Ruby-on-Rails, PHP, and the Attractiveness of Web Languages to SAS Developers

Tom Taylor, Palladian Analysis & Consulting, LLC.
Houston, TX
ttaylor@computer.org

Abstract:

In the last 12 months a programming firestorm by the name of Ruby-on-Rails has swept to the forefront of web programming languages. In the past decade languages like Basic, Java, and PHP have also held this position. Over this period, SAS has managed to either co-op or ignore each of these developments and keep their market position in tact. However, Ruby-on-Rails is the first one I have seen that might actually change the behavior pattern of those who develop SAS or SAS like applications. In this paper I will give a point of view on the evolution of these popular languages, what makes them popular, and the likely reaction in the SAS world. This paper takes a look at the arcane world of computer language evolution.

Introduction:

During 2005 a video was made available on the web. It showed how "starting from scratch" one could have a working database driven website in the time it took to make the video or 15 minutes. To those who had previously undertaken such an endeavor, this was a startling increase in productivity. The way this revelation was accomplished with a development framework called "Rails" combined with an obscure language "Ruby" to form the mighty "Ruby on Rails". Developers had to know more.

Programming languages evolved rapidly along with the hardware evolution of the 80's and 90's, In the beginning … all the world needed were two languages, A language for the accountants, Cobol, and a language for the engineers, Fortran. Others grew from here. SAS was written in Fortran. As the world became more creative with computers, languages that would better handle complexity were added. Thus came C and C++. Along the side was a language called Basic that showed the attractiveness of interpreted rather than complied languages, more on this later. The C languages, along with relational database development, handled the world fairly well until the Web came along. Then Java, JavaScript, HTML, VBScript, PHP, Python, Perl, Ruby, and ASP(X) all jumped on the scene and commanded some following. Why the blossoming of so many tongues?

The blossoming of tongues was largely driven by two elements, the attractiveness of graphical user interfaces (GUI's) and the popularity of simple front-ends to vast and complex relational databases.

To meet this demand, SAS and many other software providers developed intricate tools that would allow infinite flexibility in the development of GUI's. If I may speak for many developers, we hated their tools. Infinite flexibility brought near infinite complexity to the screen developers' code. Changes were tedious. Code was lengthy. It was difficult to keep data entry boxes aligned. Furthermore, special client software needed to be added to each user's machine. The programmers tolerated this because the end products were fairly pretty to look at and also useful.

Then the Web Browser appeared. The browser was a breakthrough in many ways but the two of interest here were that it created a standard client and most importantly it provided a way to vastly reduce GUI coding activity, while only slightly reducing the flexibility of screen layout. It is what I like to call the 98% solution.

In addition to the lengthy coding issue for GUI's, I personally was very tired of telling clients to buy SAS licenses just so they could run screens on a local laptop. To the horror of SAS and other major software houses, 99.9% of the world felt as I did and over a short period the web browser became the de facto GUI for all except the most speed sensitive applications. Now we can return to the blossoming of tongues story. Here we look at three major evolutionary languages, Java, PHP, and Ruby_on_Rails.

Java was on the scene with the Web explosion. The now well known situation occurred where Sun Microsystems saw a chance to replace Microsoft as king of the hill by wildly promoting the Java language as the ideal solution for the web programming world. Most interestingly, they gave Java away. Programmers swooned. Java experts were going for $275 an hour. Learn Java – get rich. But Java had one big problem. In an effort to be the perfect computing solution for all situations, it like the SAS GUI tools before, had to have a lot of complexity to work. While it perfectly provided a strategy for write once and use everywhere, who wanted to write something like:

```
"(Java.Util.DateFormat.getDateInstance(DateFormat.SHORT)).format(new Date());"
```

just to format a date? Not me, and not a whole lot of others as well. But Java did bring the word 'free' front and center. It also helped that a whole development system could be downloaded in less time that it took to eat lunch.

As a note, JavaScript is not Java. The name confusion came about because Sun paid Netscape to create the confusion (a.k.a. marketing opportunity) by renaming their web client language JavaScript.

PHP: With the web providing its classic sharing of information mechanism, people started to write little languages and pass them around. Who knows how many thousand of these projects were started. But just as in the butterfly effect where the theory goes that a random butterfly flapping it wings in Africa results in a hurricane a few weeks later in Florida, a guy in Sweden wrote a little language called PHP (Personal Home Page). For some reason a couple of grad students in Israel picked up PHP and enhanced it into a straightforward tool for talking between web pages and web servers. They started passing it around and 20 million web sites later it had become a worldwide tool of choice for many. So what did PHP have over Java? First off it was interpreted, not compiled. This made development much easier by cutting out all the keystrokes of running a compiler, a program that preprocesses code to make it execute faster. Second, PHP had some good innovations of passing data back and forth to web pages. That is it was a post-Web language. The quirk of PHP was that so many people ultimately contributed to its development, it became an "everything but the kitchen sink" language. Anything was do-able in about any order of events. This lead to code that was straightforward to start writing but easily became a mess in large projects.

