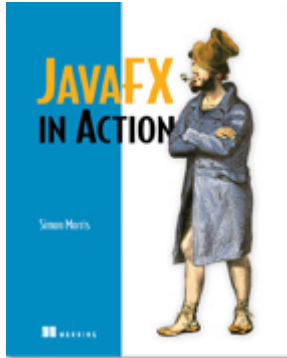


# Why do we need JavaFX?

Excerpted from



## JavaFX in Action

EARLY ACCESS EDITION

Simon Morris

MEAP Release: August 2008

Softbound print: May 2009 (est.) | 375 pages

ISBN: 1933988991

*This article is based on chapter 1 from **JavaFX in Action** by Simon Morris.*

A very good question – why **do** we need yet another language? The World is full of programming language, wouldn't one of the existing language do? Perhaps JavaScript, or Python, or what about Scala? Indeed, what's wrong with Java?

Certainly JavaFX makes writing slick graphical applications easier, but is there more to it than that?

In a broader sense JavaFX attempts to respond to a change in the eco-system of user facing software over the last twenty years, and particularly since the mid to late 1990s when the World Wide Web began to rise in prominence. To understand why JavaFX is the answer we would be well served to spell out precisely what the question is. Technology moves so quickly and in so many different directions these days that it is vital we understand the exact scope of the problem JFX is attempting to solve. So, let us first consider what makes the JavaFX Script language special for graphics programming, then provide a little context by examining the shifting trends in the software market since the arrival of the World Wide Web, before finally rounding off by seeing how JavaFX addresses the problems of this new environment.

## *If I had a hammer*

*"When the only tool you have is a hammer, every problem looks like a nail", the popular saying goes.*

Language advocacy is a popular pastime with many programmers, but what many fail to realize, be they 'fanboys' of Java or C# or Python or COBOL (do we have any octogenarian programmers reading?) is that programming languages are like tools: each is good at some things, and next to useless at others.

Java, inspired as it was on prior art like C and Smalltalk, sports a solid general purpose syntax which gets the job done with the minimum of fuss in the majority of cases. Unfortunately there will always be those areas which, by their very nature, demand something a little more specialized. Graphics programming has typically been one such area, with pretty much every popular language developed over the decades struggling to provide anything more than stodgy, cumbersome, support.

What makes graphics programming such an ill fit for modern programming languages? I'm sure if we asked a dozen experts we'd get thirteen opinions, but let me (your humble author) risk having a stab at defining a couple of likely candidate problems.

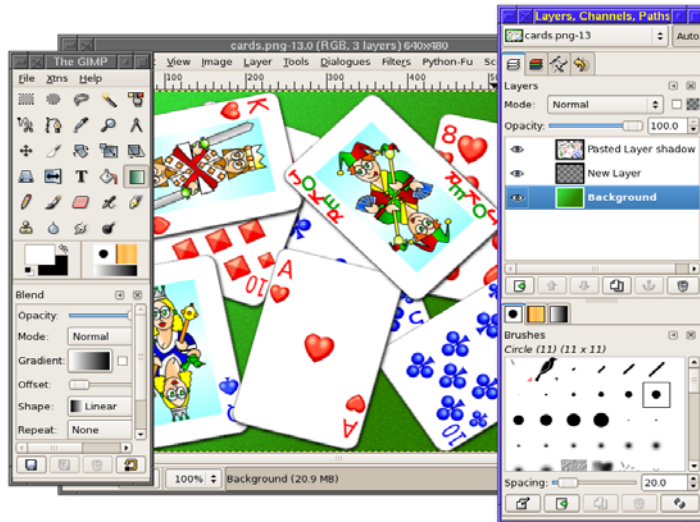


Figure 1.1 A complex GUI typical of modern desktop applications. Two windows host scrolling control palettes, while another holds an editable image and rulers.

Firstly, graphics generally require quite large nested data structures: trees of elements, each element providing a baffling array of configurable options and behaviors. Procedural languages like to work in clearly delineated steps, but this linear pattern conflicts with the tree pattern inherent in most GUIs.

Secondly, graphics code tends to rely heavily on concurrency – processes running in parallel. Modern UI fashions have amplified this requirement, with several transition effects often running within a single interface simultaneously. Unfortunately the heavy boilerplate code demanded by many languages to spawn and manage new threads makes these kind of animations cumbersome to construct.

Perhaps you can think of other problems, but the above two (at least in my experience) seem to create more than enough trouble between them.

We've only considered part of the story – JavaFX isn't just about slick visuals, it's also an important weapon in the arms race for the Rich Internet Application market. But what is a Rich Internet Application, and why is it important? To answer those questions we need to take a brief trip back in time.

## *Back to the future*

Douglas Adams once wrote "*I suppose the best way to find out where you come from is to find out where you're going, and then work backwards.*"

The internet has become such a dominant cultural force, our grandchildren will likely never believe us when we regale them with exotic tales of a life without a permanent broadband connection. "*But how did your computers ever work, Grandpa?*"

In the pre-internet age software was installed straight onto the hard drive. If, at your local bar, you were overcome by a sudden urge to share with a friend your latest poetic masterpiece, you needed to hand both the poem file itself and the software to open it. Chances are neither would be available.

