Printing from Rich Internet Applications (RIAs): A Standardized Approach

Authors: David Uyttendaele* and Chuck Gehman*

Keywords: HTML, integration, Internet, servers, software

Abstract

Until fairly recently, most Web applications employed a client-server architecture in which the browser acted as the client, and a Web or application server performed substantially all processing. This approach is not unlike the old days of "terminal and mainframe," although Web applications are often simpler and easier to use by design.

RIAs (Rich Internet Applications) and their HTML/AJAX counterparts change this by downloading software into a Web browser on the client machine, which handles rendering the application's user interface, and communication with the server. This provides the ability for developers to offer "rich" user-interface functionality which is not possible with only HTML and standard browser-based Web applications. The result is the RIA approaches the sophisticated look and feel of desktop software. This, along with the benefits associated with Software as a Service (SaaS) and Web 2.0 applications, has resulted in growing popularity of this development paradigm.

Technology giants Adobe and Microsoft are leading the pure RIA charge with their respective Flash (and its Flex development technology) and Silverlight technologies, followed closely by a global cadre of developers, independently and in standards bodies, enhancing HTML and adding AJAX (Asynchronous Javascript and XML) to create such apps without the need to embed proprietary software. With the focus inarguably on the computer screen, and on multi-media applications (think YouTube), little priority has been given by developers to generating hard copy output. At this writing, there is no ability to print application-controlled, formatted content contained in RIAs anywhere but from the desktop computer.

---

* Mimeo.com, Inc.

The challenge is that all three of the aforementioned technologies can print to a local printer in a rudimentary way, with basic controls, but cannot print with any sophistication from a server. As these technologies gain in popularity, they will contain large volumes of the world's information, much of it suitable and desirable for print output. As individuals and employees in corporations inevitably adopt "Web 2.0 technologies" to replace applications that are today done with desktop software, solutions must be created to enable Rich Internet Printing (RIP).

The opportunity is to allow the developer of an Internet application, whether they are using HTML/AJAX, Flash or Silverlight, to print in a graphically rich way, based on standards, from/to the "cloud." One solution would be to create a new kind of "print driver," that uses a standardized way to model the print stream, taking advantage of the richness of the platform it is installed on, but directing the stream out of the app/platform.

Introduction

Web 2.0 applications are generally characterized as a second generation of Internet-based services—and include things like blogs, social networking sites (MySpace, Facebook, LinkedIn), wikis, communication tools and more. These applications are as much about user generated content (UGC) as they are about creating community. Blogs and Wikis, exciting new collaboration technologies, and emerging "value-added storage" where you can "park" your digital assets (documents, graphics, music, etc.) for use by individuals and workgroups, colleagues, employees and business partners are changing the way business is done. Some content is created in the Web 2.0 application natively, while other content is posted in native document formats. Much of the content that users put into Web 2.0 applications is suitable for high quality, complex document printing and subsequent distribution.

Web 2.0 applications use new Web technologies that provide richer user experiences from within a standard Web browser. Older websites with static HTML pages, or even dynamic pages driven by content management systems, appear obviously limited when compared to the new ideas and rich functionality of Web 2.0 applications. While it might be theoretically possible to create something that might be considered a Web 2.0 application using plain old HTML, and other more "traditional" Web development techniques, they almost always are developed today using a selection of one or more Rich Internet Application (RIA) technologies.

Technologies employed in the development of RIAs include AJAX (Asynchronous JavaScript and XML), and ubiquitous proprietary technologies Adobe Flex and Microsoft Silverlight. In addition, RIAs often employ CSS

(Cascading Style Sheets), RSS (Really Simple Syndication), Mashups (combined content from more than one source in an integrated experience), and Web Services of one sort or another.

In general, RIAs require modern Web browsers in order to function. Some RIA platforms depend on advanced JavaScript engines in the browser for client-server communication, and DOM (Document Object Model) Scripting and advanced CSS techniques to enable the rich user interface. Other RIA platforms require the installation and maintenance of plug-ins, which decrease browser-compatibility issues, but introduce additional challenges.

