

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
LOUVAIN SCHOOL OF MANAGEMENT

Richer Web Applications

On Trends, Techniques and Standards

Supervisor: Prof. J. VANDERDONCKT

Master Thesis:
Partial Fulfillment of the Requirements for the Degree of
Master in Business Engineering

Charles MARSILY

2323.05.00

INGE22MS/G

charles.marsily@student.uclouvain.be

Academic Year 2009-2010

Acknowledgment

I would like to thank my supervisor Prof. Jean VANDERDONCK for providing me a subject as interesting as rich web applications. Writing this document has redefined how I use the Web today.

* * *

In addition, this work might not have been possible without my family's continuous support.



CONTENTS

CONTENTS	i
LIST OF FIGURES	iii
INTRODUCTION	v
Disclaimer	vi
Content & objectives	vi
1 UNDERSTANDING RIAs	1
1.1 Definition and criteria	1
<i>Thick vs. thin applications</i>	3
<i>Client/server applications vs. web applications</i>	4
<i>RIA features and characteristics</i>	5
<i>Deployment alternatives</i>	6
1.2 Why RIAs are emerging?	6
1.3 Limitations	8
1.4 Web applications vs. native apps: what to choose for the iPhone?	9
1.5 RIAs as a part of a broader Web 2.0 era	11
1.6 RIAs for enterprises	14
<i>Customer-oriented applications</i>	14
<i>Organizational-oriented applications</i>	16
2 TECHNIQUES	17
2.1 Ajax-based applications	18
<i>jQuery</i>	19
<i>SproutCore</i>	21
<i>Google Web Toolkit</i>	22
2.2 Adobe Flash & Flex	24

<i>Development characteristics</i>	24
<i>Deployment over Flash Player or AIR</i>	25
2.3 Curl	27
2.4 OpenLaszlo	27
<i>The LZX language</i>	28
<i>Proxied & SOLO deployment</i>	29
2.5 JavaFX	31
<i>Development tools</i>	31
<i>Deployment options</i>	31
2.6 Microsoft Silverlight	32
2.7 Design patterns for rich web applications	33
<i>Model-view-controller & SproutCore MVC+SDR</i>	34
<i>Model-view-presentation</i>	35
<i>Model-view-viewmodel</i>	35
<i>3-tier architecture</i>	36
2.8 UI layouts, controls and other components	37
<i>Screen layouts</i>	38
<i>Controls</i>	38
<i>Effects</i>	39
2.9 Features selection & platforms comparison	39
2.10 A short discussion on web applications techniques	42
3 ON STANDARDS...	47
3.1 Why Web standards?	47
3.2 Key Web standards in use today	49
<i>HTML</i>	49
<i>CSS</i>	50
<i>JavaScript</i>	50
3.3 HTML5	51
<i>HTML5 explained</i>	51
<i>HTML5 features</i>	52
<i>Other specs related to HTML5</i>	54
<i>What about Geolocation, SVG, MahML and XHR?</i>	55
3.4 A short discussion on web applications standards	56
END NOTE	61
BIBLIOGRAPHY	63



LIST OF FIGURES

1.1	Rich internet applications in use	3
1.2	Thick vs. thin clients	4
1.3	A screenshot of Google Maps running in a SSB	7
1.4	A tag cloud illustrating topics associated to Web 2.0	13
1.5	An overview of the Bivolino webstore	15
1.6	An dashboard application built with Curl.....	16
2.1	An overview of the Flash integrated platform	17
2.2	The Ajax application model: asynchronous communications	20
2.3	A screenshot of the source code of jQuery.com	21
2.4	A full scale example of a web application using SproutCore	23
2.5	A SWF file being deployed	25
2.6	A SWF file embedded to HTML document.....	26
2.7	The Curl platform.....	28
2.8	A typical workflow in proxied mode with the OpenLaszlo Server	30
2.9	An overview of the JavaFX tools	32
2.10	An overview of the Microsoft Silverlight platform	33
2.11	The Model-View-Controller paradigm	34
2.12	The common 3-tier model	36
2.13	An example of the master/detail screen layout.....	37
2.14	Illustration of some controls	38
2.15	The jQuery UI effect	39
2.16	The market penetration into the browser of Flash, Silverlight and Java	42
3.1	The Web among devices.	48
3.2	The definition of the features among WHATWG and W3C documents	52
3.3	deviantART Muro	57
3.4	HTML5-related features supported by modern browsers	58



INTRODUCTION

WHEN it comes to consumer applications, those can be classified nowadays into two main categories: desktop applications and Web applications. On one hand, desktop applications have to be completely installed on the host device, having the possibility to use Internet for further communications (to update the application or to access email services for example). On the other hand, Web applications run on Web servers, accessible in a specific environment which is commonly the Web browser.

Web applications were at first very primitive: an action from the user implied to refresh the page. Interaction was minimal and responsiveness was limited. The two having for direct consequence a poor user experience compared to native applications pre-installed on the computer.

But as the Web extended its importance to the world, the need for dynamism, usability and responsiveness has rapidly grown. And richer Web applications are the answer to that need. And the term *Rich Internet Application* is the one used by platform vendors to describe the evolution of Web applications. The appellation has been coined in a Macromedia (acquired by Adobe Systems in 2005) White Paper in March 2002 [53], but the concept was not totally new at the time.

Rich internet applications are usually seen as a mashup between desktop applications and traditional web applications. They are as interactive and responsive as desktop apps, while being deployed over the Web. The mashup usually implies to transfer part or all the processing to the client, in contrast to Web apps that are running server-side, the browser taking only care of the rendering part.

Web applications have nowadays become a day-to-day part of our lives. They allow us to perform a large variety of tasks making our lives (on the Web or not) easier: banking transactions, social networking, entertainment and online shopping to name only a few. On one end, their availability to nearly any Web-connected devices make them even more

appealing to the end-user. On the other end, it is a good opportunity for developers to deploy their apps over a unique platform: the Web.

DISCLAIMER

This paper focuses on Web-related technologies. Consequently, it should be noted that technologies around the WWW and Internet are evolving rapidly. All the content present in the thesis is up to date till mid-August 2010, the assigned period to submit the final version of this document.

It can take only a couple of weeks for a technology to improve, and sometimes updates can occur on a daily basis. This is the case for the HTML5 specification for *e.g.*

CONTENT & OBJECTIVES

The purpose of this document is to give an overview of what can be done today in terms of rich web applications, and how it can be done. The following questions are confronted throughout the thesis:

- What is a rich web application? Where can we find them on the Web?
- How rich web clients are built? What is available out there to help developers code RIAs?
- What are the standards related to Web applications? Can HTML5 lead to significant changes in the Web app area?

The web applications sphere is quite large and there is a lot to talk about. This thesis is subdivided in three main chapters, each one of them focusing on a particular subject related to the questions addressed just before.

The first chapter sets up the basics concerning rich internet applications. The objective is to understand what they are and why. Their characteristics are explained and their limitations discussed. The chapter includes also the distinction between a Web application and a client/server application, and between a thin and a thick client. The point is to build a baseline on which the subject can be studied without confusion. Topics such as the Web 2.0, Web services and enterprise-class RIAs are also explored to put Web apps into perspective. And finally the first chapter includes a debate between the use of native apps

over web apps for the iPhone, which is a good opportunity to put the two alternatives face to face in a real-life context.

The second chapter presents different solutions in order to build rich internet applications, and deploy them over the Web. Eight platforms have been selected: Flash, Silverlight, JavaFX, Curl, OpenLaszlo, jQuery, Google Web Toolkit and SproutCore. Every single solution has its own extent, and they do not exclusively compete with each other. They can indeed have different objectives. If Flash is competing directly with Silverlight, it is not especially the case with jQuery for *e.g.* The purpose of this chapter is to analyze shortly every selected platform, and then compare them over a large range of features. It includes also a discussion on how useful pre-built widgets and components can be.

And last but not least, the third chapter focuses on standards. The Web has always been seen as an open platform for all. This was a key for its success and its expansion. And the W3C was created in October 1994 to help maintain a cohesion between industry players over Web technologies. Standards can play an important role on many different levels which are discussed in this chapter. With the rise of Web applications, a need for new or revised standards has grown up rapidly. That is why specifications such as HTML5 have a strong emphasis on dynamic websites, interactivity and rich media content. That is why this chapter discusses traditional standards and Web app related standards. It also explains how these can have a positive effect on the Web.

* * *

UNDERSTANDING RIAs

THIS chapter aims to explain what is a rich internet application (RIA). Their characteristics are clarified and the reasons why they are becoming so popular are explained. Their limitations and their implications within the Web 2.0 era are also discussed. This chapter is important in order to understand why these richer web applications are useful and when they can become inadequate.

1.1 DEFINITION AND CRITERIA

The World Wide Web is one of the most accessed services that Internet can offer to its users. The system of inter-linked hypertext documents designed by Tim BERNERS-LEE¹ lets one view webpages containing text, images and other content through a simple web browser. To do so, the HTML markup language is used to define the structure of the document. The document can be found with an URL, and retrieved from a remote server with the help of the HTTP protocol. However, except the content itself and the fact that an end-user could jump from one webpage to another one through hyperlinks, little or no interactivity was available. The WWW was only built around linked static pages.

However, the WWW was full of promises. Rapidly from static, the web page went dynamic. The web page could vary according different contexts and conditions, the DOM, client-side scripting (JavaScript being the most popular language) and server-side scripting (PHP, Perl and others) helping to do so. This is where the web application came from. A traditional web application is hosted on a remote server and works according the

¹ He was also helped in his work by Robert CAILLIAU, a Belgian working at CERN where the two have met to work on the project.

request-wait-response model: the client makes an HTTP request to the server, waits for the server to process the request (accessing databases, backend processing, *etc.*), and then receives a formatted page into the browser as a response. Almost none of the processing is done client-side.

However, static pages and traditional web applications were not enough anymore. A transition to a new model was necessary and logical. That is the reason why the concept of a “rich web application” was born. RIAs are the direct improvement of traditional web applications. They are the answer to a growing need of interactivity and responsiveness concerning websites and web applications. The main objective being to improve user experience.

But rich internet applications do not aim to kill HTML. The markup language can still be widely used for simple interfaces (static pages and low-dynamic websites), rendering text, pictures and simple data. No need to use complex methods to write simple websites.

There is no specific definition for the word “Rich Internet Application” to this date. However, we can narrow a meaning by the features rich web applications have in common. But let’s start by analyzing the components of the “RIA” word, used by vendors to describe their technologies regarding the future of web applications:

- *Rich*: what does “richness” mean? The richness of an application can be defined by the interaction the application has with its user. A rich interaction model is one that can offer a wide range of input methods, high interactivity with what’s on the screen, and responsiveness [118]. Traditional desktop applications are considered to be rich.

So it means that rich web applications are more dynamic, interactive and responsive. And that they can deal with a variety of content such as raw data, text, pictures, audio and video.

- *Internet*: even if RIAs have now offline capabilities, a requirement of a RIA is to be exclusively deployed over the Internet (through the WWW).
- *Application*: Richer web applications - like any other piece of software - are designed and built with the purpose to help the user fulfill a task.

RIAs are applications aiming to be as *rich* as desktop applications, while being accessed with an Internet connection. It means they try to function like desktop applications, allowing high interactions such as drag-and-drop, animations, efficient data manipulation, real time events (email structure or password length verification to describe only a couple) and so on... Also, it can be hard to distinguish a web application (richer or not) from the webpage as they can use the same technologies.

To give a first glimpse to the reader, Fig. 2.14 illustrates 2 ways where RIAs can be proved useful. Both examples are using the Ajax technique (more on this in Section 2.1, on page 18). The whole webpage (or website for that matter) is not an application like an opened Google Docs editor, but the page includes richer components, allowing the user to do specific tasks without page refreshes. Another example, for a business in that case, can be seen Fig. 1.5 on page 15.

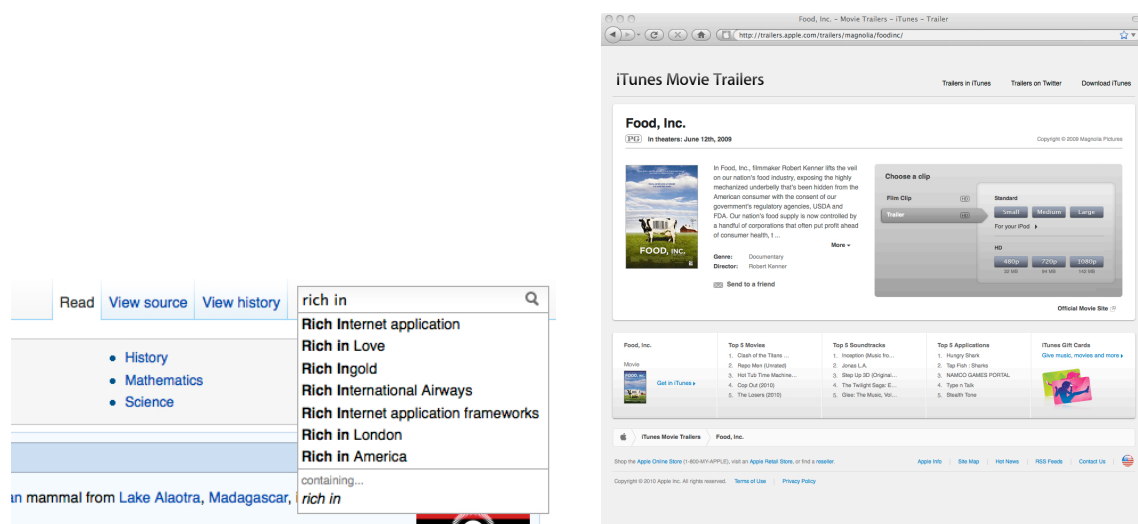


Figure 1.1 - RIAs in use. On the left is the real-time search application of Wikipedia, a common component now included easily in many websites [48]. And on the right is the Apple Trailers webpage featuring the film Food, Inc. [13].

The Wikipedia search engine gives the user search recommendations according to the letters he already has typed into the input field. And the Apple Trailers webpage for the documentary Food, Inc. lets the user do various tasks such as watching different trailers and having more information (by expanding the text to have a full synopsis or expand the poster into an intra-page pop-up window) about the movie without being bothered by multiple refreshes.

Thick vs. thin applications

Compared to simple webpages and web applications where the processing is done server-side, RIAs tend to execute a significant part of the processing client-side. That is why RIAs are often considered as “thicker” clients. Different architectures can be considered according to the amount of processing taking care of by the client and the server.

Every architecture has its advantages and limitations, and every architecture can be chosen according to the developer’s and the end-user’s needs. For example, the Apple iWork.com

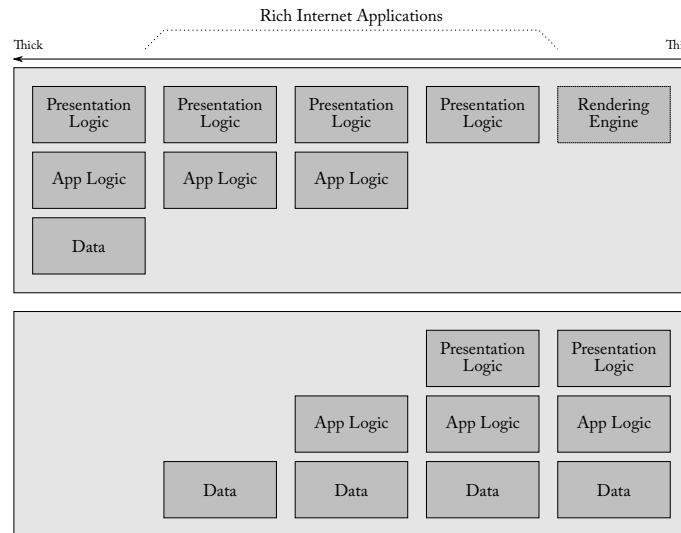


Figure 1.2 - Thick vs. thin clients. On the far left are desktop applications. The far right represents web applications where nothing except the rendering is done client-side.

online services (built with SproutCore) adopts a strategy where the business logic is transferred to the client, improving interaction and feedback, while keeping data remotely. The application and the data can be accessed with any web-enabled device as the web application model permits it, but the application has first to be downloaded into the browser, which can take some time. The app can however be cached locally, and downloaded again only if changes appear when the application is updated for *e.g.*

It should be noted that rich internet applications do not aim to kill HTML and the thin model. It is still a remarkable markup language that can be used to render simple elements such text, pictures and simple data. If a simple user interface is only needed to fill the need of the website or the web application, HTML still stands as a viable, simple and fast solution to do so.

Client/server applications vs. web applications

It's important to make the difference between client/server applications and web applications. Client/server applications are applications following a model where a client can be executed on the desktop, while having the ability to communicate with a remote server for data and other services. It means that the most part of the processing (presentation and business logic) is done on the client computer. The application is installed on the computer, and the data is generally stored locally.

Typical client applications following this architecture are web browsers (IE, Firefox and others) and email clients (Outlook, Thunderbird and others). The applications are written for a specific platform (although a cross-platform language can be used to get rid of that constrain) and have to be installed on the client device. It means also that the maintenance of those applications can be tricky as updates have to be installed on the variety of clients where the application has been implemented.

Web applications are available over the WWW, running commonly into the Web browser.

RIA features and characteristics

Web deployment: web deployment is a core characteristic of web applications, and consequently of rich internet applications. No proper installation and configuration on the client's system device is required. The administration of updates is made easy as the application can be updated directly on the server. When the end-user will request the application in its browser, he will interact immediately with the updated version. Moreover, web deployment made the application available for a variety of web-enabled devices. It helps to make abstraction of operating systems differences. The WWW is a fast, wide, and cost-effective way to deploy applications.

Browser executing environment: web applications are typically executed within the web browser. It means RIAs developers have to keep an eye on the variety of browser specifications. However, RIAs build around Flash and Java requires a plugin, the latter taking care of the browsers differences, waving that responsibility from the application itself. Nevertheless, solutions such as Adobe AIR are now available to execute rich internet applications outside the browsers.

Continuous application usability: this feature is the main improvement of RIAs compared to traditional web applications. There is no page refreshes, no waiting while wondering what's the server doing. When communications with the server are needed, they are made independently of the communications with the user. It makes the application more responsive and helps interactivity as virtually no interruption is happening. It also means behaviors will have to evolve as with RIAs we shift away from a well anchored "page model" (*i.e.* clicking to access a new page). The back and forward buttons well-known in browsers may be useless in RIAs.

Complex graphical user interfaces: complex interfaces are needed to improve the web experience of users. It is by analyzing this feature that we can see a convergence between desktop applications and web applications. The better an end-user's web experience is, the more he will stay connected to the application. The chances that he will be back and that he will communicate about the application with others are similarly improved. A complex GUI lets the user use the application without being annoyed by navigation issues for simple

tasks. Furthermore, data exploration and data interaction are much improved with richer user interfaces.

Client-side processing: RIAs tend to be “thick clients”, meaning that some or a more significant part of the processing will be done on the client, and not on the server. It helps freeing the servers of numerous requests and reduce the processing power needed to complete them.

Deployment alternatives

There are different ways to execute a rich web application on the client device. The common way to run RIAs is with the Web browser. If the application is built exclusively around Web standards, it can run directly into the browser. The other typical browser alternatives are *plugins*. RIA platforms provide a browser plugin to allow the execution of their apps within the browser, but through their own runtime environment. This is the case for the Flash or the Silverlight platform. And if the plugin is not installed on the client, the application can not run within the browser.

Aside of the browser, a few solutions provide a way to run web applications outside the browser. The apps can run as standalone applications, directly from the desktop, freed of the browser. A runtime environment such as Adobe AIR or Curl RTE needs to be installed on the client to launch properly the apps. Cross-platform desktop RTEs are useful to deploy applications on a range of different devices with minimal or no code rewriting.

A standalone web application is also referred to a “Site Specific Browser” (SSB). SSBs can be considered as lightweight browsers designed only for a specific website or a web app. Fig. 1.3 shows a SSB for Google Maps in action (the rendering is handled through WebKit). It was created with Fluid, an app available only for Mac OS X. However, Mozilla Prism can provide a similar feature on Windows. This technique allows the user to run his favorite web apps (Facebook, Flickr, Gmail, *etc.*) without having to be bothered by multiple tabs in the browser.