An aside on Visual Basic: VB and its near infinite variations have certainly done a good job of bringing innovations under the Microsoft label. Whatever I say about Ruby on Rails or PHP, rest assured that Microsoft will find a way to package the equivalent under their label. Also note that VB.net is a framework, as Rails is a framework, but .net is more general purpose.

Ruby on Rails: So to put it succinctly, people like RoR because it makes great strides towards removing the time eating hassles all system developers have experienced with the predecessor languages.

I need to remind you that Ruby is an object oriented scripting language originally created in Japan circa 1994 by one Yukihiro Matsumoto. Rails is a framework created in Chicago some ten years later by one David H. Hansson.

A framework in this case means there are strict standards for the directory organization, for naming files, directories, objects, methods, and datasets. Also in this framework the URL extensions in the address bar expect a very specific directory structure. The two are neatly tied together. These strong standards promote productivity and allow for a lot of basic structure and several beginning web pages to be created with just one or two lines of code. RoR developers are not allowed to reinvent the wheel. They have to start with the standard chassis.

The specific items that differentiate Ruby on Rails are:

1. Organization; Rails forces an organization structure that is widely known as Model-View-Controller. This structure is appropriate for most web database programming situations. This structure is helpful because when first starting a development the hardest thing to get right for the long haul is organization.
2. Interpreted (a.k.a. scripting) not compiled, language; this one item probably speeds development by a factor of two. Detractors can say the code will run slower, but the small chunks of code needed by most apps, the difference in speed is not noticeable.
3. Object oriented; Objects are mysterious to the uninitiated. Objects are just a way to keep data, and the code that knows how to handle that data together, which simplifies reuse and reduces complexity.
4. Efficient interface to the Web; Web code, HTML specifically, is very verbose. It can take a lot of typing to create. Ruby does an excellent job of cutting the typing out of this.
5. Efficient connection to the database; RoR automatically maps rows in the database to objects. It allows you to spend less time on all the CRUD (Create, Read, Update, Delete).

So why should this matter to SAS developers? In the 80's software was largely custom developed. In the 90's, GUI complexity and development expense forced companies to buy semi-packaged software and adapt it to their needs. The rise of RoR and its inherent efficiencies signals the return of custom software for companies. In the SAS world this means that SAS applications can now be more economically transformed into processes accessible by people who only know the basics.

In the presentation I will give examples of the superlative features of Ruby on Rails and how it might be used as a front end for SAS. Since 2004, I have had PHP running as production code as a SAS front end. If project time allows, I will have a Ruby example running as well. In any case it will be using SAS as the data base and the SAS procedures being accessible stored procedures.

Code Efficiency Example:

By way of example, let me summarize these benefits by reproducing the pricing on a proposal we made back it the spring. This gives our estimated time to do the same code in four different languages.

A Recent Estimate of Time for Four Development Environments

	Coding Environment	Man-days	Comments
1	C/C++	400	Typically only for research cutting edge activity.
2	Java	200	Typically used for global systems where development is by several different groups in different countries. (Not to be confused with JavaScript, a common screen control language that is used in conjunction with all of these environments) In general, use of Java is declining.
3	PHP	100	This is a common standard that has developed over the last decade, it have many speed efficiencies over Java. However, it is a bit of a kitchen sink language (has everything, with few standards) The lack of standards, makes it easy to develop, but hard to maintain over time.
4	Ruby/Rails	80	Ruby in its current form is very new and has many compelling development features that should make it the tool of choice for systems such as yours. Alternatively the Java and PHP worlds are rapidly trying to remake themselves in the more efficient Ruby on Rails form. Ruby's strict development standards and object system make it relatively quick to develop, but more importantly, make it easier to maintain and improve over time.

Conclusion:

Programming languages will continue to evolve, but for the present we believe Ruby_on_Rails as a user interface for SAS can have great economic benefit.

Web References:

Wikipedia: http://en.wikipedia.org/wiki/Ruby_on_Rails

Rails: http://www.rubyonrails.org/

Ruby: http://www.ruby-lang.org/en/

PHP: http://www.php.net/