All of this leads to difficulty in producing professional quality, formatted output from RIAs. This paper will briefly outline the history of graphically rich printing in computing, the current internet printing situation, and the challenges it presents; then will provide an inventory of currently employed strategies for printing from RIAs, and finally explore some directions that could be taken to improve the situation.

Background

In many ways, printing from RIAs can be equated to printing before Windows and Macintosh platforms were developed. Over time, display and printing technologies were "rationalized" and it became easier for developers to support high quality, professional printing in addition to formatting output for the display. Today, Quartz on Macintosh computers, and Windows Presentation Foundation on PC create a desirable environment for printing. The software development technologies, and the delivery environments that provide the end-user experience are the platforms for future applications.

In the early days of the World Wide Web (WWW), Web pages were very simple. In many ways, they could be compared to Character Mode (i.e., DOS) applications. In that environment, printing was less of an issue. You simply dumped the content of the browser, which contained little formatting, to the printer. There is a relatively obvious parallel between the evolution of OS-based printing capabilities, as well as to the evolution of the Web platform and the printing capabilities (or, perhaps more to the point, printing *needs*). To understand this evolution, let's briefly look back at how Macintosh and PC platform printing developed over time.

In the earliest days of personal computers (late '70s, early '80s), printers were directly connect to the computer from which output would be generated, usually via an interface defined by an electrical specification, for example Serial (EIA RS-232) or Parallel (IEEE 1284). These interfaces actually started out serving the same purpose on mainframe and mini computers, and assumed close computer-printer proximity. They were so unspecific that in many cases, cables

had to be made for specific computer-printer combinations. In terms of software, there was no platform-defined printing capability. The only thing the platform defined was the architecture via which data would be sent to the specific hardware interface. So to print a document, applications themselves had to build in support for the specific output devices (printers) they would support. Every time a new printer was manufactured, every single application provider had to build support for it.

Then, as networks (specifically TCP/IP- Internet Protocol) began to arrive on the scene, LPR printing (which began on Unix machines) became a good way to "separate" the hardware from the software, while adding necessary features to support printing in multi-user, multi-application environments. Although it began to address the requirement that printers be in direct proximity to a workstation, applications still had to be "printer-aware" to create any sophisticated print output.

In the early 1980s, Adobe created the PostScript language and a variety of technologies to support its use. At the time, PostScript was a major change in the way printing was approached, because instead of specifying how an already rendered image should appear, it specified how the image should be rendered. Although the difference might seem subtle, it would soon revolutionize the printing industry and eventually be applied to on-screen imaging.

In 1985, Apple's LaserWriter became the first laser printer on the market to ship with PostScript, and along with Aldus Pagemaker on the Macintosh, and the soon to follow Adobe Illustrator application, it spawned the desktop publishing revolution.

Not long after, Steve Jobs founded NeXT Computer. The computer scientists at NeXT realized PostScript's power could be extended to the computer screen, and worked closely with Adobe to produce a variant of PostScript called Display PostScript for on-screen display. This was a revolutionary step, the unification of the screen and hardcopy, both from the end-user perspective, and from the application developer's standpoint. This became an integral part of the NeXT Computer's operating system, NeXTStep.

After Apple acquired NeXT in 1996, the "digital paper" metaphor played a paramount role in the development of Mac OS X. But while the abstraction remained the same, Adobe's PDF specification (an enhanced subset of PostScript) was chosen as the model, instead of PostScript itself. Apple created a technology they called Quartz. This was the foundation for Mac OS X's graphics capabilities.

The real win is the ability to write one chunk of code that can draw on the screen or printer. OS X has such a unified imaging model. Quartz is the text and

graphics rendering library: It supports the user interface, including on-the-fly rendering and anti-aliasing. Quartz's internal imaging model is not PDF exactly, but is very similar to the PDF imaging model, making it easy to output PDF to multiple devices. The result is very robust professional printing capabilities, made easily available to the platform's software and application developers.

