

Thursday, June 15, 2006

### Reprogramming wheels

Clay Loveless recently posted a small rant [1] about loner applications, and that inspired me to write up a comment of sorts.

I think that the interoperability Clay wants requires out-of-the-box thinking, but sadly most of the programming world lives in tight boxes. To change that tide would require a lot of dedication and time, both of which I for one am not prepared to invest.

It is really quite sad, but the time and effort I need to get the author(s) of any piece of software to become aware of the non-interoperability with software XY I prefer to invest in getting a solution done - especially since the answers are usually "works for me", "don't need software XY", "it's open source, fix it yourself", "nobody needs that except you", "no time, maybe this winter". I usually try contacting the authors in case I should get lucky, but tend to avoid those kinds of discussions as they lead nowhere.

I think that one of the core reasons for the lack of interoperability is the reprogramming the wheel syndrome. I did that quite a few times myself, and while it may often be because the programmer feels he can do it better, I think there is more to it than just that. These are a few issues I can think of on the spot:

Lack of knowledge/training in development paradigms  
Mislead prioritizing of featuresets  
Ignorance of existing tools, either willing or unwilling  
Lack of interest/motivation to find out if there are existing libraries for task XY  
Confidence that it is easy and fast to develop on one's own (like database abstractions and authentication layers)  
Unwillingness to conform to existing APIs  
Lack of understanding of existing APIs  
Stressed work environments that do not allow for research of existing tools (or training, for that matter)  
General mistrust of external libraries

I am sure a psychiatrist specializing in modern software developers would be able to find quite a bunch more issues, but I am not sure I want to know about that in such detail after all.

#### Expectations, Approaches

This is one more issue that I feel compelled to write about, since it is the reason why I have reprogrammed the wheel a few times in the past, and actually still do: different approaches to a problem.

As an example, have you ever wondered why there are really that many CMSs out there? Is it really because someone said "I can do it better"? That may very well be the case, but why did he/she think so? I think it is because that someone had a different idea of how a CMS should work, dismissing the other system because it did not match his expectations. Like the users using a software like a CMS will be more at ease with one or the other system, developers will be more at ease with one or the other library. PEAR's HTML\_QuickForm [2] essentially does the same than patForms [3], but in a different way - why does one developer choose the one over the other? The choice is not always based on the featureset.

I know that when I look at a library and it does not work the way I expect it to, I am easily tempted to find another (if possible) or think of writing one of my own. My friends and colleagues have largely succeeded in curing me of the syndrome, like by teaching me how to use wrappers to overcome the awkwardness of using an "alien" API. Now I usually only program a new wheel when there are not really any good libraries for the task at hand, or the library of choice has serious issues.

#### Real life and productivity vs interoperability

When I started my own business, I knew I would need an application that would help me build the websites for my customers, so I started looking around. There are many great frameworks and CMSs out there, and I took a good look at them. My decisionmaking was based on a few simple rules:

The system had to  
allow complete freedom of layout  
feature a multilingual frontend and administration  
be extensible to add any kind of

features have enough modularity to allow frequent structure changes  
be secure work under PHP4 (most of my customers run on shared hosting)  
be open source for easy adjustment to custom needs  
have a user-friendly and self-documenting interface for end users  
have an administration sdk to easily add new administration features  
take care of all redundant website building tasks  
allow me to let loose my customers in the administration interface  
allow me to use third-party libraries (interoperability)  
be around for a while

In everyday use, the goals were

to empower me to keep the quality as high as I want it to  
allow me to reduce my prices to lessen the gap with the discount offers of the competition  
to give me an edge on development times versus the competition

I don't want to turn this post into a feature comparison, so I will skip that step. After reviewing several frameworks and CMSs, I realized that none would fit what I wanted. Only a custom solution would work, because for the rate of development I targeted I needed a tool that functions on a level I can comprehend. Inspired by patPortal, I planned and started the development of SimpleSite [4]. As a strictly solo project at first and due to the development timeline priorities, interoperability was the least of my worries. Even now, after about a year and a half of constant development it is still lacking in that regard. However, I have made sure that it is interoperability-ready: all the required interfaces and structures are built to allow swapping components as you see fit.

I think that knowing how to prepare for interoperability and adding it to your application development plans is much more important than actually offering it, as it makes the job for other developers that much easier. I also know that it is not always that simple either however - in SimpleSite, the templating engine of choice is patTemplate. While you can use any template engine you like in your modules, the main structure is always handled by patTemplate, and there are related helper methods that only work with patTemplate. It is sometimes really a matter of weighing the development effort and related costs against the added value interoperability would add.

#### Related links

[1] Clay loveless' "Stop writing loner applications" post

[2] PEAR HTML\_QuickForm

[3] patForms

[4] The SimpleSite development portal

Posted by argh in PHP at 16:48

Monday, June 5, 2006

### **XML\_Parser tutorial published**

In January 2005 I wrote a tutorial on the PEAR package XML\_Parser, that I maintain. This tutorial was supposed to be published in the PEAR section of Zend's developer zone, as they planned on sponsoring PEAR developers that were willing to write tutorials for their packages. Sadly enough this tutorial has never been published, although Zend promised me to do so and I also never received the promised discount on the Zend certification.

When cleaning up my hard disk yesterday, I stumbled across the tutorial and decided, that it would probably be better to publish it on my own website instead of waiting for Zend to finally put it online. So if you still are using PHP4 or prefer SAX-based parsing although PHP5 offers a decent DOM implementation, you can now find the XML\_Parser tutorial on my website.

Be warned: Although the tutorial itself has been written in English, the rest of the website is in German, so you might find the navigation a bit strange. But no need to worry, the this link will guide you directly to the tutorial.

Posted by schst in PHP at 11:51