1.2 WHY RIAs ARE EMERGING?

We can witness a real emergence of rich internet applications the past few years, and it is only the beginning. A few tendencies have helped RIAs to become what they are.

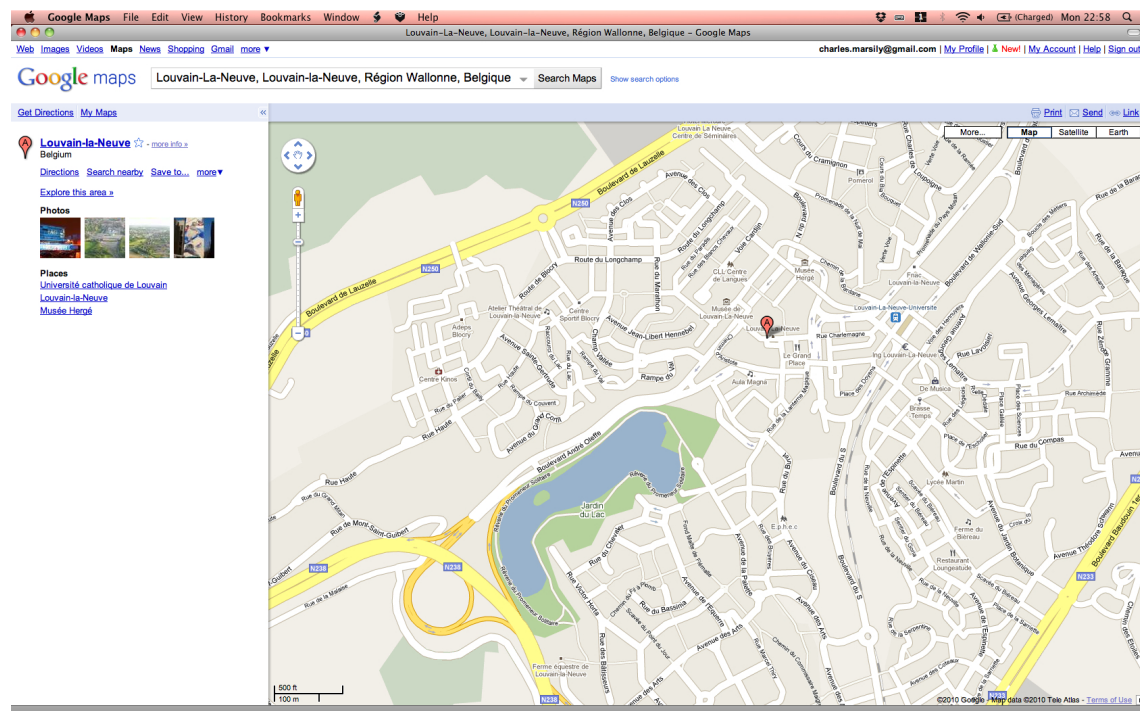


Figure 1.3 – A screenshot of Google Maps running in a SSB created by Fluid (OS X only). Google Maps is available at [76].

First, Internet and the World Wide Web have become important parts of our lives. And the Web 2.0 isn't doing anything to change that fact. We communicate, participate, share, trade and inform ourselves on the WWW. RIAs are one way they make the time we spend on Internet more worthy and valuable. Complex web applications are improving the way we interact with information and the content available on the Web. Broadband connections are also spreading rapidly all over the world, helping more bandwidth-consuming web applications to be deployed.

Secondly, people have more and more mobile devices connected to Internet. We have today laptops, tablets, smartphones and other web-enabled devices to complete or replace desktop computers. Writing complex applications for each one of these devices can be tricky, time and cost consuming. RIAs can help resolve that issue. Moreover, with users going mobile, they may need the same application on different devices. A situation that can be easily handled as web applications are one click in the browser away. RIAs are a good way to deploy cross-devices applications.

Furthermore, the rise of personal computing helped the digitalization of our communications and information. People are using more and more computers and mobile devices to assist them in their professional and personal lives. Why should we have stickies if we can share our to-do list between the computer (accessible everywhere through the browser),

the smartphone, and even colleagues? That is where a service like Remember The Milk can be very useful [34].

Finally, mainstream tech companies have done their move: Adobe is in the game for a long time now with its Flash platform and Flex framework, Microsoft has Silverlight, Sun Microsystems (and consequently Oracle) has JavaFX, IBM is putting forward its WebSphere platform, Apple uses SproutCore, Google promotes its Google Web Toolkit, and Mozilla RIA's platform resides around Firefox. Solutions supported by enterprise-centered IT companies (such Microsoft, IBM and Oracle) is crucial for the technology's adoption in the corporate world [106].

1.3 LIMITATIONS

If RIA architectures have lots of advantages, the model has nonetheless numerous limitations:

Network dependance: their strength can become their weakness. Web applications and RIAs are dependent of an Internet connection. Moreover, broadband connection are more and more required to execute rich internet applications quickly and easily. However, some rich internet applications are becoming offline capable. It was the case recently for Google Gmail.

Richness capability: if RIAs tend to be richer and richer after years, they cannot be compared properly with the richness of traditional desktop applications. Microsoft Outlook stays to this day more powerful in terms of interaction, integration and responsiveness than a rich web-based application such as Gmail. Building richness itself is however only limited to developers' imagination and the platform used. It's consequently highly probable that further richer web applications will be written in the future, becoming as good as desktop applications.

Complexity: enhanced experience for the end-user comes to a price. Compared to simple HTML coding, RIAs are much more difficult to design, write, test, debug and support. It's a cost that enterprises and developers have to take into account in their decision process.

Sandbox: rich internet applications run traditionally in sandboxes. If they are secure environment, they are nonetheless closed and with very limited access to system resources.

Scripting performance: a client-side scripting language such as JavaScript is nearly compulsory to run rich internet applications. It means it has to be enabled by web browsers for the application to work properly. Scripting performances are consequently important: the better the JavaScript engine is, the faster the script will be executed. We can see nowadays

that any major web browser vendor is trying to improve their JavaScript engine. It should be noted that JavaScript is an interpreted language, resulting in lower performances compared to compiled languages seen in desktop applications and Java applications.

Loading time: by moving part of the processing client-side, scripts have to be downloaded and cached on the client. This process can sometime take time. The issue can be minimized by compression, cache and optimization.

Search Engine Optimization (SEO): simple HTML documents are easily handled by search engines such as the ones of Google, Yahoo and Microsoft. Even if workarounds exist, search engines have difficulties to index the content of richer web applications.

Monitoring and measuring: the asynchronous characteristic of rich internet applications make performances monitoring and measuring more difficult.

Lack of tools: tools are needed to assist developers in their coding. This point is important as a good development platform attracts developers to use it, which in return can broaden the volume of applications built on the platform, which can help in its deployment on a large variety of clients. The tools available to build RIAs are relatively light and simple compared to the ones used to write desktop applications [125].

1.4 WEB APPLICATIONS VS. NATIVE APPS: WHAT TO CHOOSE FOR THE iPhone?

This section addresses the problematic of native applications against web applications when they are used on the Apple iPhone. This can be an interesting case study to see how Web apps can be useful or become restricted in some situation. The iPhone being a handheld device, it is also the opportunity to review the use of Web applications within the mobile device area, a market seen as the next battle ground for consumer devices.

On one side, iPhone apps run directly on iOS (Apple's operating system for the iPhone, iPod Touch and the iPad). They have access to the iPhone's content and hardware (such as the accelerometers, used in games for *e.g.*). They are developed for iOS only, meaning they have to be entirely rewritten if the app want to be executed on another platform (such as Palm WebOS for *e.g.*). Native applications are usually more suited for complex and generally offline applications such as games.

On the other side, web applications run in the mobile version of Safari, the in-house browser of the iPhone. Opera Mini, whose release has been approved by Apple, can be used as an alternative tough. They are developed with the Ajax technique, around standards such as HTML, CSS and JavaScript. They have very limited access to data and hardware.

Their advantage resides in their inter-operability: they are accessible on nearly any other device connected to the Web with minimal code adjustments [88].

However, technologies like Flash and Java are not supported on iOS, which means web applications cannot have the benefits of these platforms. Websites and web apps using these technologies are not rendered (if not totally) properly. Apple's CEO Steve Jobs addressed criticisms over Flash arguing that the technology is not open, unreliable, not secure, not optimized for mobile uses, and that overall performances are discussable [90]. Apple pushes HTML5 as a replacement for Flash. This can be problematic for a lot of web authors as their websites can heavily rely on Flash (the animated advertisements for *e.g.*).

Web apps were the first and only way for third-party applications to run on the iPhone [51], with Apple arguing that Web-based applications could have the look-and-feel of native applications. These applications could have accessed iPhone services (such as calls) “without compromising its reliability or security” [51]. However, Apple went backwards and the first iPhone SDK to help building native applications was released in March 2008.

If the native option is chosen, developers have to go through a validation process. Native apps can only be deployed on iPhone's through the App Store, Apple's platform to sell iOS applications to end-users. If the content of the application is deemed inappropriate, the app is simply rejected, the developer having built it for nothing.

Furthermore, Apple retains 30% of the price tag if the application is sold, Apple explaining that its share will cover the costs of their App Store platform. The bright side of this app platform is its uniqueness: developers do not have to set up their own place to sell their apps, Apple taking care of that part for them. Also, developers can insert advertisements through the iAd mobile platform. A way to integrate ads within the app which are released in the App Store, Apple retaining again 40% of the revenues generated by its platform.

Even if Apple retains 30% of the app's price, the App Store is a new way for developers to make profit from their apps. Web applications are not suited to make money up front, when the application is sold. However, native apps on the iPhone, often offering the same services than websites and web apps, can be sold for a couple of bucks directly to the user. Scalability helping, developers can make a significant profit from their work.

Web applications are not subjected to these restrictions. Inappropriate content is available through web apps, and no revenue is taken from paid applications. Web applications can also be updated more quickly. And this might be an even more important advantage in the iPhone case. The validation process can take some time before the app can be available on the Store. And any update for a native app is subjected to the validation procedure. It means developers can not update rapidly their apps, potentially harming user experience and their revenue if they are expecting to sell them.

One can argue that the decision between native applications and web applications for the iPhone is not black & white. Both worlds have strengths and weaknesses. But it seems

that the fascination of the iPhone and its App Store have generated a huge interest for native applications. It is understandable that games and other power-consuming apps are developed as native applications.

But on the iPhone, even news websites, with an underlying content much simple, are developing native applications for the iPhone. Moreover, some argue that native iPhone apps are still much more responsive than web apps. They think they are not able to be as good as native ones and that web apps can not provide a better user experience when an native alternative is available for the iPhone [81, 83].

Maybe this trend is generated by the appeal the iPhone enjoys today, the smartphone being seen as a must-have piece of technology to possess today. The App Store, heavily integrated to the iTunes integrated model, seems to work for either developers, users or Apple. Other tech companies also deployed their very own applications store, but their extent is much more limited compared to Apple's.

1.5 RIAs AS A PART OF A BROADER WEB 2.0 ERA

The term Web 2.0 can be misleading as it does *not* define a new version of the World Wide Web (technology speaking). Like RIAs, no succinct definition exists, and trying to define it can be tricky. After consideration, it can be considered as an overused buzzword that even Tim BERNERS-LEE consider as “a piece of jargon” [97]. However, the notion of Web 2.0 does exist for a reason. And as a matter of fact something has happened between the so-called “Web 1.0” and the Web as we know it today. We will attempt to describe that shift within this section, and take the opportunity to situate where rich internet applications stand in this Web 2.0 era.

The Web 2.0 can be seen as a new way the Web is used by the people and developers alike, compared to the Web as it was during its first years of existence. To put it in simple words, the Web 1.0 was mostly a medium used by the people to access information, and the Web 2.0 is seen as a participative platform filled with user generated content. The *read* Web became the *read & write* Web, converging to the idea Tim BERNERS-LEE had in the first place: a Web as a unique and global medium for collaboration. An idea that has been forgotten in the first place.

The Web 1.0 was mostly about few people creating content for a lot of end-users. The Web was used mainly as source of information, on which they didn't own the material, acting as simple readers. The first ages of the Web built only a few places for participation

and interaction². Now the Web is all about participation and creating content. The Web 2.0 lets end-users produce and manage their data, and websites are intermediaries helping them to do so. On one side, the Web 1.0 is characterized by few people publishing for a lot of people. And on the other side the Web 2.0 features many publishing for many, with creators and end-users being able to switch their respective role.

Web 2.0 services is all about facilitating content sharing: Facebook takes care of your personal information, YouTube of your videos, Flickr of your pictures and Delicious of your bookmarks for example. YouTube and Flickr does not offer videos or pictures, but a place to share the creations with others, without managing the content itself. The more the user base is large, the more the medium will be successful. Web 2.0 companies can not survive without their user base and what they create.

The Web as it is now is people-centric, a trend so massive that the Time magazine chose as the “People of the Year” in 2006 all the contributors having uploaded individual content on Web 2.0 websites. The magazine put the word “You” on its December 25, 2006 cover. Another confirmation that now every single person can select, rate, comment, publish, control and more importantly create their content. People participate and interact with what’s happening online. They need to share. They need to express themselves.

A core component of Web 2.0 is *collective intelligence*. Collective intelligence refers to the cumulative value users can bring to content within the participative Web. Ratings and comments are for *e.g.* primitive ways to build collective intelligence [151]. But we have also evolved to folksonomies. “Folksonomy is the result of personal free tagging of information and objects (anything with a URL) for one’s own retrieval. The tagging is done in a social environment (usually shared and open to others). Folksonomy is created from the act of tagging by the person consuming the information” [146]. And tagging is a major trend characterizing the Web 2.0. We tag people on Facebook, places on Flickr and URLs on Delicious, to cite only a few...

The notion of *Long Tail* is also often associated with Web 2.0. It implies that the truly useful content is situated on the many smaller websites of the Web, referred as the Long Tail [54], and not on the few major websites of the WWW (the big head). It is consequently important to value the long tail, and not only the few big and popular areas of the Web.

A good way to have an overall picture of the Web 2.0 phenomenon is to look at Fig. 1.4. The figure illustrates popular subjects that are associated to the Web 2.0, some of them being highly correlated with rich internet applications. It is the case for usability, design, Ajax and CSS for example. But as mentioned before, the Web 2.0 is not an improvement

² It is worth noting that an alternative such as Usenet existed and still exists with limited popularity today. Usenet was however created 10 years before the WWW, being a service running on the Internet as much as the Web. Now one can access Usenet discussions on the Web with Google Groups



Figure 1.4 - A tag cloud illustrating topics associated to Web 2.0. From Wikipedia [44].

of the technology itself. Even if the Web 2.0 tends to produce enhanced applications, they are still based on current practices and implementations. The new wave of Web 2.0 applications is just the answer to the users' need of tools to create and control their content. They are supposed to help making the best of the Web 2.0 platform [113]. And that is exactly where RIAs can play a prime role. After all, would people still use Flickr if they needed to take 2 hours of their time to upload a couple of pictures on their online album? Probably not... That is why RIAs and their underlying technologies such as Ajax, JavaScript, Flash or Silverlight can ease the process of content generation and manipulation.

Rich user interfaces brought a new class of web applications available on the WWW. Like their desktop counterparts, we can now access within a browser an office production suite such as Google Docs, or the online version of Photoshop. Instead of competing on desktop platforms, now applications are also challenged on the Web. And this can only be good for competition.

But the Web 2.0 is not without weaknesses. The Web 2.0 trend has brought massive user-generated content on a single platform. We assist now to a sort of information overload: finding valuable knowledge will become more and more difficult in the future. Finding quality material can also be tricky as now a lot of Web publishers are amateurs. They are usually not remunerated from their online content [82].

Trust can also be problematic with the Web as we use it today. Can we trust the knowledge found on Wikipedia? Can we rely on banking applications to manage our banking accounts? Can we be in confidence with people we've met on Facebook without having seen them in our real lives? It seems that is the case for now.

1.6 RIAs FOR ENTERPRISES

The Web becoming a unique platform accessed by nearly everybody, it was only a matter of time before companies used it to make business and to help them in their development. Two kind of rich internet applications can be used by organizations: customer-oriented applications (both for B2B and B2C) and decision-making support applications (RIAs can be ideal for Business Intelligence services for *e.g.*).

If enterprise-class RIAs can bring a lot of good, they do have to confront challenges valued by organizations [87]:

- Costs. RIA development costs can significantly increase according to the complexity of the application.
- Security, more than never.
- Consistency. Companies cannot afford to change their systems at every technology shift. That is why Flash is so popular as the technology is well anchored compared to burgeoning competitors such as SproutCore.
- Accessibility. This even more important for customer-oriented applications.

Customer-oriented applications

The Web has become an important market for businesses. Some of them are purely businesses operating on the Web (Amazon or eBay), while other are using the platform as another medium to sell their products and services (Apple). The Web can not be ignored nowadays. It is now used to provide a middle ground to promote, sell and support all kind of products and services.

RIAs being cross-devices, ubiquity can be achieved without significant issues, while frequent updates can easily improve the application as no installation is required. If the Web is used wisely, it can lead to benefits for the businesses. And RIAs are one way to use the Web effectively. Here is an example taken from an IBM white paper, showing how buyers can be guided during their shopping process [87]:

“For example, you might create an RIA for ordering products from your Web site and connect the RIA in realtime to your back-office system via a simple Web services interface. Then, if a user enters a quantity of 10 for an item, you could immediately display a message such as, “Next price break at 15,” to entice the user to increase the size of the order to get a better.”

If a commerce-oriented RIA is written properly, it might lead to a customer engaged and focused in the process. It means he might stay longer connected into the RIA, explore it further, come back later and talk about it to its entourage. Visibility is enhanced, satisfaction and loyalty are improved, error and customer support costs are reduced, and sales are increased.



Figure 1.5 - An overview of the Bivolino self-service webstore. The web application lets the client creating (and then buying) tailor-made shirts flawlessly following a series of steps [62].

Being more visual and by making abstraction of the request-wait-response model, RIAs reduce the overall complexity of the process and enrich the user experience within the

application. They allow features such as undo, redo and real-time events, all within the same Web page and without refreshes.

Fig. 1.5 illustrates the general idea of what RIAs can do for businesses. The website is page-based at first, but a rich web-based application is used to customize the shirt itself. As soon as the buyer has chosen the shirt characteristics and features, he will be redirected to a new page to take care of the financial details.

Organizational-oriented applications

As employees have to decide faster and better, data, modeling tools and indicators (visual or not) are becoming more and more pivotal for decision making. Here again RIAs, being available on extranets for *e.g.*, can be helpful for both employees and the organization.

On one hand, employees have access to performance indicators, real-time data and other visualizations to help them quickly in their analyses. On the other hand, organizations have an effective medium to provide the data, possibly coming from different sources, while performances and accuracy are up, and waiting time is down. RIAs can be used to create portals, dashboards, rich forms, and also as an information platform.

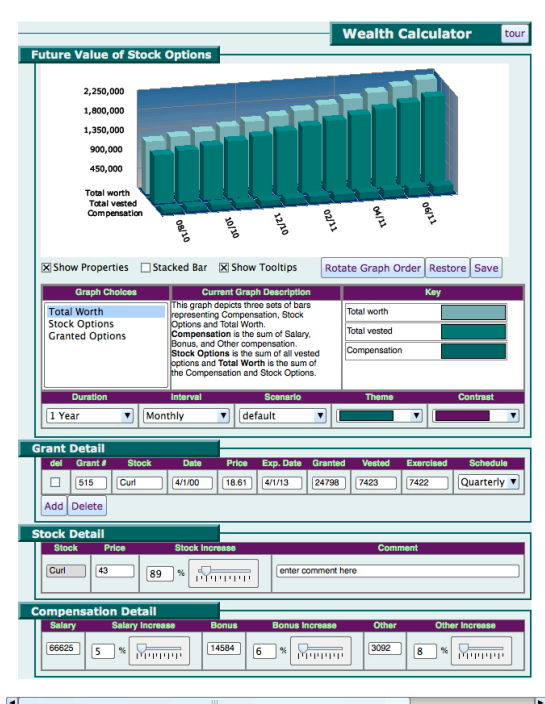


Figure 1.6 - An dashboard application built with Curl, representing information around a wealth calculator [64].

