The Embedded Muse 90

Editor: Jack Ganssle (jack@ganssle.com) December 11, 2003

EDITOR: Jack Ganssle, jack@ganssle.com

CONTENTS:

Editor's Notes

There are no current plans to host a public Better Firmware Faster seminar in the near future. I often do this seminar on-site, for companies with a dozen or more embedded folks who'd like to learn more efficient ways to build firmware. See http://www.ganssle.com/onsite.htm for more details.

Rich Schmitt wrote an excellent paper on endianess which he's put on-line at http://www.bluepeach.com/endianess.htm. It's the most concise and complete discussion I've seen of this issue. Highly recommended.

Book Review – High Integrity Software

"High Integrity Software" – the title alone got me interested in this book by John Barnes. Subtitled "The SPARK Approach to Safety and Security", the book is a description of the SPARK programming language's syntax and rationale.

The very first page quoted C.A.R Hoare's famous and profound statement: "There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies." This meme has always rung true, and is one measure I use when looking at the quality of code. It's the basis of the SPARK philosophy.

What is SPARK? It's a language, a subset of Ada that will run on any Ada compiler, with extensions that automated tools can analyze to prove `the correctness of programs.`

As the author says in this Preface, "I would like my programs to work without spending ages debugging the wretched things." SPARK is designed to minimize debugging time (which averages 50% of a project's duration in most cases).

SPARK relies on Ada's idea of programming by contract, which separates the ability to describe a software interface (the contract) from its implementation (the code). This permits each to be compiled and analyzed separately.

It specifically attempts to insure the program is correct as built, in contrast to modern Agile methods which stress cranking a lot of code fast and then making it work via testing. Though Agility is appealing in some areas, I believe that, especially for safety critical system, focus on careful design and implementation beats a code-centric view hands down.

SPARK mandates adding numerous instrumentation constructs to the code for the sake of analysis. An example from the book:

```
Procedure Add(X: In Integer);
--#global in out Total;
--#post Total=Total~ + X;
--#pre X > 0;
```

The procedure definition statement is pure Ada, but the following three statements SPARK-specific tags. The first tells the analysis tool that the only global used is Total, and that it's both an input and output variable. The next tag tells the tool how the procedure will use and modify Total. Finally a precondition is specified for the passed argument X.

Wow! Sounds like a TON of work! Not only do we have to write all of the normal code, we're also constructing an almost parallel pseudo-execution stream for the analysis tool. But isn't this what we do (much more crudely) when building unit tests? In effect we're putting the system specification into the code, in a clear manner that the tool can use to automatically check against the code. What a powerful and interesting idea!

And it's similar to some approaches we already use, like strong typing and function prototyping (though God knows C mandates nothing and encourages any level of software anarchy).

There's no dynamic memory usage in SPARK – not that `malloc()` is inherently evil, but because use of those sorts of constructs can't be automatically analyzed. SPARK's philosophy is one of provable correctness. Again… WOW!

SPARK isn't perfect, of course. It's possible for a code terrorist to cheat the language, defining, for instance, that all globals are used everywhere as in and out parameters. A

good program of code inspections would serve as a valuable deterrent to lazy abuse. And it is very wordy; in some cases the excess of instrumentation seems to make the software less readable. Yet SPARK is still concise compared to, say, the specifications document. Where C allows a starkness that makes code incomprehensible, SPARK lies in a domain between absolute computerese and some level of embedded specification.

The book has some flaws – it assumes the reader knows Ada, or can at least stumble through the language. That's not a valid assumption anymore. And I'd like to see real life examples of SPARK's successes, though there's more info on that at http://www.sparkada.com/.

I found myself making hundreds of comments and annotations in the book, underlining powerful points and turning down corners of pages I wanted to reread and think about more deeply.

A great deal of the book covers SPARK's syntax and the use of the automated analysis tools. If you're not planning to actually use the language your eyes may glaze over in these chapters. But Part 1 of the tome, the first 80 pages which describes the philosophy and fundamentals of the language and the tools, is breathtaking. I'd love to see Mr. Barnes publish just this section as a manifesto of sorts, a document for advocates of great software to rally around. For I fear the real issue facing software development today is a focus on code uber alles, versus creating provably correct code from the outset.

High Integrity Software, The SPARK Approach to Safety and Security, by John Barnes. Published by Addison-Wesley, ISBN: 0321136160.

Book Review – Embedded Ethernet and Internet Complete

Jan Axelson's latest book extends her coverage of communications protocols from RS-232 to the PC's parallel port to USB to (in this volume) Ethernet and the Internet. Like all of her works, it's clear and complete.

Ms. Axelson starts with a brief overview of the basics of networking, with an overview of the datagrams, protocols and required hardware. She goes on to describe how routing is accomplished using both UDP and TCP/IP. Technical readers will already be familiar with much of this information, but it's presented in a very readable fashion that's easy to understand.

Much of the book covers serving up Web pages and working with dynamic data on the web. Sure, there's been plenty of coverage in hundreds of books of these subjects, but she

presents everything in the context of embedded systems, especially systems with limited resources. And that's where the real strength of this book lies.

Like her previous work, this is a practical, how to use networking in an embedded system book. Two primary hardware platforms are covered: the Dallas TINI and Rabbit Semiconductor's RCM3200. I've used the Rabbit parts a lot, and find them powerful and fun to work with.

If you're already a networking whiz this book probably won't add much to your knowledge. But for those anxious to learn how to add Ethernet and/or Internet connectivity to any embedded system, especially small ones, the practical examples and clear writing will get you going quickly.

Ms. Axelson's web site (www.lvr.com) has long been the place to go to learn about USB, RS-232 and other subjects. It's already filling with lots of useful information about networking that serves as a companion to the book. Check it out!

Embedded Ethernet and Internet Complete by Jan Axelson, ISBN# 1-931448-00-0.

Joke for the Week

Ever wonder about where "foo" and "foobar" came from? Many of us think foobar derives from the WWII term FUBAR, an acronym there's no need to expand here (this is a family-oriented Muse, after all!). Turns out that may not be accurate.

The Internet community has several hundred RFCs that use foo and foobar, so of course yet another, RFC 3092, explains their etymology. It's an interesting and amusing read. See http://www.faqs.org/rfcs/rfc3092.html (an alternative site is http://www.ietf.org/rfc/rfc3092.txt).

About The Embedded Muse

The Embedded Muse is an occasional newsletter sent via email by Jack Ganssle. Send complaints, comments, and contributions to him at jack@ganssle.com.

To subscribe, send a message to majordomo@ganssle.com, with the words "subscribe embedded *your-email-address*" in the body. To unsubscribe, change the message to "unsubscribe embedded *your-email-address*".

The Embedded Muse is supported by The Ganssle Group, whose mission is to help embedded folks get better products to market faster. We offer seminars at your site

offering hard-hitting ideas - and action - you can take now to ***improve firmware quality and decrease development time***. Contact us at info@ganssle.com for more information.