In the "non-Apple & Adobe" evolution of modern digital printing, (hesitate to call it the "Microsoft" world, because so many companies drove the development), Hewlett-Packard came out with the HP Laserjet in 1984, also featuring what has now become known as a PDL (Page Description Language, of which Postscript is also one): PCL – Printer Command Language version 3. There are also numerous other PDLs, like AFP and IPDS (from IBM.) These PDLs, including Postscript, created much necessary rich-feature sets for printing, and abstracted the logic of printing from the actual device itself.

PCL 6 was introduced in late 1995, as a major upgrade, providing a stack-based, object-oriented protocol, similar to PostScript. PCL 6 was designed to match the drawing model of Microsoft Windows Graphics Device Interface (GDI). In this way, the Windows printer driver simply passes through GDI commands with very little modification, leading to faster return-to-application times. Prior to this, GDI printers (almost all other printers) almost universally sent a compressed bitmap to the output device.

But until Windows Vista, no one had come close to the model that NeXT had envisioned in the 80s, and that Apple brought to the "mass market" with OSX. With Vista, Microsoft introduced XPS (XML Paper Specification) and the Microsoft Windows Presentation Foundation, to enable rich end-to-end color document and photo printing and address many limitations of the existing GDI-based print path.

Windows Vista printing brings an advanced set of document services to the Windows platform. Enhanced color support enables high-end printing by supporting printing with more than four colorants, which is required for customers with prepress applications. The new print architecture can communicate application-generated extended color information to wide-gamut color printers. Device drivers can access and control color information from within the print pipeline.

When printing from applications built on the Windows Presentation Framework or when directing output to XPS Document–based printers or drivers, the XPS print path reduces or eliminates image data conversions and color space conversions wherever possible, enabling high-fidelity print output. XPS printing provides more faithful rendering of graphics attributes such as gradients and transparency though native support of these attributes in the XPS spool file format. The XAML in the XPS Document format is compatible with Windows

Presentation Foundation XAML. When printing from a Windows Presentation Foundation application, Windows removes animations and converts video and three-dimensional (3-D) elements to images. All other graphics data is represented in compatible graphics primitives that are ideal for device consumption. The device or driver directly consumes the printing version of Windows Presentation Foundation output.

In many ways, printing from RIAs can be equated to printing before Windows and Macintosh platforms achieved their current level of sophistication. On those platforms, over time, display and printing technologies were brilliantly "rationalized" and it became easier for developers to support high quality, professional printing. Today, Quartz on the Mac, and Windows Presentation Foundation on PC create a desirable environment for printing. Sophisticated printing became a cornerstone of the success of the OS, having an enormous impact on the usability of applications. This is what is missing today in the Web 2.0 world, and is a necessary part of the evolution of this next generation computing platform.

When Web pages were simpler, printing was less of an issue. You simply dumped the content of the browser, which contained little formatting, to the printer. Now, we need a print solution for RIAs that achieves parity or surpasses the Macintosh Quart/Windows Presentation Foundation solution for the desktop. But there's still one other major hurdle to overcome: Today's computing model has shifted from local data and processing on the desktop to many devices, including mobile phones, sharing processing and data in the internet cloud. As described earlier RIAs becoming the user interface (UI) to this new OS in the cloud. Since the OS is now in the cloud it should be logical to conclude that in addition to printing from the desktop (or in this case, the browser/application platform) to a locally attached device, we also need to "print to the cloud."

Challenges

As we have discussed, much printing today still assumes computer-printer proximity. This has always been obvious challenge for professional Print Service Providers (PSP), even in the absence of RIAs. It necessitates a workflow that involves first creating a document (in the "traditional desktop computing model," in a desktop application), and then "saving it" in an output format (e.g., PDF), or using another software package to convert it to the output format, then sending it (using a variety of methods) to a PSP.

What is happening now is that the Web has changed expectations of printing overall. In the past, a user may have been satisfied to simply acquire pages from an application. Once you printed those pages, you would assemble them yourself (i.e., binding=stapling), or you employ an "offline" methodology involving sending them to a printing company (as in the desktop software

example above) to perform "traditional" print production workflow, involving graphic arts processing to create a finished product. But emerging online applications (like Photo sites, as a prime example) provide the user with "invisible" manufacture on demand capabilities, so the expectation is no longer to just get pages from a printer, but a final product.