2

TECHNIQUES

Lots of RIA frameworks comparison articles can be found on the Web and in the literature. But here the approach is slightly different: instead of providing a technical comparison of RIA frameworks alone, RIA platforms are going to be analyzed and compared as a whole, with the framework being only an element of them. Choosing a specific solution can be tricky as one has in front of him a panel of integrated solutions to help him in his development and deployment. The easiest example of such platform would be Adobe's Flash - including the Flex framework - illustrated in Fig. 2.1:

Adobe Flash Platform




Professional Tools	Frameworks	Runtimes	Servers
 Flash Builder	Flex framework	Flash Player	Flash Media Server
 Flash Catalyst		AIR	LiveCycle Data Services
 Flash Professional			

Figure 2.1 - An overview of the Flash integrated platform. Flex is the platform's framework and the main component to build RLAs [69].

The Flash solution contains designer tools, development tools, deployment tools and other services useful to make powerful rich internet applications.

The choice of the platforms presented in this paper has been tricky as new tools and technologies are developed year after year. To keep this document within an acceptable length, 8 platforms have been picked according either their popularity among developers

(businesses and individuals alike) or their use and support by first-line tech companies. The selected solutions are listed hereafter:

- JavaScript/Ajax-based solutions including jQuery, Google Web Toolkit and SproutCore.
- Adobe Flash, pioneer in the matter.
- Microsoft Silverlight, seen as Microsoft' answer to Flash.
- JavaFX, launched by Sun Microsystems to compete with Flash and leverage its Java technology.
- OpenLaszlo, enterprise-centered.
- Curl, enterprise-centered.

The comparison will be made according several technical and non-technical factors. It will help the reader to differentiate prevalent RIA platforms to this date on many different levels. Every single solution has its advantages and its limitations, and the purpose of this chapter of the thesis is to emphasize the strengths and weaknesses of the elected platforms. Every factors used is going to be described later, and the reason why they have been chosen will be explained.

2.1 AJAX-BASED APPLICATIONS

Ajax is shorthand for “Asynchronous JavaScript and XML”. The notion *asynchronous* is important as it defines the part of the Ajax paradigm that lets the user interact with the interface without being interrupted by page refreshes. Asynchronous communication is a key point for richer internet application. The term was coined by Jesse James GARRETT in a article explaining “a new approach to web applications”. Ajax is not a brand, or a commercial product, it just describes an approach to build web applications involving several technologies to make it work:

- The XMLHttpRequest (XHR) object to allow asynchronous data retrieval with the server. An Ajax application can send HTTP requests directly to the server, retrieve data, and have the response directly injected back into the script. This can be done with XML, JSON or plain text among others. The data retrieved can alter the DOM of an active document, inducing changes for the user without having to reload the webpage.

- XML and XSLT for the interchange of data and its manipulation, respectively. Both XML and XSLT are not specifically required. Alternative such as JSON or pre-formatted plain text can be used for data interchange.
- A DOM, for dynamic presentation and interaction, and standard presentation languages such as (X)HTML and CSS for basic rendering.
- JavaScript to make all the above work together. JavaScript is the most popular client-side scripting language to execute Ajax-based applications as it is included within popular browsers. However, JavaScript is not the only scripting language that can be used to make it work.

Using properly all these technologies can improve the user experience with interactive and responsive web applications. To make this work, an Ajax engine is loaded to the client device. Written usually in JavaScript, the engine will coordinate the communication with the server and coordinate the rendering of the interface. It is the role of the engine to make the communications with the end-user independent to the communications with the server, as seen in Fig. 2.2. As Ajax-based applications cannot handle rich media such as audio and video, the engine will have to use a Flash or a QuickTime plugin to take care of that specific content.

jQuery

Many different tools - frameworks, toolkits or libraries - are able to help developers write Ajax-based applications. jQuery is one of them. jQuery is a JavaScript library that has become popular among web developers as it stands as a solid library to build more and more complex, powerful and dynamic webpages. It offers effects, widgets, interactions, plugins and range of Ajax functions and methods to allow new information to be used without having to reload the web page. If fancy controls do not make a rich web applications, asynchronous communications can help to do so. Hence it can be considered as a tool to develop RIAs.

There are basically three ways to build JavaScript code, and jQuery proposes two of them:

- *Raw code*. Everything is written from scratch. If this way is the most flexible of all, it is also the one that can lead to lots of bugs.
- *Libraries*. A library such as Prototype, Mojo or jQuery provides a syntax for pre-written code. Flexibility is possible as the developer still has to do some writing. Libraries can help making less buggy apps and websites.
- *Widgets*. They are pre-made applications. No specific writing is necessary but flexibility is nearly absent (skinning may be allowed for *e.g.*). Widgets

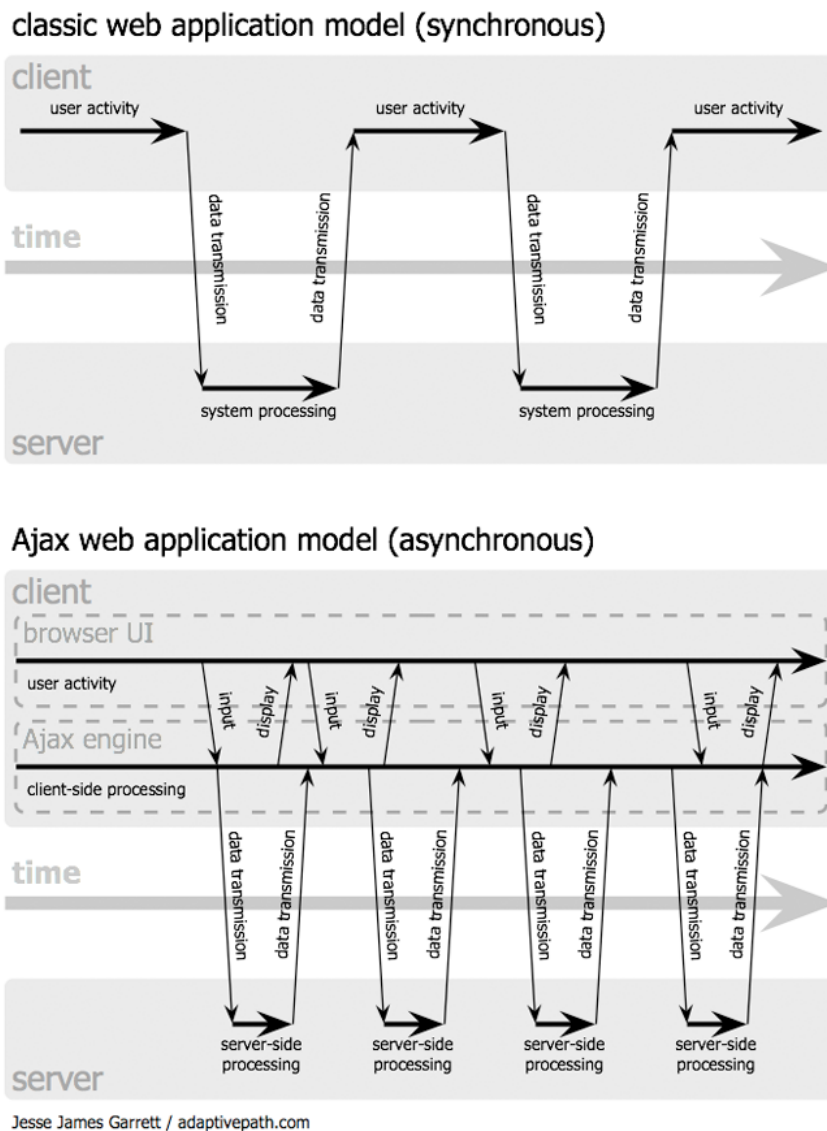


Figure 2.2 - *The Ajax application model: asynchronous communications* [74].

are tested for a variety of Web browsers. If a widget is compatible for a specific one, no bugs should arise.

jQuery aims to facilitate client-side scripting by simplifying the interaction between JavaScript and HTML. Its focus is to “find elements” and then “perform actions with them”. The library is usually contained into a unique JavaScript file, providing its common core features: effects, DOM, Ajax and events. Other features can be added *via* a plugin. jQuery UI, built on top of the jQuery library, provides high-level prebuilt widgets and low-level interactions and effects. These prebuilt components let developers take care of something more complex and time-consuming than simply reinventing the wheel in many ways.

jQuery might seem a bit *light* as a solution to develop richer web applications, but its objective is more to ease and make the coding process faster than properly propose an integrated platform. The point is to code efficiently by using a particular syntax. For *e.g.* jQuery includes a variety of functions (based on XHR) to make asynchronous requests much easier than using the XHR object itself. Also jQuery UI provides prebuilt skinnable widgets (date picker), interaction (drag and drop) and effects (transitions).

To use the jQuery library, one copy has to be called within the HTML code with the `scr` attribute within the script element. Only then one can use the functions of the library to code the JavaScript part of his webpage or the web application, as shown in Fig 2.3. To call a widget for example, the developer will use the `id` attribute in his code.

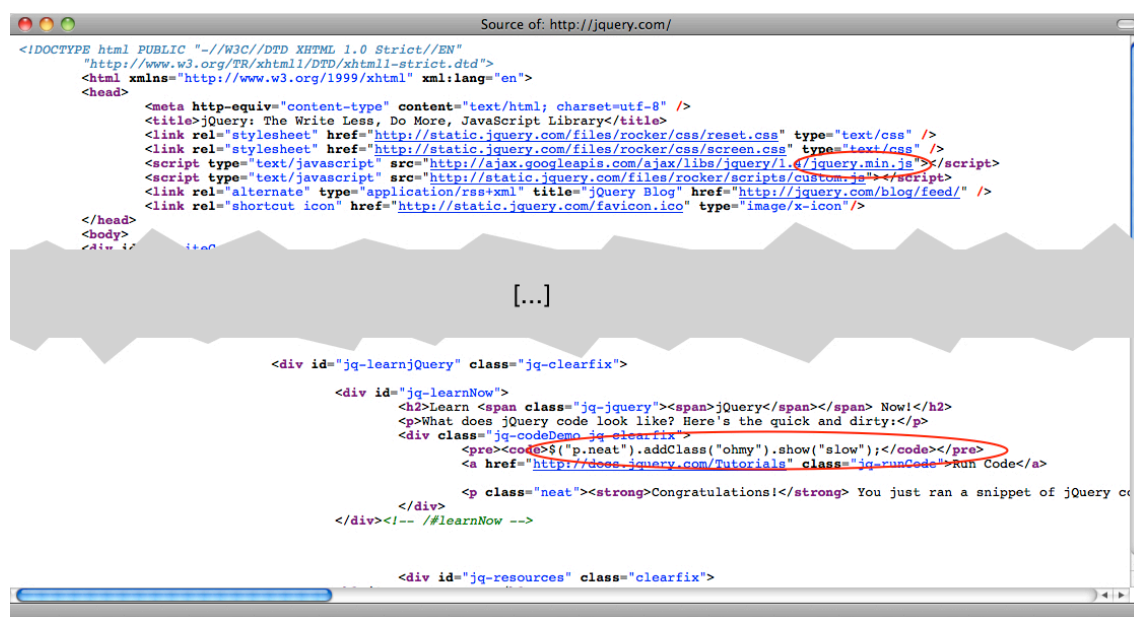


Figure 2.3 - A screenshot of the source code of jquery.com [91].

SproutCore

SproutCore is an open-source platform to help create so-called cloud applications. The platform's architecture is built around HTML5 and JavaScript, and aims to build cloud applications that are scalable, using complex features, supporting keyboard and touch events, rich animations, having offline capabilities, while being as responsive as desktop applications. HTML5 and JavaScript make SproutCore applications able to be run on modern web browsers without plugins. Although those can be used to add local storage capabilities, video support and uploading support.

The business logic of SproutCore applications is downloaded to the browser, while the data stays in the cloud. Doing so, SproutCore applications fell into the thick client category. The model can speed up interactions with the application as the processing is done on the client, without giving up on scalability as the application will be accessible to anyone using a modern browser.

SproutCore does not aim to enrich document-oriented web pages by interfering with HTML. SproutCore's objective is to build full web-capable applications, and does not want to compete with JavaScript libraries such as jQuery.

Built SproutCore apps have an `index.html` file and some static resources. Static resources can be cached for better performance, and GZIP compression can be enabled to load the application faster. The SproutCore JavaScript framework alone weights about 500Kb, but can be compressed to about 100Kb [10].

SproutCore provides several frameworks to build full-scale cloud applications [7, 9, 10, 14, 19, 36, 41, 42]:

- The Runtime framework, providing the tools to write the core of the application. It is used for *e.g.* to set up classes, to work with arrays, loops or to run benchmarks.
- The DataStore framework. It handles structured data in SproutCore applications. The component can communicate with the server for data for *e.g.*
- The Foundation framework, that contains the basics (event handling, Ajax requests, and more) to make the application run properly in the browser.
- Views, which handles the application's visuals. SproutCore comes with pre-built views (such as controls) that can be used directly within the applications.
- The UnitTesting tool.
- The BuildTools to produce optimized JavaScript, HTML and CSS files from the source code. The tools help developers to keep a well structured project, easy to maintain, and cache management.

Google Web Toolkit

Google Web Toolkit (GWT) is the platform supported by Google to build Ajax-based web applications. It has the characteristics to produce JavaScript code in the Java language. As the web application is written in Java, the developer is free to use any Java development tool of his choice. Google provides also a plugin for Eclipse IDE. GWT mission statement



Figure 2.4 - A full scale example of a web application using SproutCore. This is the Bong.TV application, which mainly provides recording services for the end-user. The app provides dynamic features such as tab browsing and popup window (inside the app) for video details [63].

is “[...] to radically improve the web experience for users by enabling developers to use existing Java tools to build no-compromise AJAX for any modern browser” [80].

A GWT application can be embedded into an HTML document with the script tag. That webpage can be seen as the host page of the web application, which not necessarily includes any strict HTML content (it means that the GWT application could use the whole webpage for its UI for example). However, GWT is also built to improve part of webpages by including specific widgets to the wanted areas of the HTML document. Attaching a widget can be done *via* the id attribute or by using the `RootPanel.get()` method. The host page doesn't have to be strictly static, the HTML can also be dynamically created by a servlet or a JSP page.

GWT is organized into modules such as the USER module containing the core functionalities (UI components, DOM programming) or the JUnit module to provide a testing framework. Each module can be added independently (*via* the inherits tag) to a project so

only the necessary ones are used during the compilation. Modules can contain associations to external JavaScript code or CSS files.

When the GWT application is written and compiled, the developer has in his hands HTML and JavaScripts files, with all the necessary files such as the CSS stylesheets and pictures to go with them. These files will be deployed on a web server or a Java servlet with the appropriate server-side coding to make them available on the Web. It's also possible to deploy GWT application on the Google App Engine.

Any GWT application will have at a given time to communicate with backend servers. A GWT optimized RPC can be used to interchange data with a Java servlet. GWT RPC is independent from the protocol used to make the calls. Custom HTTP requests can also be done to interchange HTML, plain text, JSON, XML or other form of data.

2.2 ADOBE FLASH & FLEX

Flash is Adobe's ecosystem to provide a set of technologies helping to create and distribute applications and content. The core of this solution for RIAs development is Flex, which is described on the editor website as "[...] a free, open source framework for building highly interactive, expressive web applications that deploy consistently on all major browsers, desktops, and operating systems". A common end-product illustration of this platform would be Mint.com, a website providing personal finance management services. Adobe platform provides tools to build from simple rich internet components in websites to full-scale browser applications and browserless web applications.

Development characteristics

Flex applications are built with 3 programming languages (MXML, ActionScript and CSS) and more than one hundred prebuilt rich application components if necessary (both Flex and 3rd party). MXML is used for the UI and the behavior of the application. ActionScript (in its 3.0 version to this date) is the programming language used to code the client-side business logic and to create classes.

The free Flex SDK offers a compiler and a debugger. Developers have also the possibility to use the commercial product Flash Builder (formerly named Flex Builder) to help them create Flex applications. Adobe's IDE is built on top of the Eclipse platform and can be used as a standalone application or directly on Eclipse with the help of a plugin. Flash Builder includes a design view, an enhanced code views, a visual debugger, a visual profiler,

a network monitor and a services view. Code source files (.mxml and .as) are compiled into a Flash SWF format binaries ready to be run in Flash Player or Adobe AIR.

Deployment over Flash Player or AIR

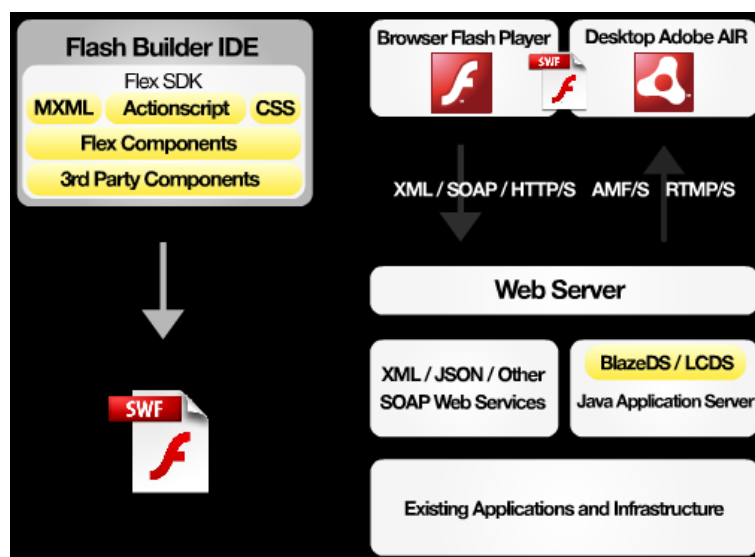


Figure 2.5 - A SWF file being deployed [2].

Flex content and applications can be rendered to the user *via* two client-side solutions: Flash Player and Adobe AIR. The two RTEs can help distribute developer's work across browsers, desktops and devices such as smartphones, netbooks and other handheld devices. Both solutions can be used on a large range of operating systems, hardware and browsers. Flash Player is one strong advantage for Adobe's platform as 98% of internet-enabled personal computers are compatible with Flash. Moreover Flash benefits of a long history of rich content delivery as the technology have been used for advertisement, game applications and video delivery since its early versions.

Flash Player lets the end-user open rich applications within the browser on both desktop computers and mobile devices. No application installation is required and updates can be pushed directly to the website. If the RIA runs within a browser, all information will be lost when this one is closed. Moreover, UI controls are directly linked on how the browser application is integrated to the desktop. Limited desktop integration and local storage are possible, but all restricted by the browser.

When the development is done, the SWF file is moved to the server where it can be requested by the user. The SWF file is then downloaded on the client where it will run (the application does not run on the server). The RIA can communicate with server

technologies (PHP, ColdFusion, ASP.NET and JSP) as needed. If the application needs further data, a request is sent to the server, which in turn can make a request to back-end resources (a database for *e.g.*). The data is sent back to the server, and then to the client on which the Flex application will update flawlessly its state according to the data received.

In order to implement a SWF file into a webpage, the Web developer can use an HTML wrapper. Flex can also work with JavaScript on the client. If the Player is installed on the client, the Flex application will be executed properly with the HTML document in the browser:

Generating a SWF and Embedding in HTML

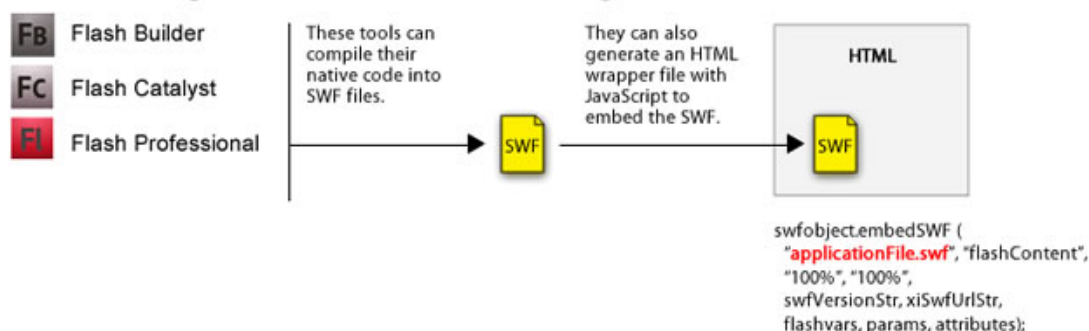


Figure 2.6 - A SWF file embedded to HTML document [69].