Although your friend might be eternally grateful for this fact, clearly it was a problem which needed a solution.

And a solution began to take form with the sudden explosion into the popular consciousness of an exciting new technology with the remarkable bizarre name of "The World Wide Web". Initially allowing no interactivity between user and server beyond the clicking of a hyper-link, the web rapidly evolved to accommodate the sending of basic form data with each page request. This meant pages were no longer static, but could be dynamically generated in response to parameters passed in by the visitor.

Initial web applications confined themselves to humble database lookups: what time is the last bus?, who played the three wise monkeys in the movie Planet of the Apes?, which pages have the words "Pamela", "Anderson" and "pictures" on them? But bit by bit novel ways were being found to utilize the simple

query/response mechanism. It started innocently enough with the likes of web based e-mail, then eventually spread to calendars, personal photo galleries, word processor and spreadsheets. Pretty soon the web was awash with clever new applications.

This marked a fundamental shift in the relationship between site and visitor. Previously the site held content which the visitor browsed or queried; but now, with the advent of web email and the like, the site provided no content itself but relied entirely on users to populate it. The role of the site had moved from information source, to storage depot, and the role of the visitor from passive consumer to active producer.

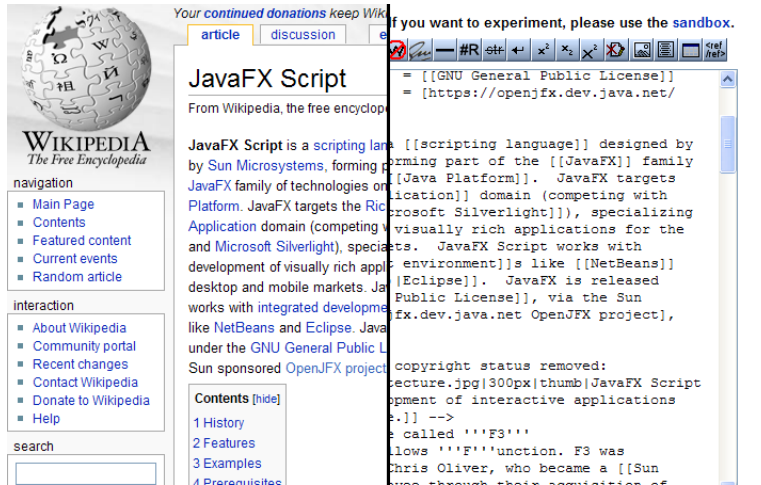


Figure 1.2 Wikipedia is an on-line encyclopedia constructed entirely by contributions from its users. The "Edit this page" link above each article allows the reader to change, in real time, the page's content.

A new meme was emerging: the concept of User Generated Content. Sites would be populated by the users themselves, millions of contributors pooling their resources to create information resources on a spectacular scale. The poster child for this revolution was (and still is, to a large extent) Wikipedia, the on-line encyclopedia in which all of the content is provided by the readers.

Don't think the entry on "The Mighty Boosh" is up to scratch? Just click "EDIT" and change it yourself! No longer would big organizations be able to shape and control information – the inmates were taking over the asylum!

Some were horrified by the prospect – wouldn't this just create a jumble of untrustworthy information from millions of unqualified contributors? – while others began to revel at what they saw as a true democratization of information. Sometime around 2004 publisher and Open Source advocate Tim O'Reilly coined the term "*Web 2.0*" to address the phenomena, and it quickly fell into use (and abuse) as a hot new buzz phrase.

The explosion in web based applications drove demand for more sophisticated user interfaces. This new generation of site, which attempted to ape the look and feel of traditional desktop software, became blessed with the moniker "*Rich Internet Application*" after Macromedia (subsequently purchased by Adobe) coined the term in a 2002 white paper noting the transition of applications from the desktop onto the web, and asking how they might be improved (unsurprisingly the answer involved Macromedia products.)

But despite all the enthusiasm, real progress was slow and frustrating. While the likes of Ajax helped paper over some of the cracks, at its heart the web was designed to show content, not run software. Web content is 'poured' into the window, left to right down the page echoing the technology's publishing origins, while input is predominantly restricted to basic form components. Mimicking the look and functionality of a powerful application-centric technology inside a document-centric environment was not easy, as numerous web developers soon discovered. Yet still they tried...