While much content is now being generated natively on the Web, Web 2.0 developers (whether platform or application) view even desktop printing as an "afterthought" at best. Documents built natively in Web 2.0 applications cannot be easily printed, in professional quality. Printing from workstations using certain RIA technologies, specifically Flash, is rather sophisticated in taking advantage of the OS printing platform, but printing "to the cloud" (e.g., to a PSP) presents bigger obstacles. The challenge is that today's popular RIA platforms can often print to a local printer in at least a rudimentary way, with basic controls, but cannot print with any sophistication from the applications server or workstation to a print service provider (i.e., in the cloud), without separately installed software (proprietary print driver) or very involved, specific proprietary printing code (create a PDF and send).

If you are a print service provider today, there are few direct benefits from anything happening in this Web 2.0/RIA world. If the print community doesn't address this lack of printing capabilities, it will lead to us having to spend time handling technology challenges in printing, and quite possibly help to accelerate the movement of content to non-print digital delivery. However, it is inevitable that people will want to take content from RIAs and have it printed professionally. This is clear from millions of dollars generated by Photo sites like Kodak Gallery and Snapfish. But document workflows are much more complicated. In either case, professional printing operations today must create their own print production workflows to support RIA-resident content. The technical challenges include the mechanisms by which print-ready files are generated, as well how to actually send, and receive them. Technically capable printing operations will win the new customer's business, while others will not be capable of participating.

The table below provides a short inventory of examples of some popular Web and RIA applications and services. This by no means is meant to be exhaustive or complete, but rather intended to supply the reader some context. To see an overwhelmingly comprehensive list of Web 2.0 companies, visit http://www.go2web20.net. Each of the examples has deployed a system that is "collecting" user generated content, much of which can be considered appropriate for professional printing. The level to which the individual application provider currently supports printing, whether to a locally attached printer or to a PSP, is briefly described.

Web 2.0 Application	Description	UGC in the app that is appropriate for printing	Support for printing
Blogspot, Blogger, Wordpress, Movable Type, TypePad	Blog sites	Blog articles and photos	Local printing (from browser), download and print (PDF), PSP (third parties sites)
Disney.com PBSkids.org, Noggin.com, kids sites	Kids media and activity sites	Characters, coloring books, activities	Local printing (from browser and flash) some products use a PSP
Facebook	Social Networking site with more than 220 million global users	Photos	None, but has an application architecture allowing third parties to add functionality (Blurb, Hotprints)
Flickr, Kodak Gallery, Snapfish, Photobucket, Picasa, Shutterfly.	Photo sharing sites	Photos, posters, photobooks, more…	Some local printing (from browser), primarily PSP
GoogleDocs	Office Productivity Suite	Office documents of all types	Local printing (from the browser)
MS Office 14 Web Applications	Office Productivity Suite	Office documents of all types	Unknown, still in development
PPTShare, Slide Share, SlideRocket, 280 Slides	Online Presentation creation and/or sharing offered as Software-as-a-Service	Presentations	Local printing (from Flash), download and print (from PDF or PPT) or a PSP
Salesforce.com	Salesforce automation, and application platform	Reports	Local printing from the browser or export to excel for printing
Scribd, docstoc, Issuu, edocr	Document Sharing	Documents of all types	Local printing (from Flash), download and print (PDF, native file)
Wikipedia	Wiki encyclopedia	Articles	Local printing (from browser), download PDF, or PSP (PediaPress)
ZOHO	Office Productivity Suite	Office Documents of all types	Download and print (PDF) locally

Table 1. *Examples of some Web 2.0/RIA applications, denoting their ability to support professional printing.*

In very high-level summary, these providers can be broken down into two groups based on the how they handle or create UGC. First, there is content that originates in the RIA (i.e., documents built on the Web). Second, there is content from non-Web applications (e.g., PDF files or Word documents) that is being "posted" into Web 2.0 repositories. Some applications take that content and convert to a "native" format for the application (like Scribd, and its iPaper format), while others keep it in its native format. Some providers who convert also keep around the native document so that it can be retrieved later. Others do a "roundtrip" when the user wants to get the document back in its original form.