The other alternative is to use the object tag to embed the flash application. An example code taken from the W3Schools website is shown below to illustrate how it is coded [145]. A Silverlight application can be embed in a HTML page in a similar manner.

```
<object width="550" height="400">
  <param name="movie" value="somefilename.swf">
  <embed src="somefilename.swf" width="550" height="400">
</embed>
</object>
```

Adobe AIR gives the developer the ability to build applications that run without a specific browser. It provides the client applications a better desktop integration, more convenience and more functionality such as clipboard access, system events and more. SWF applications can be downloaded and installed like a desktop application or installed directly from the browser. An API is provided to ease updates. AIR RIAs can store and access local data and run offline. Compared to a Flash Player installed on nearly all internet-capable desktop computers, Adobe AIR has to be installed by the end-user. The installation can be done both manually or automatically when an RIA built for AIR requires it. Moreover, Adobe AIR can be used to run HTML, JavaScript and AJAX built applications.

2.3 CURL

Curl has been marketed as a platform to seamlessly migrate client-server applications to the internet without giving up on interactive features. The platform is highly enterprise-centered and aims to produce B2B and B2C applications capable to handle larger datasets and complex interfaces. The Curl platform includes the following components:

- The Curl programming language, designed specifically for developing on the web, offers rich text formatting and GUI layouts like HTML does, the full object-oriented programming paradigm of C++, C# and Java, and delivers the ease of the JavaScript scripting language.
- An RTE to run Curl applications and render code, text and graphics on the client. Available as a browser plugin, the end-user simply requests for the Curl application with its URL. Curl applications can also be installed and run directly from the desktop (like Adobe's AIR), offline using local data if necessary.
- An IDE providing the tools needed by developers to build Curl RIAs. It includes a visual "WYSIWYG" editor, a source code editor, a debugger and deployment tools among others. It is available as a standalone application or as a set of plugins for Eclipse (Curl CDE).
- The Curl Web Services Software Development Kit (Curl WSDK) to provide a way to use data resources and web services directly into Curl applications. This can be done with SOAP Web Services and WSDL, or by using the WSDK XML document model.
- The Curl Data Kit (CDK) and Data Kit Data Services (CDK-DS) ease the development of data-centered applications. The first one provides support of local SQLite databases and facilitates the development of occasionally (OCC) connected applications. The second one provides a library to help building applications that need to interact with servers for data resources.

2.4 OPENLASZLO

OpenLaszlo (OL), by Laszlo Systems, is an integrated platform to build and deploy rich internet applications. The platform is characterized by two major features: a custom XML-type language known as LZX and a Java servlet that officiates as a proxy server if the developer desires it.

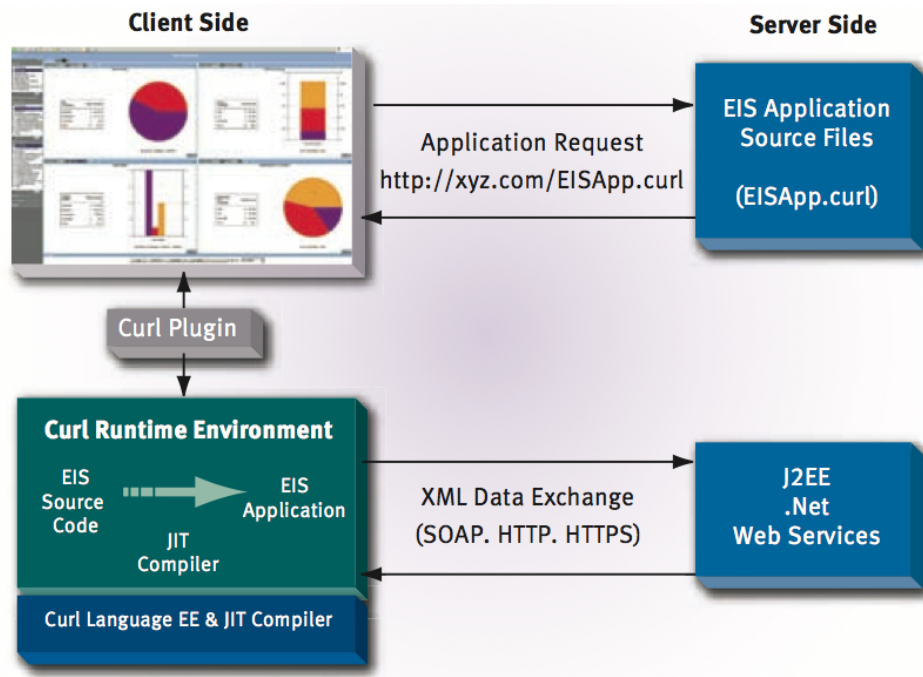


Figure 2.7 - The Curl platform [65].

The LZX language

“LZX is an object-oriented, tag-based language that uses XML and JavaScript syntax to create the presentation layer of rich Internet applications” [109]. XML and JavaScript being popular languages for the Web, the learning curve for LZX shouldn’t be steep for experienced Web developers. LZX code is compiled into the SWF format to be executed in Flash Player, or it is compiled into DHTML files which are run directly in the browser. DHTML contains JavaScript, however it is not the JavaScript the developer may have coded, but more a new JavaScript code generated during the compilation process.

LZX provides everything needed to handle animations, layout, databinding, event handling or server interchange. The XML part of the language will take care of the presentation, while the JavaScript part will take care of the application logic. Pre-built widgets and components are also available for common controls. The code of whole application can be contained from a single to multiple files.

Like HTML and XML, LZX code can be written with a simple text editor. Eclipse IDE (IDE4Laszlo) can however be used to ease the development of OL applications. The application can be compiled in the IDE if necessary, though the OL Server can take care of that part. When the LZX files are in the proper server directory, the application can be tested *via* the browser. If changes appear in the files, the app is re-compiled and cached to be available right away in the browser (but more on deployment options hereafter).

The platform also contains an Unit Testing API, a debugger, and a size and speed profiler to pinpoint where performance drops may happen.

Proxied & SOLO deployment

On the deployment side, OpenLaszlo has a variety of options to offer. First a developer has the choice between using the OpenLaszlo Server as a proxy, or deploy what he's built as a standalone application (called "SOLO" deployment).

- Proxied applications are deployed as LZX files and the OL Server compiles source code when needed. The output files are sent and executed on the client. Data interchange and client/server communications are all proxied by the OL Server. The OL Server is a Java application executed in a servlet (OL Server comes with the Tomcat servlet by default). It contains 4 major parts: the Interface Compiler, the Media Transcoder, the Data Manager and a Cache. The OL Server takes care of the communications with backend resources whenever needed.
 - The Interface Compiler converts OL code into the necessary output files. The files are cached and sent to the client. The Cache contains the latest version of the compiled application, preventing compilation at every request.
 - The Media Transcoder prepares the media content in a single format to be sent to the client. The objective is to have an unique way to take care of the media.
 - The Data Manager compresses the data in a format readable by OL applications. It provides also data connectors to help applications retrieve data with XML, thus exposing the applications to backend resources such as databases and XML-based Web services.
- SOLO applications are precompiled and can be deployed on other servers than the OL Server. No proxy is needed in this mode. This option is easier and faster for delivering web applications but it's less rich in features compared to the proxied mode. For example, the SOLO mode does not support SOAP and XML-RPC requests, or media files other than SWF, MP3 and JPG [110].

An OL application contains the Laszlo Foundation Class (LFC), a runtime library providing application services, graphics rendering and media playback, a data loader/binder to associate coming data to the proper presentation elements (menu or fields for *e.g.*), and an event system to deal with events such mouse clicks or server pushes. All communica-

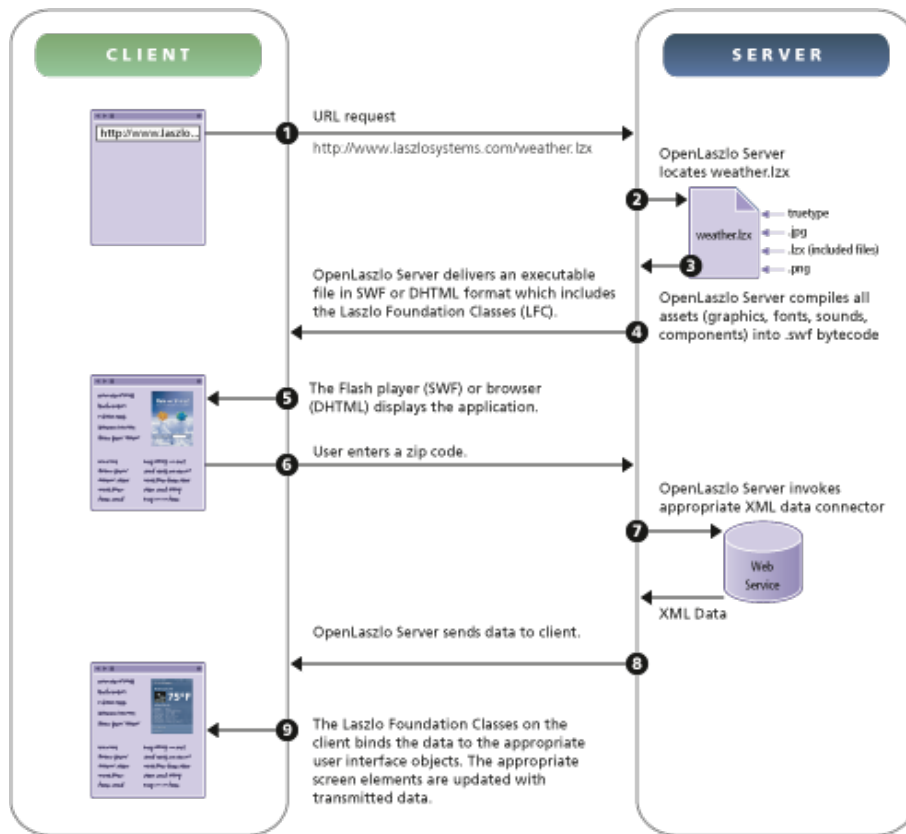


Figure 2.8 - A typical workflow in proxied mode with the OpenLaszlo Server [108].

tions are usually done over HTTP(S), including streaming and notifications. However, streaming is possible over RTMP with the help of a Media Server [111].

Apart from the proxy or SOLO deployment decision, the developer has also the option between 2 runtime environments: Flash or DHTML. If the application has to be executed in Flash Player, LZX code will be compiled in the SWF file format. The SWF is either sent embed in HTML document, or sent as a simple SWF file, and Flash Player is used for rendering purposes (Flash services are not used here). If one wants to develop an OL application that runs natively within the browser, he can then compile a DHTML output (HTML, JavaScript & CSS files). If audio and video are wanted in DHTML mode, a SWF player can be embed in the HTML. The platform is open to any future RTE that can become popular in the future.

2.5 JAVAFX

JavaFX is the platform developed by Sun Microsystems to build and deliver rich internet applications. The platform is powered by the Java technology: web applications are written in JavaFX Script (although Java code can also be integrated into the program), the source code is compiled into a Java bytecode that can be executed on any desktop or browser that supports the Java Runtime Environment (JRE), or any mobile phone that supports the Java ME.

Development tools

Three tools bundles are made available to help the development of JavaFX applications:

- A developer bundle: containing NetBeans IDE for JavaFX, a development environment to build, preview and debug JavaFX applications. It comes with the JavaFX SDK. If one already possesses NetBeans IDE, he can use the plugin to support the development of JavaFX applications in the software.
- A designer bundle: consisting of JavaFX Production Suite. The Production Suite contains plugins for Adobe Photoshop and Adobe Illustrator allowing to save creative media into the JavaFX format. Saving creative content into the right format helps to visualize how the content will look when the application will be executed on the client end, allowing designers to make the right changes to fit the needs of the application. The Production Suite contains also JavaFX Media Factory, able to convert SVG graphics into the JavaFX format and to visualize any JavaFX creative content.
- The JavaFX SDK: a stand-alone SDK is available for the ones who want to develop JavaFX application in a command-line interface. The SDK contains a desktop runtime, a mobile and TV emulators, APIs and their documentation, a compiler and samples.

Deployment options

JavaFX applications are powered by Java, which is available on millions of computers and other devices. JavaFX applications can only run on desktop computers, laptop computers and mobile phones for now. But compatibility with televisions and other devices will be added later. JavaFX can be delivered as a Java applet within the browser, with the Java Web Start technology, or as a standalone desktop application requiring no web browser.

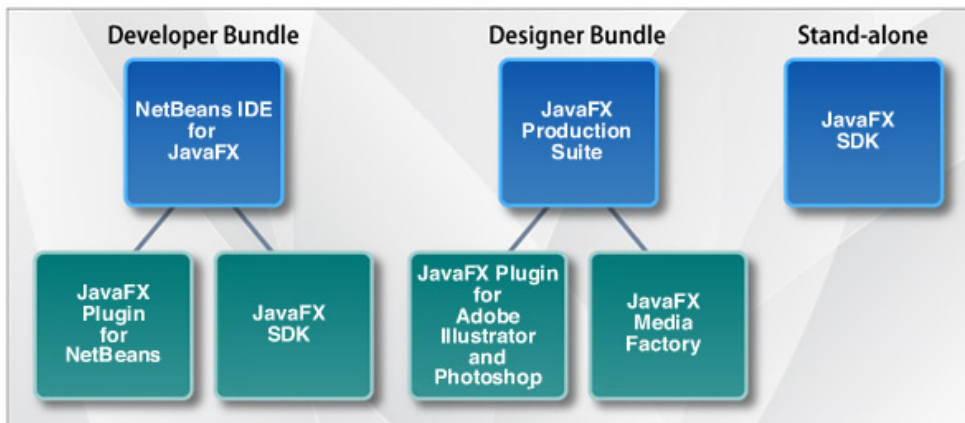


Figure 2.9 - An overview of the JavaFX tools [23].

Java Web Start applications are almost like standalone Java applications with the difference that they are delivered *via* the browser when the application is required the first time. The application is downloaded and the user has to accept a security certificate to validate the application. Updates for Java Web Start applications are automatically downloaded.

2.6 MICROSOFT SILVERLIGHT

Microsoft Silverlight has been seen as an answer to Adobe's Flash platform. Microsoft's platform helps develop web applications for the browser, the desktop and mobile devices, either running online or offline. A Silverlight application has the possibility to contain media content, graphics, effects and interactivity running in a unique runtime environment. The Silverlight platform contains three major components:

- A core presentation framework: to focus on UI matters. The framework is used for UI rendering (graphics, animations and text), input controls (keyboard, mouse, *etc.*), media playback and management, using the deep zoom, accessing widgets, the layout, data binding, handling digital rights management, and for providing XAML to spice up the presentation layout. XAML is also used as a joint between the .NET framework and the presentation layer.
- A .NET framework for Silverlight: is a subset of the .NET framework specially design for Silverlight, containing libraries and other components to build Silverlight applications. It contains WPF controls, the base class library, the common language runtime, and data handling tools.

- An installer and an uploader: to help first users to install the Silverlight applications and manage their updates.

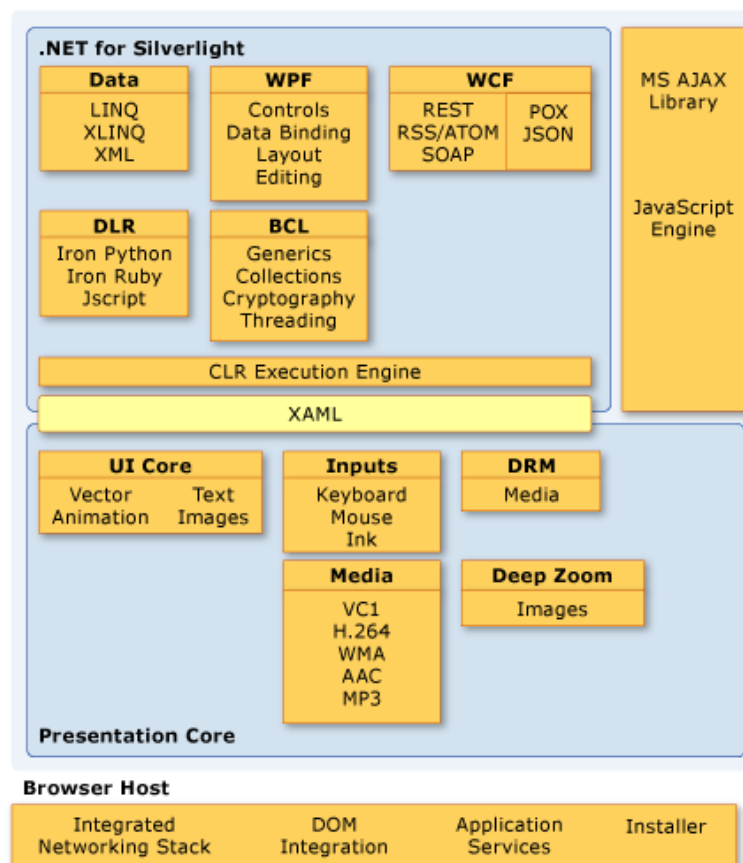


Figure 2.10 - An overview of the Microsoft Silverlight platform [99].

To help developers building applications, Microsoft can offer tools featuring all they need for Silverlight development. Visual Studio can be used for design creation, such as UI development for *e.g.* And Microsoft Expression Blend can be used to create graphics, animations and interactions.

2.7 DESIGN PATTERNS FOR RICH WEB APPLICATIONS

A pattern can be seen as a solution solving common recurring problems regarding software design. Using design patterns for software development have a variety of benefits. The software will be easier to maintain, to test, to expand and to understand the app code in

the future. They set up a template regarding the construction of the application, and how it should be coded.

A design pattern can ease the development of large applications by providing a way for developers to work together on the same code, while keeping a lean code. This works usually because focus is divided in multiple parts like we will see hereafter. A first good example of a design pattern would be the Model-View-Controller (MVC).

Model-view-controller & SproutCore MVC+SDR

MVC applications divide their code into three separate sets, each one of them being responsible of their part. The pattern divides data, interface and application logic.

- The *model* element manages data and the business logic of the application. The model does not care how the output will be rendered for *e.g.*
- The *view* element takes care of the display and the user interface (which is not necessarily the device screen). The view does not care of what will be done when it receives an input for *e.g.*
- The *controller* fills the gap between the previous parts. A controller modifies the model according user inputs, or sometimes it can directly modify the view without touching the model

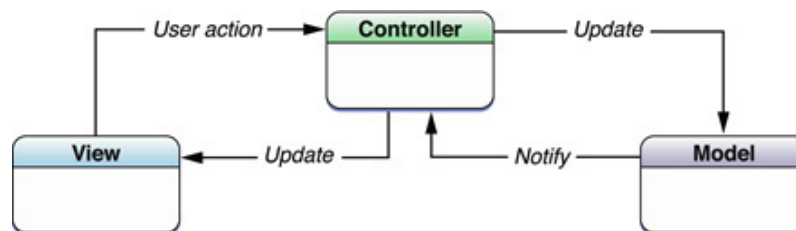


Figure 2.11 - The Model-View-Controller paradigm [25].

MVC is a pattern commonly used for desktop applications design. This is the case for Cocoa applications (Apple's native programming environment). However, it can also be used for web applications, most notably for SproutCore applications. SproutCore MVC is loosely based on Cocoa MVC, and expands the MVC pattern with three other sets (to address web apps new challenges):

- The *server interface* handles the data circulating between the server and the model layer.

- The *display* renders the user interface. It takes care of the actual “painting”. It can use library such as jQuery and the web browser to render the interface. View on the otherhand controls the display and responds to events such as mouse clicking or keyboard presses.
- The *responder* layer controls the state of the SproutCore application (for example using a specific UI to do a specific task). This is the part to control models, views and factors according a state or other factors.

The MVC+SDR model is a pattern embracing the overall architecture of the SproutCore application. Other design patterns can be implemented at a smaller level. These include run loops, singletons, observers, bindings, delegates and many more.

