Technical Explorations

by Keith R. Bennett

Android GUI's: The Case Against GUI Builders and Data Driven GUI Configuration

May 28, 2011 • keithrbennett

GUI builders are great, but for building anything more than a trivial solitary application, without discipline and diligence, the duplication can create a productivity and quality quagmire. In addition, XML is a human-hostile configuration language.

Copy and Paste - Bad!

As the number of windows increases, the natural tendency is to copy and paste. For the same reasons copy and paste degrade the quality of source code, they do the same to user interface configurations.

Changing Project Wide Settings

On a Java Swing project on which I once worked, it was decided to follow the Java Look and Feel Design Guidelines, and to use some standard project-wide conventions. We had to change lots of windows and components, even down to the minutiae of border pixel widths. Fortunately, we had been creating our UI's in source code, so with some refactoring, I was able to confine these common characteristics to a single shared place in the code.

Consider what would have happened, though, if our GUI configurations had been encoded in XML by a GUI builder. We would have had to painstakingly edit each window's XML file and hope that the repetition and boredom did not numb us into introducing errors. Even if a smart developer automated the parsing, modifying, and regenerating of the XML, there would probably be errors because the way builders are usually used, there is no semantic information in them to communicate intentions…for example, a border width of twelve pixels in one instance might be there because absolute placement was necessary, whereas in another instance it was merely 'whatever our standard border is'.

Even if the rework were done with a GUI builder it would have been a lot of repetitive work. Note too, that in all the cases above, the work involved to make a change does not decrease very much with each successive change. Because of the huge cost of change, the natural result is a fierce resistance to proposed improvements by developers and management alike.

Mitigating the Damage – Using Custom Components instead of Framework Components

However, this could work much better by assembling pre-built custom project components. These project components would configure and combine framework components and conform to the project-wide settings without the need for any attention by the developer that uses them.

The Android platform has some features that enable the developer to extract duplicated configuration in an application into shareable fragments. In addition, across applications, it supports the creation of shared "libraries" containing both configuration and code that can be statically linked into an Android application's .apk file. This enables the use of standard configurations across applications (or even across the paid and free version of the same application). See Marc Lester Tan's excellent post about the available options here.

Creating a component could be done either by creating an XML file for it, or by creating a custom component class in the framework's programming language (Java in our case) and making it available to the GUI builder.

While either approach would centralize the customized settings, creating the component in code would enable assigning meaningful names to option values (e.g. *LookAndFeel.STANDARD_BORDER_WIDTH* as opposed to *12*). In addition, it would be possible to calculate values (for example, to accommodate different display device characteristics).

The Code / XML Disconnect

As the size of the UI grows, it's almost inevitable that duplication will happen, and the resolving of that duplication by refactoring will lag. However, when development occurs both in source code and in XML (with or without a GUI builder), the distance and disconnect between the two would logically result in a greater lag. Lazy or high pressure projects that routinely trade long term debt for short term velocity may even give up resolving duplication entirely.

Why Not Just XML?

So, if using both code and XML present a problematic disconnect, why not use only XML? Because XML is a suitable language for computers, but a miserable language for humans. A wise man once said "XML is like lye; very useful to humans, but they should never touch it."

While formatting and color coding XML code help make it a little more readable, it is nevertheless not very efficient at communicating. Consider, for example, a twenty line element repeated twenty times, where all elements vary only in the value of a single integer. Source code could easily communicate the similarities and differences, whereas XML would hide them.

Android – Java and XML

The Android development team encourages the use of XML rather than Java code for defining user interface elements. The Java language is verbose, rigid, and ceremonial, so defining the user interfaces in code is not that dramatic an improvement over XML.

In contrast, using a more flexible language (my choice would be Ruby) would make a better approach feasible. It would be possible to write a domain specific language (something like Ruby on Rails, but probably a lot smaller) that would simplify development and facilitate the creation of living (i.e. executable) configuration that is comprehensible, maintainable, and extensible. I think I'll call it…*Ruby on Roids*. 😉

There are, of course, technical challenges to using Ruby on the Android platform. The most difficult one is probably the limited amount of memory on handheld Android devices. Languages like Ruby use more memory than Java. The JRuby team is working on Ruboto and Mirah, both of which are promising, but not quite there yet.

What About Our Designers?

If all design is implemented as code, then it would be more difficult to separate design work from programming work. However, a solution to this would be to have designers design mockups with XML, and then have developers implement them in code. Alternatively, the two could work together when designing.

What Can We Do Now?

I eagerly await the coming of age of Ruby alternatives for Android development. Until then, my plan is to use Java, and to write my UI's in Java rather than XML as much as possible.

Personally, I've done very little work with GUI builders. For those of you who have, what was your experience? How did you handle duplication, maintainabilty, and extensibility? What were the challenges, and how did you overcome them?

Published with GitHub Pages