automatically interface with other software without the need for human interaction. There are three roles in the Web Service architecture that work together to accomplish this: a service provider, a service registry and a service requestor. [8]

In the experiment we performed, RSS feeds were initiated over both an HTTP request-response system and a Web Service approach using SOAP. With SOAP (Simple Object Access Protocol), structured data is passed between network applications using XML.[8]
The SOAP program was published on the UNF .NET server and the services were consumed using the .NET Compact Framework.

V. UTILIZING RSS FEEDS BY WEB REQUEST AND WEB SERVICES

The basis of the performed test was not to parse an RSS into a viable or sustainable User Interface, but to compare data access times for two different methods of retrieving RSS feeds.

For the HTTP request/response approach, the test program sends a HTTP request to an RSS feed, and retrieves the data. It should be clear that this would be much faster then a web service call for receiving the data from the site. However, there is a fair amount of processing that now resides on the Mobile Device. This includes transforming the response from the site into a useable format for parsing.

For the Web Service approach, the test program invokes the Web Service through an object and method series. When the Service is added to the project, Visual Studio wraps it into an instantiable object that allows its methods to be used. The Web Service is contacted by the program, which in turn contacts the same RSS site as in the HTTP approach, see Figure 3.

Then the Web Service handles the reformatting of the data, and sends it to the device. The device then does minor processing [5] to load the formatted XML for use.

The experiment measures the time in seconds to complete the two major tasks of utilizing RSS. It measures the time to make the actual web request to the respective locations (direct RSS Site and the Web Service) and the processing time of the program after receiving the response.

VI. RESULTS

Using the Dell Axim x51, the .NET program created for this experiment was tested on the device connected via IEEE 802.11b.

The test was conducted as follows:
1) Launch the program
2) Select either call by HTTP or call by Web Service
3) Wait until the call and processing is complete
4) Select the same option to send another request
5) Wait until the response and processing is complete
6) Close the program

To test the size of the XML file returned from an RSS feed, two sites were chosen for the testing. At 147kB, http://blogs.technet.com/MainFeed.aspx is one of the larger RSS feeds on the Internet. While at http://rss.news.yahoo.com/rss/topstories, the RSS XML is only 49kB, a much more average size.

To analyze the JIT compiler and the respective speed of processing, the process is done twice because the code has already been loaded into memory.
Table 1. Request and Processing Time for Accessing the RSS via HTTP or a Web Service.

<table>
<thead>
<tr>
<th></th>
<th>First Call through the Web</th>
<th>Second Call through the Web</th>
<th>First Run Processing Time</th>
<th>Second Run Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct HTTP</td>
<td>2.5643s</td>
<td>1.7021s</td>
<td>4.8882s</td>
<td>3.2099s</td>
</tr>
<tr>
<td>Web Service</td>
<td>10.078s</td>
<td>4.8841s</td>
<td>2.5185s</td>
<td>0.5982s</td>
</tr>
<tr>
<td>Direct HTTP</td>
<td>1.5288s</td>
<td>.4209s</td>
<td>3.624s</td>
<td>1.4362s</td>
</tr>
<tr>
<td>Web Service</td>
<td>4.4539s</td>
<td>1.7463s</td>
<td>2.4949s</td>
<td>0.5921s</td>
</tr>
</tbody>
</table>

Table 1 shows the results collected from the program. The delays are evaluated by using the Environment.TickCount variable. This allows the program to see how many milliseconds it has been since the processor has been turned on [4]. 20 access attempts were executed for both HTTP and Web Services.

The results show that the in the first run, the direct HTTP call is much faster than a request through a Web Service. The web request time on the second call to a Web Service is drastically less than its first call. The Web Service architecture places much higher processing requirements on an actual remote PC, alleviating processing times on the mobile device. The direct HTTP model shows that while the processing is higher on the mobile device, the speed of the HTTP request is four times faster than the first run of the Web Service model. These results allow for different conclusions based on the type of program architecture that is required.

VII. CONCLUSION

The performed experiment has shown that HTTP and Web Services both allow for different ways to utilize an RSS feed that results in different request and processing delays even when total end user delays are similar.

Both models could be acceptable for utilizing RSS, or other Web application processing. The processing requirements need to be considered into the decision which model is appropriate as the Web Service architecture places much higher processing requirements on an actual remote PC thus alleviating processing times on the mobile device.

REFERENCES