A singleton for example is one object of a particular class, having its own properties and methods. Controllers in SproutCore are usually singletons.

Model-view-presentation

The model-view-presentation pattern can be seen as a derivative of the MVC model.

- The *model* contains the data of the application, and provides a way to access it.
- The *view* provides a way to display something on the screen.
- The *viewmodel* is “intended to be an abstraction of the UI” [31]. It also binds the data between the model and the view.

The MVP model is the pattern chosen to work with the Google Web Toolkit. Google arguing it is better for separating concerns, testing and develop large scale applications [117]. The GWT add however a new component: ApplicationController. The new element handles the logic that does not concern the presenters.

Model-view-viewmodel

The MVVC model is another architectural design pattern used to engineer applications. It is coming from Microsoft, and it is based on the MVC pattern. It works well to develop Silverlight applications.

- The *model* contains the data to be retrieved or manipulated
- The *view* provides the UI and displays the model’s content to the user.

- The *presenter* fills the gap between the model and the view by accessing the model to retrieve data. It also handles user inputs and adjusts the model in consequence.

3-tier architecture

When it comes to RIA architecture, the n-tier model is often mentioned. The three-tier architecture is the N-tier model commonly used. It has the 3 following tiers:

- The *presentation tier* contains everything related to the user interface. It will make the requests needed to render the content.
- The *logic tier* contains the business logic of the application. Low-level processing and number crunching are done at this level.
- The *data tier* contains the resources such as raw data to be displayed.

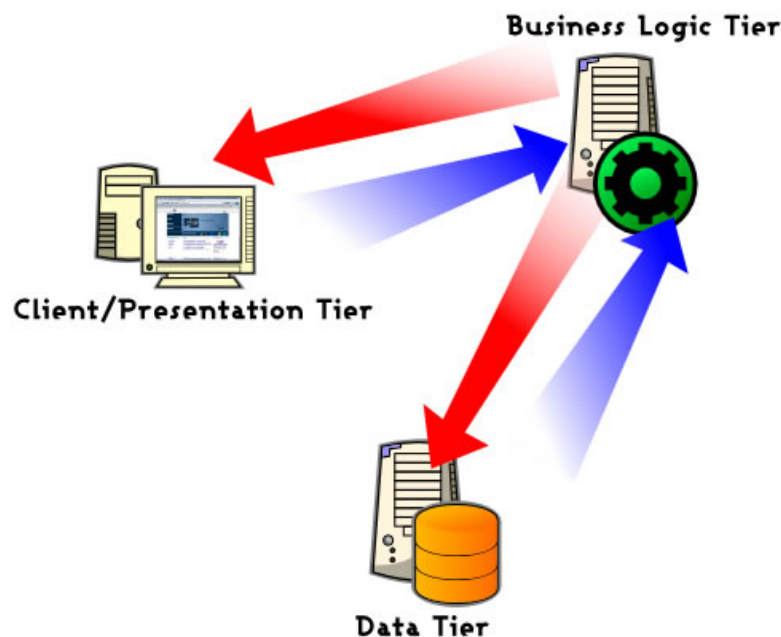


Figure 2.12 - The common 3-tier model [116].

Adopting a n-tier model have several benefits. First, separating core components (presentation, logic and data in the 3 layers approach for *e.g.*) of the application make them upgradable indepently from each other. Large applications are also easier to develop as developers can separate the work between themselves according specific concerns. Separating web applications into tiers make them also more scalable [116].

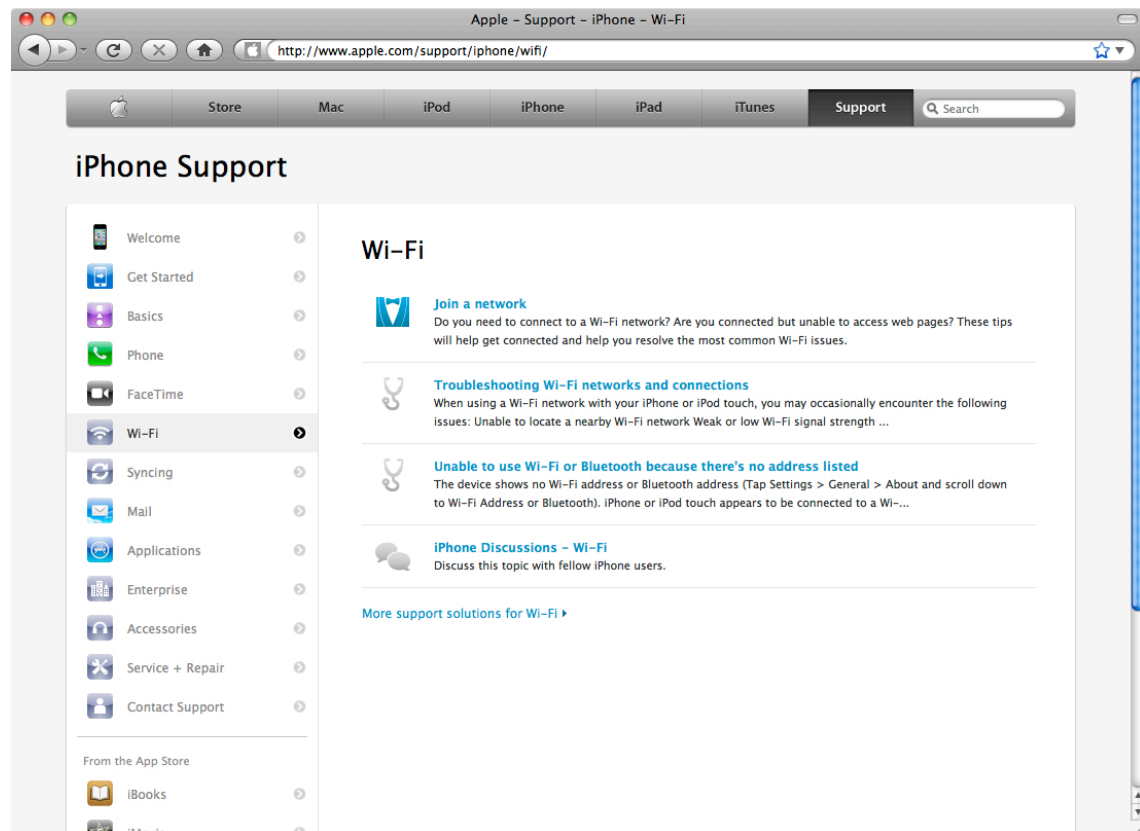


Figure 2.13 - An example of the master/detail screen layout [57].

2.8 UI LAYOUTS, CONTROLS AND OTHER COMPONENTS

When it comes to analyze rich web applications development platforms, it is often advertised what and how many prebuilt elements are available to developers. These are important to facilitate the development of websites and web apps. First, they let the developer use prebuilt commonly used components without to code them on every single new project he is working on. And secondly, they help less experienced developers to use complex functionalities [53]. Re-usability is encouraged, that is why they can be considered as patterns. They try to fit what the end-user wants: a creative application, an informative application, or a platform to realize a specific task [105]. The UI (layout, behavior, controls, effects, *etc.*) may differ according to these needs. But patterns can emerge.

Using these elements alone does not built a rich internet application. They can however contribute to it. That is why platforms usually provide sets of prebuilt elements to the developers. Some libraries and toolkits are also available with the simple goal to provide those elements. There are several kind of prebuilt elements for Web development: screen

layouts, controls (or widgets), or effects. And here are examples of some of them. It should be noted that a popular article written by T. NEIL [102] (including also [103, 104]) on the Web provides a large collection of patterns of all sorts. Some following examples are directly inspired from that article.

Screen layouts

Layout patterns defines how the general user interface is constructed. There is no best or worst patterns, but a pattern that fits the user the best. Typical layout patterns include the master/view, the palette/canvas or the dashboard layout. The master/view layout is illustrated in Fig. 2.13 (page 37). The palette/canvas pattern can be illustrated by Fig. 3.3 on page 57. Fig. 1.6 (page 16) shows the dashboard layout in action with a Curl application. NEIL's article describes 12 of them.

Controls

Controls (also called widgets) are parts of the graphical user interface. They are commonly implemented to be manipulated by the user. Again, some re-usable patterns emerge and can be implemented directly by the developer into his website or web app. When they are available out of the box in a particular platform, they can be integrated with a minimum amount of code, and are usually skinable (through CSS for *e.g.*). NEIL's article illustrates 30 common controls and comments in which solutions (jQuery, Flex, SproutCore, *etc.*) they are available. Popular controls include the date picker, the carousel, the dialog box, progress bars, buttons, sliders or the auto search suggestions (the search engine being illustrated in Fig. 2.14).

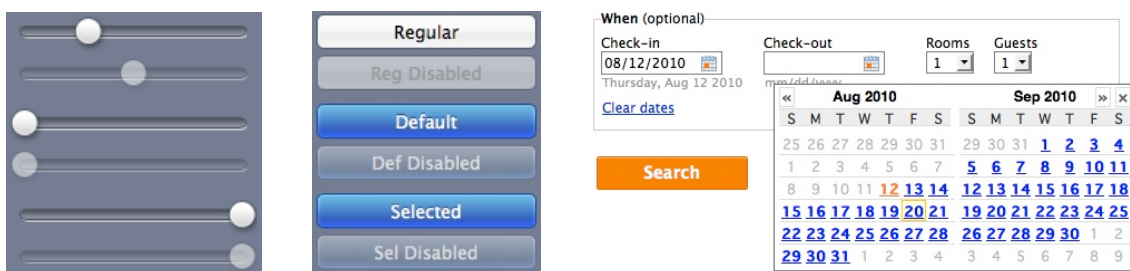


Figure 2.14 - From left to right, illustrations of sliders [119], buttons [119] and a date picker [94].

Effects

Effects are important for websites and web applications as they can be used to indicate a change of state. They include (among others) show and hide, bounce, highlight, animations and 2D and 3D transitions. Fig. 2.15 demonstrates the `animate` effect provided by the jQuery UI library.

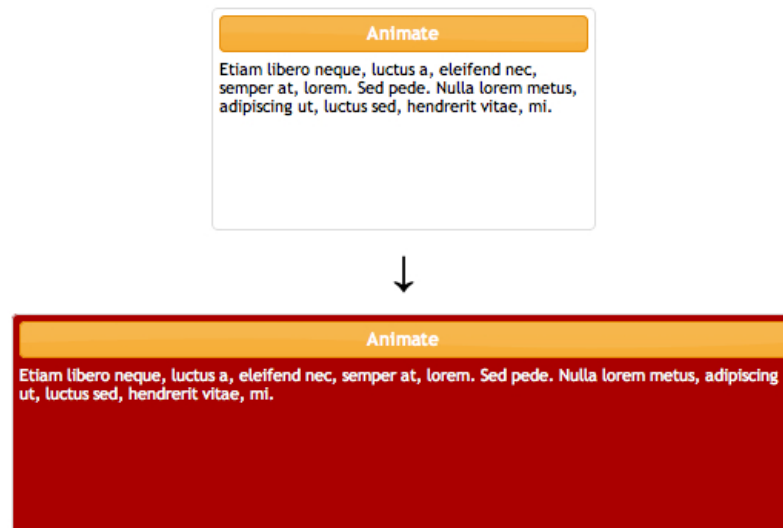


Figure 2.15 - The `animate` effect, part of jQuery UI [92].

2.9 FEATURES SELECTION & PLATFORMS COMPARISON

This section focuses on the technical details of every single solution that was discussed in this chapter. Third-party expansions are not considered here. So for example when it is said that jQuery does not support the offline feature, it is not entirely true as the jQuery Offline plugin could be used to that end. The features elected for the comparison are explained hereafter:

Programming language

Indicates which languages can be used to write the rich internet application. JavaScript is common used client-side scripting language to help spice up UIs and websites within the browser. However, it isn't an exclusive language to build rich internet applications (or standard web applications for that matter).

Client-side Technology

Represents the technologies used to run the application on the client. The triplet HTML,

CSS and JavaScript is the Web standards alternative. SWF files (Flash platform) are also popular among the Web.

Server-side Architecture

Details which technologies can be used to power the server communicating with the client. Common technologies include .NET, PHP or simple HTTP servers.

Client/Server Communication

Attempts to define which technologies can be used for client-server communications, to retrieve data from remote databases for example.

Platform tools

They are all the tools that are included within the solutions. They can include a SDK, designer tools and developer tools. Third-party softwares or support will not be detailed here.

RTE

Stands for runtime environment. The RTE is a key component for web applications as it is exactly where the application will be executed when the app is requested. If the runtime is poorly written, the experience using the application can be affected badly. Most traditional web applications are using a raw web browser as environment, as they can execute JavaScript natively for *e.g.* Things are different among the RIA world: some solutions use the web browser, and others make a browser plugin available to execute their applications. Browserless RTEs are also available in some platforms.

Browser support and other devices support

Detail on which devices the applications of a specific platform can be run in. Ubiquity can be an important factor for a developer who wants to make its application available to the larger public possible.

Consistency

Remarks if consistencies may appear between different clients using the same application (a SproutCore app used in Safari and Firefox for *e.g.*).

Offline capabilities

Details if the a platform supports offline apps. It means usually that the web application will have to be cached or installed on the client host. As desktop installation, offline capabilities let web applications feel more like desktop applications.

Desktop installation

Mentioned if it is possible to install an application from a specific platform on the desktop, freeing its execution within the browser. This option puts RIAs one step closer from standard desktop applications.

External Input Support

What inputs can receive the client application? Keyboard and mouse are common, but other inputs such as *touch* are becoming more and more needed for example.

Components and widgets

They are commonly included into RIA solutions to help developers not reinventing the wheel. The point is to let developers access a base of reusable components they can include in their applications. Some platforms includes more components and other widgets than others.

Effects and animations

Effects and transitions are important to help the user to understand a change of state within the application.

Audio and video

Is video and audio support natively available on the platform? Some solution requires a plugin (Flash Player or QuickTime) to take care of video and audio playback within their applications, while other platforms let their applications execute natively video and audio content within their RTE.

Hardware optimization

Optimization is possible in some solutions. It can indeed be interesting to allow the processing of HD video playback directly to the GPU for *e.g.*, freeing CPU cycles for something else.

Local Data Storage

Do web applications have the possibility to store data locally? This is important for further web/desktop integration, thus enhancing user experience.

Accessibilty

The Web has always wanted to be accessible to everybody, even people with disabilities. Hence a criteria to show how platforms are doing regarding accessibility.

Printing

Does printing is possible. It is for applications using HTML, CSS and JavaScript. However, things can be a bit more complex with other platforms.

SEO

Stands for Search Engine Optimization. This field details if the solution is optimized for search engines or not. SEO is one main limitation of web applications as search engines cannot usually index properly the content available trough the applications. However, workarounds can be found to help the process.

Latest analyzed version

Is just mentioned to specify which version of the platform was used to make the table.

Price

The minimum fee to pay to use the tools of a specific platform.

Adoption Examples

Shows a few remarkable clients using the technology being analyzed.

The comparison tables can be seen on page 45 and 46. The information was retrieved on vendors' websites [1, 8, 15, 22, 33, 38, 39, 91].

2.10 A SHORT DISCUSSION ON WEB APPLICATIONS TECHNIQUES

Flash has a strong presence in the industry, and Adobe is developing RIA tools for a long time now. The Flash player can be considered as an “industry standard” because of its importance on the Web. More than 95% of browsers are Flash-compatible (*i.e.* Flex compatible). But it is interesting to see how its opponents perform in terms of runtime market penetration (*cf.* Fig. 2.16).

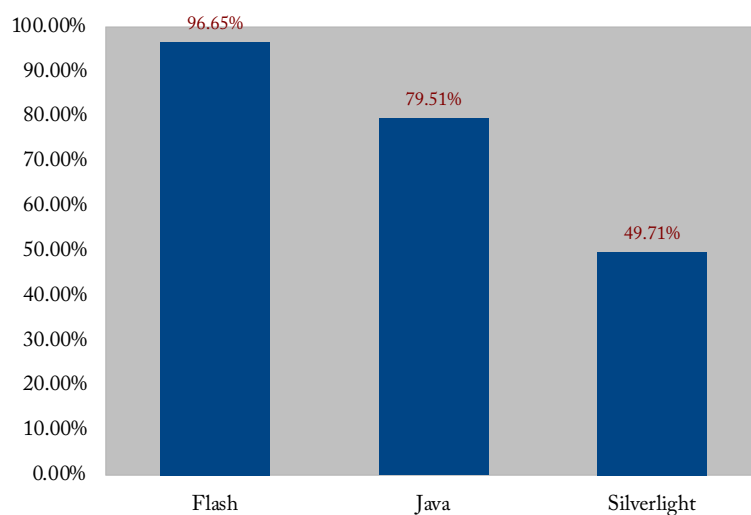


Figure 2.16 - *The market penetration into the browser of Flash, Silverlight and Java [122].*

The data comes from the website statowl.com [122], and shows that Silverlight has the least market share of all three. When, it comes to the Ajax technique, no plugin is

necessary as HTML, CSS and JavaScript are directly handled in the browser. It means no stats has to be calculated as virtually all browsers supports HTML4 (it is different for HTML5, but more on that in Chapter 3), CSS2 and JavaScript.

OpenLaszlo uses either Flash Player or directly the browser, so its penetration is at least equals to Flash's. There is however no viable data for Curl RTE's market penetration.

A runtime market penetration is important as the factor can orientate developers more towards one platform than the other. Ajax is popular because it relies on Web standards, meaning the web application will run directly into every modern browser, assuming that JavaScript is allowed. For this concern, Ajax techniques and Flash applications have the advantage. However, Silverlight market share increases steadily, so it will be interesting to see where it stands in the coming months.

If HTML5 features are used to design the web application, problems may arise with Internet Explorer 8 as the browser does not support well the latest standard. However, IE9 (expected at the end of the year) promises to assure HTML5 compatibility.

Apart form market penetration, developers will also evaluate the UI elements included in the platform solution. Tools to build third-party components and to skin existing components are also appreciated. Re-usability is important. These characteristics give an advantage to jQuery, Flash and Silverlight.

Tools are also crucial to help the developer in his task. He will need usually an IDE, a unit testing tool and a debugging tool. Designer tools are also an advantage. From this perspective, Flash, JavaFX and Silverlight lead the way with their fully integrated design and development tools.

Each platform has its own goals. Flash, Silverlight and OpenLaszlo can be seen as direct competitors, but they do not compete directly against Curl (enterprise-centered) and jQuery (just a low-level JavaScript library) for example.

Flash has a good reputation in the industry, and enjoys a significant market penetration of its runtime Flash Player. Adobe came in the game when they bought Macromedia in 2005, bringing in house all their competencies in the matter. Flash Player is ideal as it is not too heavy nor the light (Ajax engines are lighter), providing a good option to access rich experiences in the browser [126].

It is worth noting that since iOS devices do not support Flash, some websites have converted their Flash content (videos, ads, *etc.*) into HTML5. This can not be seen as significant threat for Flash to this day, but maybe we will see in the future more and more websites whose content will be rendered with HTML5, CSS3 and JavaScript exclusively. Flash is not dead anyway as it is excellent technology applicable in a wide range of domains (such as games for example).

Ajax is an extremely popular technique to build interactive web apps. The use of web standards without plugins contributes to its appeal. However, Ajax relies heavily on JavaScript. That is why consistency is not assured among the large variety of browsers available out there. JavaScript performances play also a key part in the overall usability of the web applications.

Silverlight can be seen as the greatest menace of Flash. Both technologies work the same way, through a plugin, and propose the same features. Microsoft did a great job in terms of market penetration as in only a couple of years they managed to have a bit less than 50% of desktops installed with their plugin. Microsoft is a serious menace for Adobe because the company is much larger in scale and importance.