Until this problem is solved in a robust way (discussed in the next section of this paper), we're going to see only highly motivated companies with niche professional printing applications knitting them into RIA/UGC applications and services. One timely example is Hotprints (www.hotprints.com), which has created an application used within the phenomenally popular social network Facebook. This new U.K. company, whose product are books called "HotBooks," is in public beta now. A Hotbook is a 25x20-cm color photo book with 8 sheets and 16 total printed pages, 6 photos per page. It's an exciting application because the books are completely "cookie cutter." The company constrains the size of the photos that will appear in the book, so that almost any image that looks good on the screen in Facebook will also look good in the "Hotbook" you order. The company sells the books for $2.99 plus shipping, which is about $0.75 to the United States. They will likely sell millions of books because they are embedded in Facebook, which at this writing has the richest collection of photos on the Web, already-tagged and ready to be turned into printed books.

Another example is the collaboration between my company, Mimeo.com, and SlideRocket. In this case, our two companies had to work together to invent a way that Sliderocket presentations could be professionally printed by Mimeo. SlideRocket is part of this new generation of RIAs. It is a Web 2.0 online presentation application that allows users to produce slideshows that have some key advantages over creating presentations in desktop applications like Microsoft PowerPoint. SlideRocket runs in a Web browser, instead of requiring the installation of software on your computer. It features collaboration tools that let users share slides and other assets (like graphics and movies) between presentations. Because the application is Internet-based, these users can be geographically dispersed and still share nicely. This collaboration, and user-generated content sharing, is what makes SlideRocket a unique and powerful Web 2.0 application.

The SlideRocket relationship with Mimeo involved the use of the MimeoConnect SOAP API technology. This is a set of Web Services that let partners submit files, define product intent, get quotes and proofs, and submit orders, recipient addresses, shipping methods and payments. This technology

solves the problem of delivering content files and placing orders; primarily the communication between our two applications in the cloud (i.e., cloud to cloud).

With that connectivity in hand, we then needed to solve how SlideRocket would take their rich content (screen-based Flash presentations) and turn it into PDF files to send to Mimeo (this was a Mimeo requirement, due to our automated PDF-based production workflow). Finally, we needed to create a way to let the non-technical SlideRocket user to specify the product intent to communicate to Mimeo for manufacturing.

To solve the former problem, SlideRocket created a server-based application that "printed" the "slide deck" to a PDF file. To solve the latter problem, Mimeo created a special interface designed to reside within the SlideRocket Flash application. When the user chooses "Print" from the SlideRocket menu, they can either choose to print on their local printer, or print "high quality with finishing options" via Mimeo.

The interface we built incorporates a subset of Mimeo's new photo-realistic document viewer. We call it MimeoProof, because it is about as close as one can get to holding a print product in their hands by viewing it as an image on the screen. The Sliderocket version lets the user see their presentation, visually depicting paper stocks, covers and binding choices. This is an important part of helping the non-technical user specify product intent, it gives them the confidence they need to proceed with the order.

Once the user places their order on the SlideRocket site, e-commerce and print-specific transactions flow: we are sent a PDF file containing the slides, and an XML transaction containing the order information (desired quantity of books, destination, shipping method, etc.) The next day, or at some point in the near future depending on the user's choice, a beautiful bound version of the Sliderocket slides shows up on the user's doorstep in whatever quantity they specified.

Both of these examples demonstrate the difficulty of printing from RIAs. In each case, much effort by software developers was required to get the content into the printing operation in a usable way. They also demonstrate how powerful solutions to these challenges can be, for the professional PSP.