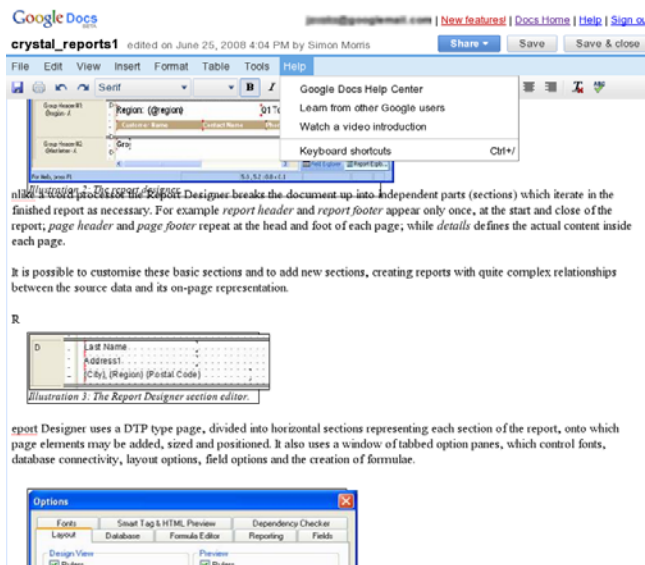


Figure 1.3 Google Docs runs inside a browser and has a much simpler GUI than Microsoft Office or OpenOffice.org, even struggling to render a simple Word file correctly.

At the bleeding edges of the software development world some programmers dared to commit heresy; they asked whether the web browser was really the best platform for creating Rich Internet Applications?

They looked back to see where they had come from, and saw a wealth of elegant desktop software with high fidelity user interfaces and sophisticated user interactivity. But this software was written using the old desktop toolkits which bound themselves firmly to their OS. Not only would software written using the Mac UI libraries fail miserably on Windows (and vice-versa) but the desktop libraries themselves were written back in an age when all software was run from a local disk – huge monolithic beasts which occupied dozens (sometimes hundreds) of megabytes of disk space.

The web had none of these problems, and that is precisely why it had flourished. Web applications were nimble, yet they lacked any capacity for sophistication.

If only there was a middle way, uniting the power of the desktop UI with the nimbleness and visual impact of the web! If only desktop applications could behave like they lived on the web! If only the web, the desktop and the mobile spaces could come together under one unified UI technology! And so the heretics began to cast about for a solution...

## *Form follows function*

From its first release in 1996 Java had featured a powerful technology for deploying rich applications within a web page. So called *Java Applets* could be placed on any page, just like an image, and ran inside a secure environment which prevented unauthorized tampering with the underlying computer or operating system. While applets boosted the visibility of the Java brand, the idea initially met with mixed success. The applet was a hard-core programming technology in a world dominated by artists and designers, and while many page authors drooled over Java's power, few understood how to install one onto their own site, let alone how to create an applet from scratch.



Figure 1.4 An applet (the game 3D-Blox) runs inside a web page, living along side other web content like text and images.

Java's main rival in the applet space was Macromedia Flash, an animation and presentation tool which boasted a considerably more designer-friendly development experience. Once Macromedia's plugin entered the ring the writing was on the wall for Java applets, and they were quickly defeated by a knockout to leave Flash as the undisputed World (Wide Web) champion. But by then focus of the Java community was already elsewhere; Java had secured an audience on the server side, where it powered blue-chip web sites, and in the mobile market, where it powered phone games and applications. And so applets were forgotten, receiving little or no attention in the updates which followed Java's debut release.

And this is where they might have languished, had the web not taken a collaborative path...

Ten years on (thereabouts) from the death of the applet, and the buzz was once again about on-line applications. Not Java applets this time, but the new breed of Rich Internet Applications which many saw as holding much promise. Yet RIAs were finding it hard to deliver an experience comparable to the desktop, and it wasn't immediately apparent from where a solution might come. Certainly Ajax and HTML were struggling to provide the kind of refined user interaction many now wanted, and Flash's strengths lay more in animation than solid, traditional, 'functional' GUIs and data manipulation.

Was Java about to be given a second chance?

Java had already proved itself as an effective software platform away from the desktop, amassing many followers in the software community and a vast archive of third party libraries in the process. Yet Java still had one major handicap – as far as the desktop was concerned it remained a tool for cola swigging, black t-shirt wearing, code junkies, not trendy cappuccino sipping, goatee stroking, artists. If Java was to be the answer to the RIA dilemma it needed to be less techie and more sympathetic of UI design and aesthetics.

In 2005 Sun Microsystems had acquired a company known as SeeBeyond Technology Corporation, and in the process picked up a talented software engineer by the name of Chris Oliver. Oliver's experimental F3 programming language (Form Follows Function) sought to make GUI programming easier by designing the language around the specific needs of user interface programming. As they pondered how best to exploit the emerging Rich Internet App market, the folks at Sun could surely not have failed to note the potential of combining the existing Java platform and Chris Oliver's new graphics power tool. And so in 2007, at the JavaOne Conference (the community's most important annual gathering), F3 was given center stage as Java's beachhead into the new Rich Internet Application market.

And as if to demonstrate its importance, F3 was blessed with a sexy new name: "*JavaFX*"!



Figure 1.5 The StudioMoto demo, one of the original JavaFX examples, shows off a glossy user interface with animation, movement and rotating elements all responding to the user's interaction.

JavaFX would marry the power of the network-centric Java platform with an efficient graphics-centric language, creating the perfect environment for both programmer and designer to push their skills to the limits on the desktop, the web and on smart phones.

At least, that's the plan...