Features & Characteristics	jQuery	SproutCore	Google Web Toolkit	Flex/Flash	Silverlight	JavaFX	Curl	OpenLaszlo
Programming Language	JavaScript	JavaScript	Java	MXML, ActionScript & CCS	Any .NET Language	JavaFX Script & Java	Curl	LZX
Client-side Technology	HTML + CSS + JS	HTML + CSS + JS	HTML + CSS + JS	SWF	Silverlight (.xap)	Java	Curl	SWF or HTML + CSS + JS
Server-side Architecture	HTTP, PHP, ColdFusion, ASP.NET, Perl, Rails & Others	HTTP, ColdFusion, PHP, ASP.NET, Perl, Rails & Others	HTTP, ColdFusion, PHP, ASP.NET, Perl, Rails & Others	PHP, ColdFusion, BlazeDS, LiveCycle Data Service, HTTP, Web Services, J2EE & ASP.NET	ASP.NET, PHP & JSP	J2EE & Web Services	J2EE, .NET & Web Services	Standard HTTP Server or OpenLaszlo Server (Trough J2EE or Java Container)
Client/Server Communication	JSON & XML	Ajax Requests & JSON	JSON, XML & GWT Optimized RPC	XML, SOAP, HTTP, AMF, REST & RTMP	HTTP, SOAP, POX, WCF & JSON	HTTP, XML & JSON + Everything Coming With Java	XML, SOAP & HTTP(S)	XML-RPC, REST, SOAP, JavaRPC, HTTP & RTMP
Platform Tools	jQuery Library	SproutCore Framework	GWT SDK, Eclipse Plugin & Speed Tracer	Flex SDK, Flash Catalyst, Flash Builder, CS Suite (For Creative Content) & Eclipse Support (Standalone or Plugin)	Expression Studio & Visual Studio	JavaFX SDK, NetBeans IDE Support & Java Production Suite (For Creative Content)	Curl IDE, Curl CDE, Curl WSDK, CDK & CDK-DS	OL SDK & Eclipse Plugin

Features & Characteristics	jQuery	SproutCore	Google Web Toolkit	Flex/Flash	Silverlight	JavaFX	Curl	OpenLaszlo
RTE	Web Browser	Web Browser	Web Browser	Flash Player & AIR	.NET Common Runtime	JRE & Java ME	Curl RTE	Flash Player or Web Browser
Browser Support	All Modern Browsers	All Modern Browsers	All Modern Browsers	All Modern Browsers	All Modern Browsers	All Modern Browsers	All Modern Browsers	All Modern Browsers
Other Clients Support	Any With Web Browser	Any With Web Browser	Any With Web Browser	Any Device Running Flash Player (i.e. No iOS)	Very Limited (Only on Symbian OS for Now, WP7 Planned)	Any Device Supporting JRE or JME, TV, Blu-Ray Players and Game Consoles Are Planned (No iOS)	No (Only via Curl RTE on Desktop Computers)	Any Device Running Flash Player (i.e. No iOS)
Platform Consistency	Differences May Appear	Differences May Appear	Differences May Appear	No Differences	No Differences	No Differences	No Differences	No Differences If Flash Is Used
Of ine Capabilities	Not Native	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Desktop Installation	No	No	No	Yes (Trough AIR)	Yes	Yes (JRE)	Yes (Curl RTE)	No
External Input Support	Keyboard & Mouse	Keyboard, Mouse & Touch	Keyboard & Mouse	Keyboard, Mouse, Webcam, Microphone & Multitouch Support	Keyboard, Mouse, Webcam, Microphone & Multitouch Support	Keyboard, Mouse & Touch	?	Keyboard, Mouse, Webcam & Microphone

Features & Characteristics	jQuery	SproutCore	Google Web Toolkit	Flex/Flash	Silverlight	JavaFX	Curl	OpenLaszlo
Widgets & Components	Yes (A Lot)	Yes (A Few)	Yes (Moderate)	Yes (A Lot)	Yes (A Lot)	Yes (Moderate)	Yes	Yes
Effects & Animations	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Audio/Video Support	Yes (Plugins or HTML5)	Yes (Plugins or HTML5)	Yes (Plugins or HTML5)	Yes	Yes	Yes	Audio	Yes
Hardware Optimization	No	No	No	Yes	Yes	Yes	Yes	Yes
Local Data Storage	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Accessibility	WAI-ARIA Support	N/A	WAI-ARIA Support	20+ Prebuilt Accessible UI Components & MSAA Support	MSAA & Accessible Components	?	?	MSAA Partial Support
Printing	As Standard Webpage	As Standard Webpage	As Standard Webpage	Yes	Yes	Yes	Yes	Yes
SEO	Partial (Tags)	Partial (Tags)	Partial (Tags)	Partial (Trough Metadata)	Partial (Trough Metadata)	Partial (Trough Metadata)	?	Partial (Trough Metadata)
Latest Analyzed Version	1.4.2	1.0.146	2.0.4	4.0.0.14159 (Flex)	4.0.50524	1.3	7.0	4.8.0
License	GPL or MIT License	MIT License	Apache License 2.0	Various	MS-EULA & MS-PL	EULA	?	Common Public License
Price	Free	Free	Free	Flex SDK is Free. Tools from US\$249+	SDK is Free. Expression Studio from US\$599+	Free	Basic Version Free & Pro Version from US\$12000+	Free
RTE Penetration	Any Browser	Any Browser	Any Browser	<95%	>50%	>80%	?	<95%
Adoption Examples	Twitter, Netflix & Amazon	Apple (Mobile.me)	Google (Gmail) & Bong.TV	Mint.com & NASDAQ Market Replay	Helnix Media Show		Nexus & Hitachi Displays	Verizon & WalMart

ON STANDARDS...

THE World Wide Web Consortium (W3C), founded and directed by Tim BERNERS-LEE, is the predominant organization to maintain standards related the WWW. The W3C is working on numerous projects, with some of them leading to final *Recommendations*, which can be used to build a more consistent Web. The Internet Engineering Task Force (IETF) is another organization working on standards, but it is focused more on Internet standards (such as IPv4, IPv6 or TCP/IP) than *stricto sensus* Web standards. However, it works closely with the W3C on some projects. Ecma International is noted for its ECMAScript standardized scripting language, from which JavaScript is implemented.

Some specifications and work-in-progress projects are closely related to web applications and their enhanced version which are RIAs. For *e.g.* a major discussion among Web professionals is HTML5. The new implementation of the markup language could play a key role in the WWW's future. It is developed by the W3C HTML WG and the WHATWG, and is reviewed in further details in Section 3.3, starting page 51.

The point of this chapter is to review all the standards, or standards to be, concerning the development of richer web applications.

3.1 WHY WEB STANDARDS?

This section attempts to understand why standards are important for the sake of the Web, and its stakeholders (users, authors, developers, businesses, *etc.*).

A major advantage of the Web is its ubiquity. Internet and the WWW are available nearly everywhere, but on devices that can be radically different. Fig. 3.1 emphasizes that fact. The Web is commonly accessed on desktop computers, with the use of a Web browser. A

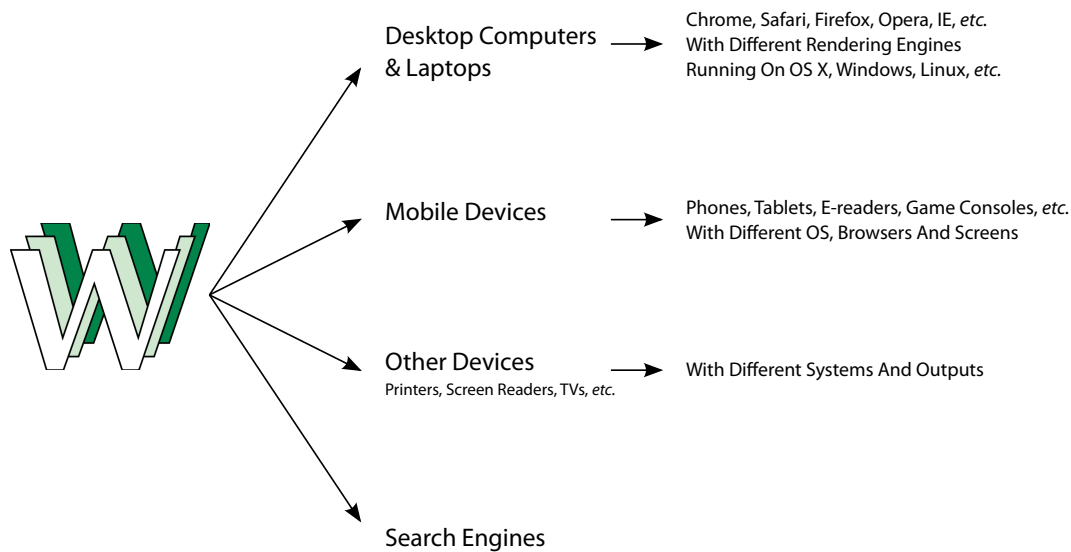


Figure 3.1 - *The Web among devices.*

variety of browsers are available according the OS used: Firefox, Safari, Chrome, Internet Explorer, *etc.* Every browser having a specific rendering engine which can interpret the code differently.

Aside of them, the mobile market cannot be ignored as more and more handheld devices are connected to WWW. We are talking here about mobile phones, portable game consoles or tablets... Again, different solutions are available to access the Web on these devices: IE Mobile, Safari, Opera Mobile, NetFront, BlackBerry browser, *etc.* Issues like the screen size or the bandwidth are becoming predominant.

Along with desktop and mobile uses, the Web can also be accessed on TV, printed on paper, read through a screen reader and so on. An adequate layout for these uses is also needed.

And finally, it can be useful if its content can be indexed by robots, to make it searchable *via* Google or Yahoo.

Content accessibility and compatibility on a variety of devices can be problematic. Standards can play a key role to ease the process of making the information available on every wanted end. That is exactly why standards such as HTML, CSS and JavaScript have been developed.

But the cross-device compatibility is not the only advantage of Web standards. Other qualities are on the side of standards such as HTML, CSS and JavaScript:

- Performances can be improved in many ways. A lean code is interpreted faster on the client browser. A standardized code usually leads to a smaller

code, which can have favorable consequences on the bandwidth consumption.

- Maintenance is facilitated. A standardized code is easier to read, so it is easier and quicker to make updates if needed. And if a new developer has to work on the code, it is easier for him to understand what has been done.
- Backward and forward compatibility. Backward compatibility means that even if a browser is too old, content will still be rendered even if some features will not be available. Forward compatibility is crucial as the code written today is better off if it is readable in the future on new devices.
- Accessibility can be improved for people with disabilities. Standards set up guidelines to make Web content easily manageable on screen readers such text-to-speech softwares or Braille devices.
- No dependance to proprietary lock-in. The WWW always has been designed has an open platform.
- It is easier for search engines to interpret standardized content. Standards can help optimize a website to make it more noticeable to search engines. It is important as it can lead to a better visibility among everything that can be found on the Web.

3.2 KEY WEB STANDARDS IN USE TODAY

Most of the Web is built around three core standardized technologies: a description structure featuring the content with HTML or XHTML, a presentation layer usually written with CSS, and a behavior block to manipulate the content and the presentation (JavaScript being the most common client-side scripting language). The DOM is also a common Web-related standard. On top of the standards, other pieces such as Flash or Silverlight components can be added and manipulated to enhance webpages if needed.

HTML

HTML stands for HyperText Markup Language. HTML documents are constituted of *elements* which are delimited by start tags and end tags: `<head>` and `</head>` for *e.g.* An HTML spec defines a set of elements and how they can be used. Elements can contain *attributes*, specified in the start tag, and controlling how the elements should work. To

create an hyperlink with the word “demo”, the `a` element and the `href` attribute can be used: `demo`.

XHTML (for Extended Typertext Markup Language) does the same job than HTML, but differently. XHTML uses XML for its syntax.

CSS

If HTML is describing the content, CSS is taking care of its presentation. CSS is used to set up the look and feel of webpages. The CSS language handles layouts, colors and fonts apart from the content, and can be used to define different presentation styles according to the end device (various screen sizes, printer, screen reader, *etc.*).

The presentation has been separated from the content to answer the following issues [131]:

- Accessibility to assistive technologies, such as screen readers that cannot interpret properly presentational code. Stripping the document from its presentational information makes it more accessible to more users.
- Document sizes: presentational markups are usually redundant, increasing the size of the document.
- Maintenance: the code can be more simple if presentation and content are separated. A single CSS stylesheet can affect an entire website.

JavaScript

JavaScript is a variant of the ECMAScript language. It is used for client-side scripting in Web browsers to spice up user interfaces and websites. Scripts written in JavaScript are embed in the HTML document with the `script` element.

JavaScript interacts with the DOM (Document Object Model). The DOM is a W3C Recommendation defined as a “[...] language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents” [127]. The document being in our case an HTML webpage. It is by interfering with the DOM that JavaScript adds interaction and dynamism into webpages and websites.

3.3 HTML5

HTML5 explained

HTML5 is the fifth major revision of HTML, the main markup language for the WWW. It is also supposed to be the successor of XHTML1.1 and DOM Level 2. If we refer the specification's words, HTML5 attempts to address the issues raised by “a vague subject referred to as Web Applications” [131]. It includes several significant features to help authors shape a better Web, answering a need of tools to build rich and interactive web content. Although it aims the backward compatible to make content written in HTML5 still available for older Web browsers.

HTML5 is maintained by the Web Hypertext Application Technology Working Group (WHATWG) since 2004, and the W3C HTML Working Group (W3C HTML WG) since 2007. The HTML5 specification is a work in progress in both groups, but parts of HTML5 are already supported by several browser.

The WHATWG was created due to concerns towards W3C's former decision to focus on XHTML rather than HTML. The venue was then constituted by Apple, Mozilla and Opera, with its members working on several projects with the successor of HTML as a primary focus. Aside from the WHATWG HTML spec, the group is also working actively on the Web Workers spec along with the W3C WebApps Working Group. All the WHATWG work, including specs other than HTML and Web Workers, are regrouped under the Web Applications 1.0 document [149]. These other specs (*cf.* Fig. 3.2) are however better known to be published at the W3C or the IETF [150].

In 2007 the W3C HTML WG was formed with the intention to work on HTML after all, using the WHATWG work as a base for its development. There are no plans to merge both groups, but they are working together on HTML5 since then. They do have nonetheless the same editor - Ian HICKSON of Google, Inc. - in both groups to keep a proper continuity between both versions of the spec [131, 148].

The work in both groups is not exactly the same. The W3C decided to develop some features included in the WHATWG HTML spec as standalone projects, or to support them in future versions of HTML (and not HTML5, the most notable postponed feature being the device element). If the core of HTML5 is nearly the same in both groups, they do have different timelines, and different ambitions. The WHATWG is for *e.g.* working on a larger HTML spec, which the actual Working Draft includes experimental features, and goes beyond HTML5.

Fig. 3.2 helps to understand where the features are specified among the documents available in both sides.

Features	WHATWG Documents	W3C Documents
<i>HTML5 Only</i>	In WHATWG HTML & Web Applications 1.0	Main Spec (HTML WG)
<i>Microdata</i>		S.A. Spec (HTML WG)
<i>2D Context</i>		S.A. Spec (HTML WG)
<i>Cross-document Messaging</i>		S.A. Spec (HTML WG)
<i>Channel Messaging</i>		S.A. Spec (HTML WG)
<i>Device Element</i>	S.A. Spec & In Web Applications 1.0	S.A. Spec (HTML WG)
<i>Web Workers</i>		S.A. Spec (WebApps WG)
<i>Web Storage</i>	In Web Applications 1.0	S.A. Spec (WebApps WG)
<i>Web Sockets API</i>		S.A. Spec (WebApps WG)
<i>Web Sockets Protocol</i>		S.A. Spec (IETF)
<i>Server-Sent Events</i>		S.A. Spec (WebApps WG)
<i>Web SQL Database</i>	-	S.A. Spec (WebApps WG)

Figure 3.2 - *The definition of the features among WHATWG and W3C documents. S.A. Spec stands for standalone specification, and WebApps WG refers to the W3C Working Group taking care focusing on web applications specs. Modified version from the WHATWG FAQ [147].*

HTML5 features

From now on the rest of this section will have the W3C point-of-view in mind. It should also be noted again that HTML5 is a work in progress, so differences with the present Working Draft may arise before it becomes a final Recommendation. These remarks being said, HTML5 can be proved interesting in the richer web applications area. Most notably with the section 5.6, 6 and 7 of the Working Draft defining respectively offline Web apps, Web apps APIs and user interaction specifications. Hereafter are selected features and changes focusing on or impacting web applications:

- No presentational markup anymore. Elements such as `font` or `big`, and attributes like `align` or `size` are no longer authorized in HTML5. This follows the trend established by HTML4 which despised presentational markup within the document, and favoring instead the use of CSS stylesheets [131].
- HTML5 introduces several new elements for the content's structure: `section`, `article`, `aside`, `hgroup`, `header`, `footer`, `nav`, `figure` and `figcaption`. Other new elements include also [131, 132]:
 - `video` and `audio` elements to render rich media content. This improvement was highly publicized when YouTube demon-

strated a part of its videos available directly with HTML5, without using Flash Player.

- embed for plugin content.
- canvas to render graphs, games and other 2D visuals on the fly.
- new values for the type attribute of the input element: tel, search, url, email, datetime, date, month, week, time, datetime-local, number, range and color. They provide predefined data fields (or forms) that can be used with user interfaces (a date picker for *e.g.*), and offers a way to check the data before it is sent server-side. The point is to improve user experience and reduce waiting time [131, 132]. These can also be used to change the virtual keyboard according to the context: an @ or the numerical keyboard can be added when the email field or the tel field has to be filled for *e.g.*
- A set of new APIs to help web applications development [132]. These APIs aim to provide ways to build a better user experience on the Web:
 - A drag & drop API, used with the draggable attribute. This feature can let the user drag and drop files within the browser and from the browser to the desktop (and vice-versa) for *e.g.*
 - An API for video and audio content that can be used with the video and audio elements. The API provides controls such play() and pause() to manipulate the multimedia content for *e.g.* It means that no plugins such as Flash Player or Quicktime Player are required to run the multimedia content.
 - An API for content editing.
 - An API for undo history. Undo history is regularly used in desktop applications to undo errors. It has to be differentiated from navigational history, commonly represented by the back button on the browser [70].
 - An API for offline Web applications. The objective being to allow users to use Web applications without an Internet connection. The browser is able to download the necessary files (designated in a manifest) locally to keep the Web app running without a connection.
 - An API to enable websites or webapps to register themselves as possible handler for specific protocols or content. For *e.g.* if a website registers itself as an handler for a type of content that

is not supported by any desktop application, the browser could then advice to use the website to open that content.

Other specs related to HTML5

Aside from the *stricto sensus* HTML5, the W3C defines other specifications (*cf.* Fig. 3.2 to get an overall picture) that can be determinant to build more powerful Web application. Even if they are not exactly part of the HTML spec, they are often associated to HTML5 as they were included in the WHATWG Web Applications 1.0 document. That is why we are discussing them in this section.

- The 2D Canvas Context providing a way to work with the canvas element. The drawing API helps carve canvas elements and it features basic shapes, complex shapes (paths), lines, strokes, shadows, gradients, fillings, pixel manipulation, and text rendering among others [130].
- The Web Storage specification explains two ways to store structured data on the client: session storage and local storage. In both cases, these methods can do more than the actual and limited cookies [140].
 - Session storage is limited in life by the opened window. The data is accessible from any page from the website if it is still in the window. If more than one window from the same website is open, than every session (*i.e.* window) will have it is own storage space without interfering with the other sessions.
 - Local storage goes beyond the current session, and provides storage for more than one window simultaneously. Each website will have its own storage space, and it can persist after the browser is closed. That kind of storage can help to increase performances as large quantity of data can be stored locally.
- The Web Worker spec “[...] defines an API for running scripts [called workers] in the background independently of any user interface scripts” [141]. This provides a way to execute background tasks without interrupting the user interface and *vice versa*. The workers are typically used to crunch data client-side or to update a database stored locally for *e.g.*
- The WebSocket API spec allows to use the WebSocket protocol (standard developed by the IETF) for bi-directional communications between web-pages (and consequently webapps) and remote servers [138].

- The Web Messaging spec describes 2 ways to communicate between browser contexts: cross-document messaging and channel messaging [133,149].
 - Cross-document messaging refers to documents from different source domains communicating with each other. Usually this is a feature not allowed for security issues, however the spec explains a way to do it safely.
 - Channel messaging allows different browser contexts (from window to window or window to another inner-window context for *e.g.*) to communicate with each other.
- The Server-Sent Events spec “[...] defines an API for opening an HTTP connection for receiving push notifications from a server in the form of DOM events. The API is designed to be extendable to work with other push notification schemes such as Push SMS” [136].
- Web SQL Database is a spec defining a set of APIs to create and handle databases located on the client with a SQL variant [139]. This can be useful to store data locally to resolve performance issues and for offline uses [58].