Investigation

We previously discussed the fact that developers of Web 2.0 applications address printing as an "afterthought." But in the overall scheme of global media, Web 2.0 should be viewed in the context of "channels of engagement," which include PCs, Mobile, Consumer Electronics (i.e., gaming platforms), and print channels. In terms of print itself, these channels incorporate traditional print

publishing, electronic documents, interactive media, websites, and RIAs. While the audience is present and receptive, and the content in many cases is suitable, there are a couple of technical obstacles that make it less likely Web 2.0 developers will incorporate professional printing into their applications, even if they desire to do so. Today, only the most motivated developers will do this. Hotprints, as an example, has business model that is centered on the creation of a printed product, so they needed to solve these problems. ZoHo (see Table 1) similarly had to do so, in order to compete with Microsoft Office. But most of these websites and RIA apps are motivated by online user experience, not print, and we believe will never even attempt to address these challenges, without it being much easier.

The reason it is difficult is that RIAs are built using new technologies like Flash, Microsoft Silverlight, Ajax/HTML, or one (or more) of dozens of other emerging technologies. At the desktop Flash does offer WYSIWYG printing, but does not provide the ability to "print to the cloud." Silverlight does neither effectively. HTML/CSS is only as good as what the combination of HTML/CSS and each browser allows. So a big part of the adoption limitation for print is that even the platform developers are not focused on print. A notable reason for this is that the platform developers are instead consumed by a focus on interactive. Microsoft had a great opportunity to solve these printing problems with Silverlight, using XPS and XAML, but it appears that was de-emphasized in their market-focused desire to achieve technical parity with Adobe and Flash-specifically in the video area, in the interest of recovering market share.

Most technical readers of this paper are familiar with the technologies mentioned above. Flash, for example, is from one of print's favorite vendors: Adobe. Further, most are aware of Google Chrome, which is being touted as an "application platform," not a browser. No doubt we've heard about the success of salesforce.com, too. Applications *are* moving to the Web. Increasingly, we will see graphically rich applications moving to the Web. The user can print to their local printer by doing "File-Print" from the Web browser—but only in limited circumstances will the resulting output be professional quality.

So there clearly is a need to create a standardized solution to printing from these applications, both to local output devices and "in the cloud." Let's examine some of the ways developers have solved this problem to date.

Technology	Platform	Printing Capability
Pure RIA	Flash	Client OS printing to local printer (high quality, easiest to implement)
		Control HTML/CSS in browser from flash and print via browser. (good enough output, some challenges to implement)
		Cloud Generated PDF downloaded to local printer via browser (most portable, involves coding and support of 2 applications, RIA and server)
		** Cloud Generated PDF to Cloud print provider (Truly professional quality product possible, involves coding and support of 2 applications and working with a proprietary PSP API)
	Silverlight	Control HTML/CSS in browser from Silverlight and print via browser to local printer (good enough output, some challenges to implement)
		Cloud Generated PDF/XPS downloaded to local printer via browser (most portable, involves coding and support of 2 applications, RIA and server)
		** Cloud Generated PDF to Cloud print provider (Truly professional quality product possible, involves coding and support of 2 applications and working with a proprietary PSP API)
DHTML/AJAX		HTML/CSS in browser print via Client OS to local printer (rudimentary output, easy to implement)
		Cloud Generated PDF downloaded to local printer via browser (most portable, involves coding and support of 2 applications, RIA and server)
		** Cloud Generated PDF to Cloud print provider (Truly professional quality product possible if assets are high res, involves coding and support of 2 applications and working with a proprietary PSP API)

Table 2. *Printing destinations/capabilities of popular Web 2.0 platforms.*

Without exception, with any of these platforms, if you are trying to create completely controlled output to the cloud today, you are constrained to building a separate application that interprets your front-end content in a different way and then generates a print stream.

The result is a major software development burden to support the printing subsystem on an ongoing basis, with all the maintenance issues and divergent application issues associated with supporting two platforms, one for the screen and one for hardcopy. More complex documents and formatting necessitate more difficult development and arduous ongoing maintenance tasks.

Once the developer has solved this basic problem of creating a document model within the application, then the choice has to be made as to how to create something that is compatible with a PSP's production workflow. Choices might include generating PDF or XPS files, or creating high resolution images.

Furthermore, since Web 2.0 applications are characterized by providing the user with much more control of their overall experience, in an ideal world print should be able to be controlled in the same unique and wonderful way. It becomes a question of whether the output is under the control of the content owner (or RIA developer) versus the content user. However, the amount of "control," actual or perceived, varies from one application to another.

For example, a German company called PediaPress recently launched an application that lets you choose pages from the popular Wikipedia, and through a user-friendly interface, create a 8x5.5-in. perfect bound book with a color cover and black & white interior. The finished book includes a table of contents and index, which are automatically generated. The company is currently selling the books start at US$8.90 for 100 pages, and they are printed and shipped worldwide within 2–15 business days. We ordered a book, and it was produced and shipped by LightningSource,

There are a couple of important things to study in this implementation. First, Wikipedia's content is uniformly structured. This makes it easier to manipulate in general, than systems that accept freeform content in native application file formats. Second, PediaPress also created and licenses as Open Source some of the technology they use to access the content in WikiPedia. This was, according to new reports, apparently required under their contract with the non-profit Wikimedia foundation. The goal is to ease the reuse of wiki content in other media or applications, so other developers can use their software to build an application to print books from content stored in WikiPedia, too. PediaPress's tools include a Python (programming language) library for parsing MediaWiki (the technology underlying WikiPedia, and now other systems) articles, and a library for writing PDF documents from MediaWiki articles, as well as parts of

the interface to WikiPedia itself, a "Collection extension" for MediaWikis that would let a developer collect articles and output them in various formats (PDF, ODF, XML, etc.)

In this case, the resulting printed product matches the fidelity of what the content looks like on the screen very well. This is important, because there is an expectation that the same level of rich tools the user has to control the experience in on the screen, should be available to control printing, resulting in a high fidelity printable document.

As described earlier, a developer seeking to create professional output today by printing to the cloud from an RIA application has one basic option: to write a duplicate proprietary server based application (custom built and/or leveraging licensed software) to:

1. Generate a PDF from their natively generated, or stored front-end content
2. Write proprietary software to submit their PDF content
3. Collect intent and generate job ticket information and order details
4. Send this all to a Web-enabled PSP in a way that PSP can accept, which may also require additional coding

We believe it is worthy of additional investigation to print directly from each of the three RIA platforms discussed earlier; since it is not commonly employed and could provide a valuable, easier alternative. Some of the following discussion requires at least limited knowledge of the development platforms discussed. It's out of scope of this paper to go into depth on these how these technologies work, or to provide a tutorial on object oriented programming, but you can readily find the deep background of what is described on the Web.

An efficient and powerful (as well as simplified) way to print from the Flash/Flex platform would be to enable the developer to use the same mechanism and code path to print to the cloud as they can use to print locally today. In Flex 3 (at this writing, the latest iteration of the technology), the developer simply uses an instance of a *PrintJob* class, to *start()*, then uses *addObject()* one or more times, and finally employs *send()* print output to the OS for local printing.

To address printing in the cloud, a more universal (or Web service) version of the PrintJob class would be created, perhaps called *UniversalPrintJob* that inherits it interface and capabilities from *PrintJob* but that adds the capabilities necessary for formatting and redirection of the output to the PSP. This class could provide an alternate implementation of *PrintJob* that would directly create a PDF or some other portable format from any Flex object added to the print job through the *addObject* method providing a great solution to the problem of creating an appropriate printable format.

Continuing this idea there could be additional properties, methods or parameters that could provide the additional information needed when the *Send* method is invoked to submit the content to a RESTful webservice. By simply providing a URL, additional job information and order data formatted as XML or JSON the send method could use the internal Flex HTTPService and/or WebService classes to seamlessly send this print job to a Web-enabled PSP.

Microsoft had an amazing opportunity for Silverlight to solve all these problems with their XAML and XPS technologies, but unfortunately there isn't even an ability to robustly support local printing in Silverlight 2. At this writing, you will find many references to incomplete and/or cumbersome point solutions to this problem on the Web. This does mean, however, there is still a great opportunity to define a universal printing model from scratch, taking advantage of what could be possible in Silverlight.