All these specifications, still in the Working Draft stage at the W3C, aim to complete HTML5 to build enhanced Web applications. The objective is to standardize these methods to make an efficient Web available for everybody, while answering some issues and limitations of the actual foundations of the Web.

What about Geolocation, SVG, MathML and XHR?

Other features have been affiliated (often by the press) to HTML5, but that are not even discussed in the Web Applications 1.0 specification document. It is the case for the W3C Geolocation API. The latter is used to attempt accessing geographical location information concerning the used device. Sources include user input, GPS, WiFi and Bluetooth MAC addresses, IP address and GSM/CDMA cell IDs.

SVG is also a key technology - however mis-associated to HTML5 - for the Web future. It can be seen as a vector-based alternative to the canvas and its 2D API. SVG can produce 2D still, animated and interactive graphics with a language deviated from XML. SVG Tiny 1.2 is the latest W3C Recommendation since its development in 1999. SVG Full 1.2 is currently a Working Draft at the W3C.

The Mathematical Markup Language (MathML) is another Recommendation from the W3C, which aims to handle properly scientific content and mathematics on the World Wide Web, the same way HTML handles text on the Web [134].

As seen in Chapter 2, Section 2.1, the XMLHttpRequest (XHR) object plays a key role in asynchronous communications between the client and the server. It enables sending and receiving in the background XML data that can affect the HTML document and what the user sees. The XHR is a work in progress at the W3C. Apart from the XHR spec, the W3C is working also on the XHR Level 2 spec to extend the object with new features such as cross-domain requests or progress events [142, 143].

3.4 A SHORT DISCUSSION ON WEB APPLICATIONS STANDARDS

HTML5 has lots of qualities in its pocket: native multimedia support, canvas for advances 2D rendering, Web app APIs to enable drag & drop or offline uses, and many others. And the objective of the previous section was to show a glimpse of what HTML5 can do for Web applications. And the spec defines ways to built fully-capable, real-time, web applications. Fig. 3.3 shows a good example on how far HTML5 can go in terms of features in rich web applications.

Apart from its own staff, people who are working on W3C specifications are also coming from the companies that implement them. HTML5's editor is working at Google for example.

HTML5 can however attain its full strength only if the specifications are completely supported by Web browsers. And this is a major issue for HTML5, as cross-browser consistency can be problematic, even more when it is compared to plugin-based technologies such as Flash or Silverlight. Fig. 3.4 shows how parts of HTML5 and other related specs are supported in diverse modern Web browsers:

We can see that Internet Explorer 8 does not support well latest standards. This can be problematic because developers might give up using HTML5 features to make their websites and web apps if they cannot be available on Internet Explorer, the most used web browser to this day. However IE9 will feature a broader support of HTML5 and CSS3. Browser support of standards is essential for their propagation. If 80% of web browsers cannot handle a specific technology, the developer who wants to reach as much audience as possible will probably not use that technology.

And that is a main challenge for standards and their ideals. If they are providing a good, it does not mean they are used everywhere. Today most websites are not compliant with standards [96]. Education and business problems are often cited as main causes to the lack of interest to Web standards [43, 96].

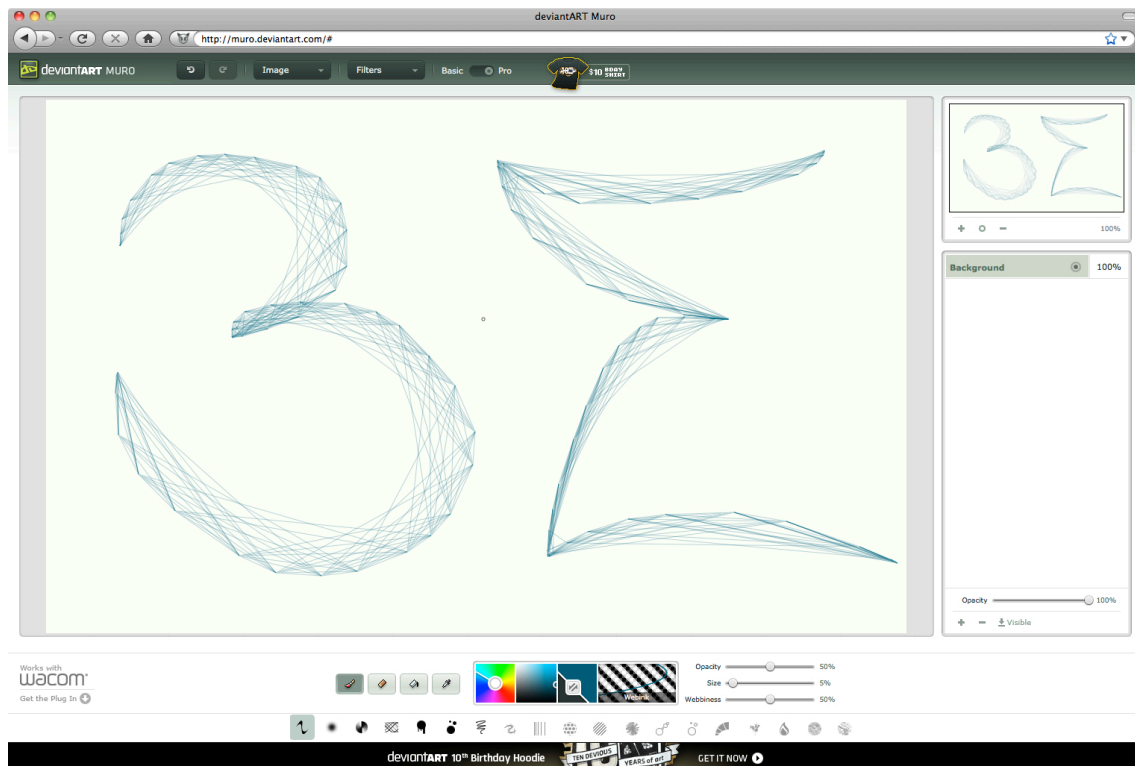


Figure 3.3 - *A screenshot of deviantART Muro. A webapp designed to produce artworks directly in the browser, without plugins. It works with any browser supporting HTML5 so far. It means it can run on any modern browser, excluding Internet Explorer 8, but including Safari Mobile on the iPad (a touch platform that could be useful for this kind of applications) [67].*

In many ways, HTML5 overcomes HTML4 limitations, and sometimes can do better than Flash. However, HTML5 is still a spec in development, meaning the document can change before it reaches its Recommendation status. Ian HICKSON, HTML5's editor, estimates that the spec could obtain its Candidate Recommendation status in 2012. Final Recommendation status is estimated for 2022 or later [147]. However, mature parts of HTML5 are already being supported by many modern Web browsers.

And even if the video tag has generated a lot of attention around the HTML5 spec, as it enables video playback without plugins, problems may however arise due to patent issues. The HTML5 document does not define a video format or a video codec that should be used in the browser. So developers have then the liberty to choose the format they deem relevant. Like its predecessor, HTML5 is file format neutral. Safari and Chrome support the H.264/MPEG-4 AVC codec¹ (MP4 container). But Mozilla, the company behind

¹ The H.264/MPEG-4 AVC codec is appreciated for its quality, compression ratio and hardware acceleration. However, the codec is subject to patents which are licensed by the MPEG LA organization.

Features	Firefox	Safari	Opera	Chrome	IE
	3.6.8	5.0.1	10.61	6.0.486.0	8
<i>Canvas Basics</i>	Yes	Yes	Yes	Yes	No
<i>Cross-document Messaging</i>	Yes	Yes	Yes	Yes	Yes
<i>Device Element</i>	No	No	No	No	No
<i>Offline Web Applications</i>	Yes	Yes	Yes	Yes	No
<i>Web Workers</i>	Yes	Yes	Yes	Yes	No
<i>Web Storage</i>	Yes	Yes	Yes	Yes	Yes
<i>Web Sockets</i>	No	Yes	No	Yes	No
<i>Server-Sent Events</i>	No	Yes	Partial	No	No
<i>Web SQL Database</i>	No	Yes	Yes	Yes	No
<i>Geolocation</i>	Yes	Yes	Yes	Yes	No
<i>Drag and drop</i>	Yes	Yes	No	Yes	Partial
<i>Forms</i>	Partial	Partial	Yes	Partial	No
<i>Video and Audio</i>	Yes	Yes	Yes	Yes	No
<i>MathML</i>	Yes	No	Partial	No	No
<i>SVG Basics</i>	Yes	Yes	Yes	Yes	No
<i>Embed</i>	Yes	Yes	Yes	Yes	Yes
<i>XHR</i>	Yes	Yes	Yes	Yes	Yes
<i>File API</i>	Yes	No	No	No	No

Figure 3.4 - A comparison of the HTML5-related features supported by modern browsers. Rework from [45, 47].

Firefox, may not want to pay the royalties needed to enable H264 videos natively within the video tag. The fact that the codec is closed does not fit Mozilla's philosophy. That is why they backed up on the Ogg Theora codec (which does not rely on known patents) and the Ogg container.

On one side, choosing HTML5 may imply to write the website or the webapp having in mind the differences of every single Web browser out there. More code may have to be built in consequence, with a different version for each end. On the other side, a SWF file can be compiled, running on every browser on which the Flash Player is installed.

Aside from consistency, Flash has features that are still being standardized by the W3C. For example, local webcams and microphones are fully supported by Flash, compared to the device tag planned for a post-HTML5 spec. That is a reason why Flash has still a role to play to build rich websites and web applications.

Regarding tech companies promoting standards closed to web applications, the industry is divided. On one hand, we have Adobe et Microsoft promoting their in-house products such Flash and Silverlight, whose technologies are not entirely opened. Flash can be seen as an “industry standard” as 98% on web-enabled computers have Flash installed out of the box. It is estimated that 75% of the video available on the web are seen on Flash.

On the other hand, we have Apple and Google promoting latest web standards such as HTML5, CSS3 and Javascript. Apple has a showcase on its website to demonstrate the power of HTML5² [56]. Furthermore, Apple uses SproutCore, a JavaScript framework heavily relying on HTML5, for its cloud applications. On the Google side, if GWT applications are written in Java, they are nonetheless compiled into standard files (HTML, CSS and JavaScript) for direct use in the browser.

And last but not least, it is interesting to conclude this chapter by noticing that the W3C or the WHATWG do not use the term “rich internet application” to describe the improvements discussed in their relative specs. They are indeed talking about simple “web applications”, even when the specs define a way to make them richer. The only time the notion of RIA is used is in the WAI-ARIA initiative to promote accessibility in richer internet application. After all RIAs are web applications. They are just supposed to be better.

* * *

² The showcase can only be seen on Safari (both on OS X and Windows), a limitation that generated some criticisms as the viewer could believe that HTML5 is only available on Safari, which is not the case. Demos from Google are also available in their own showcase [79].



END NOTE

RICH web applications and web applications in general have changed the way we use the Web. Connected, we have a direct access to information, but we have also a way to create and realize different tasks. Some of these tasks can even be part of our daily life: Remember the Milk is a good application to manage ToDo's on the Web for example. And banking applications have emerged to provide us a new medium to manage our accounts and savings (after banks' offices, faxes and phones).

The first year of the Web have been characterized with web applications having their logic located server-side, while the client had only few processing to do locally (simple UI rendering). The issue with that particular model is its dependance on the network. Lots of requests have to be done to the servers, which can saturate in case of heavy usage of the app. That is a matter RIAs can handle better as some or all the processing is done client-side, freeing the servers from requests and substantial communications. Doing so, richer web applications are able to provide us more usable, faster and better web applications, all within the browser.

But web applications are not a new concept, but it is only with the emergence of Ajax in 2005 that we can see plentiful of truly rich applications are becoming available on the Web. New technologies and new platforms have come aboard the rich internet application ship, pushing new limits in terms of interaction and user experience. Some of the applications having even integrated our desktops *via* technologies such as AIR.

We have seen there are a lot of solutions to design and build richer web applications. From lighter tools such the jQuery JavaScript library to the fully integrated platform that Flash and Flex can provide. These tools are improving on a regular basis to propose new features and new technologies to make web applications even better than before. There is still a long way to go before having the same developing power that developers can enjoy when they write native applications.

Rich internet applications give end-users and businesses alike a lot of opportunities to take advantage of the Web. The following quote illustrates well how far web applications could change the way we interact with applications:

“[...] Dr. Michael B. Johnson of Pixar gave a lunchtime presentation where he pointed out that if you don’t need 64-bit addressing, multithreading, or other desktop-only features, it makes a lot of sense to deploy apps using the web” [68].

If this quote makes a lot of sense, it has also to be nuanced. It is true that heavy-processing applications such as video-editing softwares and compositing softwares do not belong in the browser. It is too restrictive, and complex applications need in fact to access directly both software and hardware resources on the computer. But if simpler applications were great in the web browser, their native alternatives are still better. One cannot compare Google Docs to Microsoft Office. The latter stays more responsive and more capable than a suite deployed on the Web. Same thinking for Adobe Photoshop: the online version of the software cannot compete with its native version in terms of processing power and features.

The Web is an amazing platform, which will become even more important with the rise of mobile devices. These devices are seen as the next battleground for businesses alike. And web apps can be an interesting way to deploy cross-platform applications for multiple ends. With only few re-writing, a web application could be available on smartphones, tablets, game consoles, and other handheld devices along with more the more traditional desktops and laptops. But even now the trend seems to be for native applications. Apple’s App Store works incredibly well, and Android OS, Nokia and RIM have also their store for native applications.

Maybe rich web applications are not entirely ready to provide a fully engaging experience as of today. Technologies cannot compete with their native equivalent for now, but it looks like the human’s ingenuity will transcend these issues, and provide a way to develop richer web applications as good as desktop applications. Because after all, it makes sense to use applications directly on the World Wide Web.

* * *



BIBLIOGRAPHY

- [1] “Adobe Flex.” [Online]. Available: <http://www.adobe.com/products/flex/>.” Last checked in August 2010
- [2] “An animated overview of Flex.” [Online]. Available: <http://learn.adobe.com/wiki/display/Flex/Animated+Overview>.” Last checked in August 2010
- [3] “App Store.” [Online]. Available: http://en.wikipedia.org/wiki/App_Store.” Last checked in August 2010
- [4] “Architectural pattern (computer science).” [Online]. Available: [http://en.wikipedia.org/wiki/Architectural_pattern_\(computer_science\)](http://en.wikipedia.org/wiki/Architectural_pattern_(computer_science)).” Last checked in August 2010
- [5] “Basics-Design Patterns.” [Online]. Available: <http://wiki.sproutcore.com/Basics-Design+Patterns>.” Last checked in August 2010
- [6] “Basics-Introducing SproutCore MVC (SproutCore Wiki).” [Online]. Available: <http://wiki.sproutcore.com/Basics-Introducing+SproutCore+MVC>.” Last checked in August 2010
- [7] “BuildTools About (SproutCore Wiki).” [Online]. Available: <http://wiki.sproutcore.com/BuildTools-About+Build+Tools>.” Last checked in August 2010
- [8] “Curl.” [Online]. Available: <http://www.curl.com/>.” Last checked in August 2010
- [9] “DataStore About (SproutCore Wiki).” [Online]. Available: <http://wiki.sproutcore.com/DataStore-About>.” Last checked in August 2010
- [10] “Deployment Introduction (SproutCore Wiki).” [Online]. Available: <http://wiki.sproutcore.com/Deployment-Introduction>.” Last checked in August 2010
- [11] “Design pattern (computer science).” [Online]. Available: [http://en.wikipedia.org/wiki/Design_pattern_\(computer_science\)](http://en.wikipedia.org/wiki/Design_pattern_(computer_science)).” Last checked in August 2010
- [12] “Dynamic HTML and XML: The XMLHttpRequest Object,” Adobe Developer Connection. [Online]. Available: <http://developer.apple.com/internet/webcontent/xmlhttpreq.html>. Last checked in August 2010
- [13] “Food, Inc. Apple Trailers Website.” [Online]. Available: <http://trailers.apple.com/trailers/magnolia/foodinc/>.” Last checked in August 2010

- [14] "Foundation About (SproutCore Wiki)." [Online]. Available: <http://wiki.sproutcore.com/Foundation-About>." Last checked in August 2010
- [15] "Google Web Toolkit." [Online]. Available: <http://code.google.com/webtoolkit/>." Last checked in August 2010
- [16] "HTML." [Online]. Available: <http://en.wikipedia.org/wiki/HTML>." Last checked in August 2010
- [17] "HTML5." [Online]. Available: <http://en.wikipedia.org/wiki/HTML5>." Last checked in August 2010
- [18] "iAd." [Online]. Available: <http://en.wikipedia.org/wiki/Iad>." Last checked in August 2010
- [19] "Introducing SproutCore (SproutCore Wiki)." [Online]. Available: <http://wiki.sproutcore.com/Basics-Introducing+SproutCore>." Last checked in August 2010
- [20] "iOS (Apple)." [Online]. Available: [http://en.wikipedia.org/wiki/IOS_\(Apple\)](http://en.wikipedia.org/wiki/IOS_(Apple))." Last checked in August 2010
- [21] "iPhone." [Online]. Available: <http://en.wikipedia.org/wiki/IPhone>." Last checked in August 2010
- [22] "JavaFX." [Online]. Available: <http://javafx.com/>." Last checked in August 2010
- [23] "JavaFX Overview." [Online]. Available: <http://javafx.com/about/overview/>." Last checked in August 2010
- [24] "JavaScript." [Online]. Available: <http://en.wikipedia.org/wiki/JavaScript>." Last checked in August 2010
- [25] "Model-View-Controller," Apple Developer Connection. [Online]. Available: <http://developer.apple.com/mac/library/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>. Last checked in August 2010
- [26] "Model-View-Controller." [Online]. Available: <http://en.wikipedia.org/wiki/Model-view-controller>." Last checked in August 2010
- [27] "Model-View-Presenter." [Online]. Available: <http://en.wikipedia.org/wiki/Model-view-presenter>." Last checked in August 2010
- [28] "Model-View-Presenter Framework." [Online]. Available: http://www.mimuw.edu.pl/~sl/teaching/00_01/Delfin_EC/Overviews/ModelViewPresenter.htm." Last checked in August 2010
- [29] "Model-View-ViewModel." [Online]. Available: http://en.wikipedia.org/wiki/Model_View_ViewModel." Last checked in August 2010
- [30] "Multitier architecture." [Online]. Available: http://en.wikipedia.org/wiki/Multitier_architecture." Last checked in August 2010
- [31] "MVVM – Philosophy and Case Studies – Introduction." [Online]. Available: http://devlicious.com/blogs/rob_eisenberg/archive/2009/07/07/mvvm-philosophy-and-case-studies-introduction.aspx." Last checked in August 2010
- [32] "Object Oriented Design." [Online]. Available: <http://www.oodesign.com/>." Last checked in August 2010
- [33] "OpenLaszlo." [Online]. Available: <http://www.openlaszlo.org/>." Last checked in August 2010