Since Silverlight basically uses XAML as its imaging model and XPS is a subset of XAML it would seem to make sense that one could create methods for converting Silverlight content to XPS, then send this data to the Cloud for printing in much the same way we described doing this with Flex, only this time using XPS instead of PDF. XPS is not (yet) universally accepted, but then again neither is Silverlight. That said, it is just as feasible to generate PDF from Silverlight as it is from Flex.

The pure HTML/CSS/AJAX world can also find a similar model, although depending on where the content is located it may not be as universal. CSS already has support for media types "screen" and "print." By setting a Stylesheet's media attribute to "print" you can customize HTML content specifically for printing. Often the developer defines an entirely separate Div, invisible to the user, styled for print. This same model might work for printing to the Cloud. One could simply post that special *Div*'s content directly to a RESTful API (a popular Web 2.0 inter-application, inter-site, communication method), which is handled by software that would have to be built or acquired by the PSP. This would only work for content that is either entirely HTML text or that has an absolute path URLs to any CSS or images. Then, of course, the PSP would have to possess software that can support HTML as a valid print format, or convert the incoming HTML to PDF or another format for output, via conversion service (which would also have to be built or bought).

Another approach could involve a JavaScript library or jQuery plugin that would walk the Document Object Model (DOM) of this aforementioned *Div*, and would appropriately resolve any *src uris* and *css* so it would convert easily to a printable format. In this case, the software might also incorporate the capability to generate PDF or XPS directly, greatly simplify the number of steps necessary on either side to complete a job submission.

The goal we would look forward to would be for the three platforms to support a standardized way to model the print stream and handle the submission metadata.

Conclusion

A creative and industrious few professional Print Service Providers, with vision and technical expertise, are beginning to reap the benefits of the growing body of user generated content in Web 2.0/RIA applications, but many others cannot participate in the current status quo.

The likelihood of these challenges being address by an industry standards body is relatively low, because of the fast moving nature of the RIA development world. There may be major developments from the platform vendors which address at least some of these challenges in the future.

In an ideal world, the three platforms would employ a standardized way to model the print stream and the submission metadata. The authors wish that, as a result of our investigation, we could name a single solution to all of the problems articulated here. But we aren't there yet. The biggest beneficiary of a standardized approach, in our analysis, would be the Print Service Provider, because it would free us from having to build and support multiple technologies to manufacture professional print from RIA content.

It is very possible that the best way for us to achieve that goal might be via a collaborative approach, similar to Open Source efforts. A community that forms around this need should involve major industry software vendors, content owners, Web 2.0 developers and service providers, and print service providers, among others. The Open Source model is a pragmatic way of building software quickly to address the needs of a rapidly changing environment, and could benefit many of the constituencies in this ecosystem. We will continue in our quest to address these challenges, and we encourage and invite anyone interested to contribute their ideas to an emerging community.

Literature Cited/Selected Bibliography

Ferrailolo, John. 2008. "Good News for AJAX – The Browser Wars Are Back!" *Ajax World Magazine*. http://ajax.sys-con.com/node/547209.

Russell, Matthew. 2005. *What Is Quartz (or Why Can't Windows Do That?)*. O'Reilly Media.

Apple Computer. 2007. *Graphics & Imaging Overview.*

Paquette, Mike. 2006. *Why Apple Didn't Use X for the Window System?* Apple Computer.

Microsoft Corporation. 2007. *XPS and Color Printing Enhancements in Microsoft Windows Vista*.

Thricovil, Raghunath Rao and Lambda, Prabhdeep Singh. 2008. *Introduction to RIA.* Adobe Systems.

Sylvester, Carrie. 2009. "A picture may be worth a thousand words… but it costs only $2.99!" InfoTrends InfoBlog.

Follesoe, Jonas 2008. *Printing in Silverlight 2 Using CSS and ASP.NET AJAX4*. http://jonas.follesoe.no.

Bernius, Matthew. 2009. "Taking Open Source Publishing Further: Tools from the Open Publish Lab at the Rochester Institute of Technology," presentation from the O'Reilly Tools of Change for Publishing Conference.

Wauters, Robin. 2009. *Print Your Favorite Articles as Books, Courtesy of PediaPress*, http://www.techcrunch.com.