- [34] "Remember The Milk: Online to do list and task management." [Online]. Available: <http://www.rememberthemilk.com/>." Last checked in August 2010
- [35] "Rich Internet Application Design," Adobe Developer Connection. [Online]. Available: http://www.adobe.com/devnet/flex/articles/design_patterns02.html. Last checked in August 2010
- [36] "Runtime Introduction (SproutCore Wiki)." [Online]. Available: <http://wiki.sproutcore.com/Runtime-Introduction>." Last checked in August 2010
- [37] "Scalable Vector Graphics." [Online]. Available: http://en.wikipedia.org/wiki/Scalable_Vector_Graphics." Last checked in August 2010
- [38] "Silverlight." [Online]. Available: <http://www.silverlight.net/>." Last checked in August 2010
- [39] "SproutCore." [Online]. Available: <http://www.sproutcore.com/>." Last checked in August 2010
- [40] "The Model-View-Controller Design Pattern," Adobe Developer Connection. [Online]. Available: http://www.adobe.com/devnet/flash/articles/mv_controller.html. Last checked in August 2010
- [41] "UnitTesting About (SproutCore Wiki)." [Online]. Available: <http://wiki.sproutcore.com/UnitTesting-About+Unit+Testing>." Last checked in August 2010
- [42] "Views About (SproutCore Wiki)." [Online]. Available: <http://wiki.sproutcore.com/Views-About>." Last checked in August 2010
- [43] "WaSP: Fighting for Standards." [Online]. Available: <http://www.webstandards.org/about/mission/>." Last checked in August 2010
- [44] "Web 2.0." [Online]. Available: http://en.wikipedia.org/wiki/Web_2.0." Last checked in August 2010
- [45] "Web Design Checklist." [Online]. Available: <http://www.findmebyip.com/litmus/#target-selector>." Last checked in August 2010
- [46] "Web Storage." [Online]. Available: http://en.wikipedia.org/wiki/Web_Storage." Last checked in August 2010
- [47] "When Can I Use?" [Online]. Available: <http://caniuse.com/>." Last checked in August 2010
- [48] "Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/wiki/Main_Page." Last checked in August 2010
- [49] "XMLHttpRequest." [Online]. Available: <http://en.wikipedia.org/wiki/XMLHttpRequest>." Last checked in August 2010
- [50] "Curl – Building RIA Beyond AJAX," White Paper, 2006. [Online]. Available: <http://www.curl.com/products/whitepaper/curl-ria.pdf>. Last checked in August 2010
- [51] "iPhone to Support Third-Party Web 2.0 Applications," Apple Inc., 11 June 2007. [Online]. Available: <http://www.apple.com/pr/library/2007/06/11iphone.html>. Last checked in August 2010
- [52] M. ABERNETHY, "working with jquery, part 2: Building tomorrow's web applications today," IBM, 23 September 2008. [Online]. Available: <http://www.ibm.com/developerworks/web/library/wa-jquery2/index.html>. Last checked

- in August 2010
- [53] J. ALLAIRE, "Macromedia Flash MX - A next-generation rich client," White Paper, Macromedia, March 2002. [Online]. Available: <http://www.adobe.com/devnet/flash/whitepapers/richclient.pdf>. Last checked in August 2010
 - [54] C. ANDERSON's, "The Long Tail," 1 October 2005. [Online]. Available: http://www.thelongtail.com/the_long_tail/2005/10/web_20_and_the_.html. Last checked in August 2010
 - [55] D. ANDERSSON, "HTML5, XHTML2, and the Future of the Web," 10 April 2007. [Online]. Available: http://www.digital-web.com/articles/html5_xhtml2_and_the_future_of_the_web/. Last checked in August 2010
 - [56] APPLE, "HTML5 and web standards." [Online]. Available: <http://www.apple.com/html5/>. Last checked in August 2010
 - [57] —, "iPhone Support - Wifi." [Online]. Available: <http://www.apple.com/support/iphone/wifi/>. Last checked in August 2010
 - [58] A. AVRAM, "Chrome 4 Now Supports the HTML5 Web SQL Database API," InfoQ, 18 February 2010. [Online]. Available: <http://www.infoq.com/news/2010/02/Web-SQL-Database>. Last checked in August 2010
 - [59] S. BECKER, "Sproutcore - a next-gen javascript framework." [Online]. Available: <http://www.slideshare.net/joydivider/sproutcore-a-next-gen-javascript-framework>. Last checked in August 2010
 - [60] T. BERNERS-LEE, "Realising the full potential of the web," W3C, 3 December 1997. [Online]. Available: <http://www.w3.org/1998/02/Potential.html>. Last checked in August 2010
 - [61] —, "The World Wide Web: A very short personal history," 1998 7 May. [Online]. Available: <http://www.w3.org/People/Berners-Lee/ShortHistory.html>. Last checked in August 2010
 - [62] BIVOLINO, "Customized Shirts on the Web." [Online]. Available: <http://www.bivolino.com/en/default.html>. Last checked in August 2010
 - [63] BONG.TV, "Bong.TV." [Online]. Available: <http://www.bong.tv/>
 - [64] CURL, "Curl Demo - Wealth Calculator." [Online]. Available: <http://www.curl.com/demos/wealthcalculator/start.curl>. Last checked in August 2010
 - [65] —, "Product data sheet." [Online]. Available: <http://www.curl.com/pdf/Curl-DataSheet.pdf>. Last checked in August 2010
 - [66] —, "RIA for the Enterprise." [Online]. Available: <http://www.curl.com/pdf/Curl-CorpBrochure.pdf>. Last checked in August 2010
 - [67] DEVIANTART, "deviantART Muro." [Online]. Available: <http://muro.deviantart.com/>. Last checked in August 2010
 - [68] D. E. DILGER, "Cocoa for Windows + Flash Killer = SproutCore," 14 June 2008. [Online]. Available: <http://www.roughlydrafted.com/2008/06/14/cocoa-for-windows-flash-killer-sproutcore/>. Last checked in August 2010
 - [69] FLEX DEVELOPER CENTER, "Understanding Flex in the client/server model." [Online]. Available: http://www.adobe.com/devnet/flex/articles/fcf_flex_client_server.html. Last checked in August 2010

- [70] P. Fox, "Flex vs. HTML5," 2009. [Online]. Available: <http://www.scribd.com/doc/20822331/Flex-vs-HTML5>. Last checked in August 2010
- [71] M. N. FRANCIS, "The Basics of HTML," Opera, 8 July 2008. [Online]. Available: <http://dev.opera.com/articles/view/12-the-basics-of-html/>. Last checked in August 2010
- [72] —, "The history of the Internet and the Web, and the evolution of Web standards," Opera, 2 July 2008. [Online]. Available: <http://dev.opera.com/articles/view/2-the-history-of-the-internet-and-the-w/>. Last checked in August 2010
- [73] M. V. V. GADGE, "Technology options for rich internet applications," IBM, 25 July 2006. [Online]. Available: <http://www.ibm.com/developerworks/web/library/wa-richiapp/>. Last checked in August 2010
- [74] J. J. GARRETT, "Ajax: A New Approach to Web Applications," 18 February 2005. [Online]. Available: <http://www.adaptivepath.com/ideas/essays/archives/000385.php>. Last checked in August 2010
- [75] S. GLEDHILL, "Corporate Web Standards," July 16 2007. [Online]. Available: http://www.digital-web.com/articles/corporate_web_standards/. Last checked in August 2010
- [76] GOOGLE, "Google Maps." [Online]. Available: <http://maps.google.com/>. Last checked in August 2010
- [77] —, "GWT Overview." [Online]. Available: <http://code.google.com/webtoolkit/overview.html>. Last checked in August 2010
- [78] —, "HTML5 Presentation." [Online]. Available: <http://slides.html5rocks.com/#slide1>. Last checked in August 2010
- [79] —, "HTML5Rocks." [Online]. Available: <http://www.html5rocks.com/>. Last checked in August 2010
- [80] —, "Making GWT Better." [Online]. Available: <http://code.google.com/webtoolkit/makinggwtbetter.html>. Last checked in August 2010
- [81] P. GRAHAM, "Apple is not evil. iPhone developers are stupid." [Online]. Available: http://www.quirksmode.org/blog/archives/2009/11/apple_is_not_ev.html. Last checked in August 2010
- [82] —, "Web 2.0," November 2005. [Online]. Available: <http://www.paulgraham.com/web20.html>. Last checked in August 2010
- [83] J. GRUBER, "iPhone Web Apps as an Alternative to the App Store," 23 November 2009. [Online]. Available: http://daringfireball.net/2009/11/iphone_web_apps_alternative. Last checked in August 2010
- [84] C. HEILMANN, "CSS basics," Opera, 26 September 2008. [Online]. Available: <http://dev.opera.com/articles/view/27-css-basics/>. Last checked in August 2010
- [85] M. HOLZSCHLAG and S. KAISER, "FAQ - The Web Standards Project," WaSP, 27 February 2002. [Online]. Available: <http://www.webstandards.org/learn/faq/>. Last checked in August 2010
- [86] IBM, "Rich Internet Applications (RIAs)." [Online]. Available: <http://www-01.ibm.com/software/info/web20/mashups-rias/ria.html>. Last checked in August 2010

- [87] —, “Putting the power of Web 2.0 into practice,” White Paper, July 2008. [Online]. Available: <ftp://ftp.software.ibm.com/software/lotus/lotusweb/product/expeditor/LOW14003-USEN-00.pdf>. Last checked in August 2010
- [88] INFOBEANS, “iPhone Native Apps vs Web Apps,” June 2009. [Online]. Available: <http://www.slideshare.net/infobeans/iphone-web-and-native-apps-comparison>. Last checked in August 2010
- [89] INFOTECH RESEARCH GROUP, “RIA - What’s the Business Case,” Research Note, 3 December 2008. [Online]. Available: <http://www.infotech.com/research/rich-internet-applications-whats-the-business-case>. Last checked in August 2010
- [90] S. JOBS, “Thoughts on Flash,” Apple Inc. [Online]. Available: <http://www.apple.com/hotnews/thoughts-on-flash/>. Last checked in August 2010
- [91] JQUERY, “The Write Less, Do More, JavaScript Library.” [Online]. Available: <http://jquery.com/>. Last checked in August 2010
- [92] JQUERY UI, “Animate Effect.” [Online]. Available: <http://jqueryui.com/demos/animate/>. Last checked in August 2010
- [93] C. KAMBALYAL, “3-Tier Architecture.” [Online]. Available: <http://channukambalyal.tripod.com/NTierArchitecture.pdf>
- [94] KAYAK, “Kayak homepage.” [Online]. Available: <http://www.kayak.com/>. Last checked in August 2010
- [95] J. LANE, “The Web standards model - HTML, CSS and JavaScript,” Opera, 8 July 2008. [Online]. Available: <http://dev.opera.com/articles/view/4-the-web-standards-model-html-css-a/>. Last checked in August 2010
- [96] —, “Web standards – beautiful dream, but what’s the reality?” Opera, 8 July 2008. [Online]. Available: <http://dev.opera.com/articles/view/5-web-standards-beautiful-dream-bu/>. Last checked in August 2010
- [97] S. LANINGHAM, “developerWorks Interviews: Tim Berners-Lee,” 22 August 2006. [Online]. Available: <http://www.ibm.com/developerworks/podcast/dwi/cm-int082206txt.html>. Last checked in August 2010
- [98] MICROSOFT SILVERLIGHT, “Search Engine Optimization for Silverlight applications.” [Online]. Available: <http://www.silverlight.net/learn/whitepapers/seo-for-silverlight/>. Last checked in August 2010
- [99] —, “Silverlight Architecture.” [Online]. Available: [http://msdn.microsoft.com/en-us/library/bb404713\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/bb404713(VS.95).aspx). Last checked in August 2010
- [100] —, “Silverlight Overview.” [Online]. Available: <http://www.silverlight.net/getstarted/overview.aspx>. Last checked in August 2010
- [101] C. MILLS, “Introduction to The Web Standards Curriculum,” Opera, 8 July 2008. [Online]. Available: <http://dev.opera.com/articles/view/1-introduction-to-the-web-standards-cur/>. Last checked in August 2010
- [102] T. NEIL, “12 Standard Screen Patterns.” [Online]. Available: <http://designingwebinterfaces.com/designing-web-interfaces-12-screen-patterns>. Last checked in August 2010
- [103] —, “15 Common Components.” [Online]. Available: <http://designingwebinterfaces.com/15-common-components>. Last checked in August 2010

- 2010
- [104] —, “30 Essential Controls.” [Online]. Available: http://designingwebinterfaces.com/essential_controls.” Last checked in August 2010
 - [105] —, “Designing rich applications.” [Online]. Available: <http://www.slideshare.net/theresaneil/designing-rich-applications>.” Last checked in August 2010
 - [106] T. NODA and S. HELWIG, “Technical Comparison and Case Studies of AJAX, Flash, and Java based RIA,” UW E-Business Consortium, 16 November 2005. [Online]. Available: <http://www.uwebc.org/opinionpapers/archives/docs/RIA.pdf>. Last checked in August 2010
 - [107] F. NONNENMACHER, “Web standards for business,” The Web Standards Project, November 2003. [Online]. Available: http://www.webstandards.org/learn/articles/web_standards_for_business/. Last checked in August 2010
 - [108] OPENLASZLO, “Chapter 1. OpenLaszlo Architecture (OL Documentation).” [Online]. Available: <http://www.openlaszlo.org/lps4.8/docs/developers/architecture.html>.” Last checked in August 2010
 - [109] —, “Chapter 2. Language Preliminaries (OL Documentation).” [Online]. Available: <http://www.openlaszlo.org/lps4.8/docs/developers/language-preliminaries.html>.” Last checked in August 2010
 - [110] —, “Chapter 25. Proxied and SOLO Applications (OL Documentation).” [Online]. Available: <http://www.openlaszlo.org/lps4.8/docs/developers/proxied.html>.” Last checked in August 2010
 - [111] —, “Chapter 42. Audio and Video (OL Documentation).” [Online]. Available: <http://www.openlaszlo.org/lps4.8/docs/developers/video.html>.” Last checked in August 2010
 - [112] —, “OpenLaszlo - An Open Architecture Framework for Advanced Ajax Applications,” White Paper, November 2006. [Online]. Available: <http://www.openlaszlo.org/whitepaper/LaszloWhitePaper.pdf>. Last checked in August 2010
 - [113] T. O'REILLY, “What is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software,” 30 December 2005. [Online]. Available: <http://oreilly.com/web2/archive/what-is-web-20.html>. Last checked in August 2010
 - [114] —, “Web 2.0 and Cloud Computing,” 26 October 2008. [Online]. Available: <http://radar.oreilly.com/2008/10/web-20-and-cloud-computing.html>. Last checked in August 2010
 - [115] T. O'REILLY and J. BATTELLE, “Web Squared: Web 2.0 Five Years On,” Special Report, October 2009. [Online]. Available: http://assets.en.oreilly.com/1/event/28/web2009_websquared-whitepaper.pdf. Last checked in August 2010
 - [116] J. PETRSEN, “Benefits of using the n-tier approach for Web applications,” Adobe. [Online]. Available: <http://www.adobe.com/devnet/coldfusion/articles/ntier.html>. Last checked in August 2010
 - [117] C. RAMSDALE, “Large scale application development and MVP,” Google Developer Relations. [Online]. Available: <http://code.google.com/intl/fr/webtoolkit/articles/>

- [mvp-architecture.html](#). Last checked in August 2010
- [118] B. SMEETS, U. BONESS, and R. BANKRAS, *Beginning Google Web Toolkit: From Novice to Professional*. Apress, 2008.
- [119] SPROUTCORE, “Sample Controls.” [Online]. Available: http://demo.sproutcore.com/sample_controls/. Last checked in August 2010
- [120] M. STACHOWIAK, “Proposal to Adopt HTML5,” 9 April 2007. [Online]. Available: <http://lists.w3.org/Archives/Public/public-html/2007Apr/0429.html>. Last checked in August 2010
- [121] A. STAMOS and Z. LACKEY, “Attacking AJAX Web Applications,” iSEC Partners, 3 August 2006. [Online]. Available: https://www.isecpartners.com/files/iSEC-Attacking_AJAX_Applications.BH2006.pdf. Last checked in August 2010
- [122] STATOWL, “Rich Internet Application Market Share.” [Online]. Available: http://www.statowl.com/custom_ria_market_penetration.php. Last checked in August 2010
- [123] M. ŞUCAN, “SVG or Canvas? Choosing between the two,” ROBO Design. [Online]. Available: <http://www.robodesign.ro/coding/svg-or-canvas/>. Last checked in August 2010
- [124] B. SUDA, “Introduction to the mobile Web,” Opera, 23 June 2009. [Online]. Available: <http://dev.opera.com/articles/view/introduction-to-the-mobile-web/>. Last checked in August 2010
- [125] A. ŠUŠNJAR, “RIA and AJAX,” December 2006. [Online]. Available: http://en.wikipedia.org/wiki/User:Aleksandar_Šušnjar/RIA_and_AJAX. Last checked in August 2010
- [126] R. VALDES, E. KNIPP, D. M. SMITH, G. PHIFER, and M. DRIVER, “MarketScope for Ajax Technologies and Rich Internet Application Platforms,” Gartner RAS Core Research, 31 December 2009. [Online]. Available: <http://www.adobe.com/enterprise/pdfs/gartner-ajax-ria.pdf>. Last checked in August 2010
- [127] W3C, “Document object model.” [Online]. Available: <http://www.w3.org/DOM/>. Last checked in August 2010
- [128] —, “History of HTML.” [Online]. Available: <http://www.w3.org/html/wg/wiki/History>. Last checked in August 2010
- [129] —, “HTML & CSS.” [Online]. Available: <http://www.w3.org/standards/webdesign/htmlcss>. Last checked in August 2010
- [130] —, “HTML Canvas 2D Context.” [Online]. Available: <http://dev.w3.org/html5/2dcontext/Overview.html>. Last checked in August 2010
- [131] —, “HTML5 - A vocabulary and associated APIs for HTML and XHTML.” [Online]. Available: <http://dev.w3.org/html5/spec/Overview.html>. Last checked in August 2010
- [132] —, “HTML5 differences from HTML4.” [Online]. Available: <http://dev.w3.org/html5/html4-differences/>. Last checked in August 2010
- [133] —, “HTML5 Web Messaging.” [Online]. Available: <http://dev.w3.org/html5/postmsg/Overview.html>. Last checked in August 2010
- [134] —, “MathML.” [Online]. Available: <http://www.w3.org/Math/>

- [whatIsMathML.html](#)." Last checked in August 2010
- [135] —, "Scripting & Ajax." [Online]. Available: <http://www.w3.org/standards/webdesign/script>." Last checked in August 2010
- [136] —, "Server-Sent Events." [Online]. Available: <http://dev.w3.org/html5/eventsource/>." Last checked in August 2010
- [137] —, "The Mobile Web." [Online]. Available: <http://www.w3.org/2007/Talks/0222-3gsm-tbl/>." Last checked in August 2010
- [138] —, "The WebSocket API." [Online]. Available: <http://dev.w3.org/html5/websockets/>." Last checked in August 2010
- [139] —, "Web SQL Database." [Online]. Available: <http://dev.w3.org/html5/webdatabase/>." Last checked in August 2010
- [140] —, "Web Storage." [Online]. Available: <http://dev.w3.org/html5/webstorage/>." Last checked in August 2010
- [141] —, "Web Workers." [Online]. Available: <http://dev.w3.org/html5/workers/Overview.html>." Last checked in August 2010
- [142] —, "XMLHttpRequest." [Online]. Available: <http://dev.w3.org/2006/webapi/XMLHttpRequest/>." Last checked in August 2010
- [143] —, "XMLHttpRequest Level 2." [Online]. Available: <http://dev.w3.org/2006/webapi/XMLHttpRequest-2/>." Last checked in August 2010
- [144] —, "W3C Publishes HTML5 Draft, Future of Web Content," 22 January 2008. [Online]. Available: <http://www.w3.org/2008/02/html5-release>. Last checked in August 2010
- [145] W3SCHOOLS, "Flash Embedded in HTML." [Online]. Available: http://www.w3schools.com/flash/flash_inhtml.asp." Last checked in August 2010
- [146] T. V. WAL, "Folksonomy," February 2 2007. [Online]. Available: <http://www.vanderwal.net/folksonomy.html>. Last checked in August 2010
- [147] WHATWG, "FAQ." [Online]. Available: <http://wiki.whatwg.org/wiki/FAQ>." Last checked in August 2010
- [148] —, "HTML5 (including next generation additions still in development)." [Online]. Available: <http://www.whatwg.org/specs/web-apps/current-work/>." Last checked in August 2010
- [149] —, "Web Applications 1.0." [Online]. Available: <http://www.whatwg.org/specs/web-apps/current-work/complete.html>." Last checked in August 2010
- [150] —, "WHATWG Specifications." [Online]. Available: <http://www.whatwg.org/specs/>." Last checked in August 2010
- [151] B. WHITE, "The Implications of Web 2.0 on Web Information Systems," in *Web Information Systems and Technologies*. Springer Berlin Heidelberg, 2007, vol. 1, pp. 3–7.