Architectural Balkanization in the Post-Linguistic Era (Volume Two)

Brian Foote
University of Illinois at Urbana-Champaign
foote@cs.uiuc.edu
http://www.laputan.org
25 October 2005

The Eclipse Juggernaut is one of this young century's most striking software phenomena. It's Smalltalk heritage and industrial-style roots gave little initial indication that it would mushroom into a bona-fide platform and potentially challenge the likes of Windows, and whatever laundry product (Ajax? SOAP?) is associated with the global web services movement this week.

Much can be said of Eclipse, most of which has already been said by others, about its open source heritage, and the like. I shall forego that, for now at least.

My focus here will be on how Eclipse came to be a platform that was good for everyone by being a platform that was good for programmers. By attracting programmers. By being a place that they wanted to live. Where they were willing to cooperate. Where they were anxious / eager to pitch in. What's sauce for the gander is sauce for the geese.

Eclipse is not a construction site. The construction metaphor for software is badly broken, but staggers on, because the suits still wish it were so. We don't build software like we build a skyscraper and walk away from it. A viable system is a living organism, a honeycomb, as long as the colony is there. It's a ghosttown, its useless, once they are gone.

If you want the honey, you have to tolerate the bees. Sure you get stung once and a while. God help me, now, I'm promoting a Winnie-the-Pooh model of sustained yield software cultivation. Blame my post-OOPSLA buzz.

So where was this going? Oh yeah. Eclipse works because programmers live there. Lots of them. We're building a what? A place, a community, a cathedral, Our Town, Houston, Brasilia, a boomtown, a what? But we are building it.

It works because we are focused on the code. First and foremost. The program is our artifact of record. It the focus of our efforts. Tending it is what we do.

So what it is?

Before I try to answer that question, a quick observation: Eclipse has a fascinating amalgam of brute-force Java components held together with XML Glue. It's what I've called a Big Bucket of Glue. It's very post-modern, very worse-is-better, very 21st century. It's very reflective. I got really excited when I realized that it was held together with reflection. Duct tape and mirrors.

I was amazed when my first cut and paste plugins were two distinct black-box Java programs neither of which anywhere directly referenced the other. There were bound together instead by reflection and XML glue.

XML is, generally speaking, too horrible to gaze upon directly with the naked eye. It is sad that one who gazed upon the Medusa, it was said, was instantly petrified, literally turned to stone. I find I'm figuratively petrified when I see XML.

When I look at it through the filters of a form based interface, however, it doesn't seem so formidable…

The Five-Hundred Pound Gorilla

Nonetheless, the 500 pound gorilla in the middle of the Eclipse room, the unspoken focus of everything we do in this Developer Disneyland we are creating, is The Program. It's **program representation**.

All of Eclipsedom is focused on this to some level, though not all realize this.

Many had spoken over the years of moving beyond our ASCII/EBCIDIC (sp?) punch card text base 95 character legacy / notion of what a program is.

The discussion of expanding the character sets and type setting constraints of the punch card era is not noew. Think of APL, or Algol 68.

The notion of moving beyond text to something else, something more central, something more indigenous to our world, is newer, but still old. The Grail: buildling programs out of objects. What we'd settle for: a much richer, more malleable program representation than is possible in traditional programs.

The text? We'd just "round-trip" create it from the real, official, canonical program.

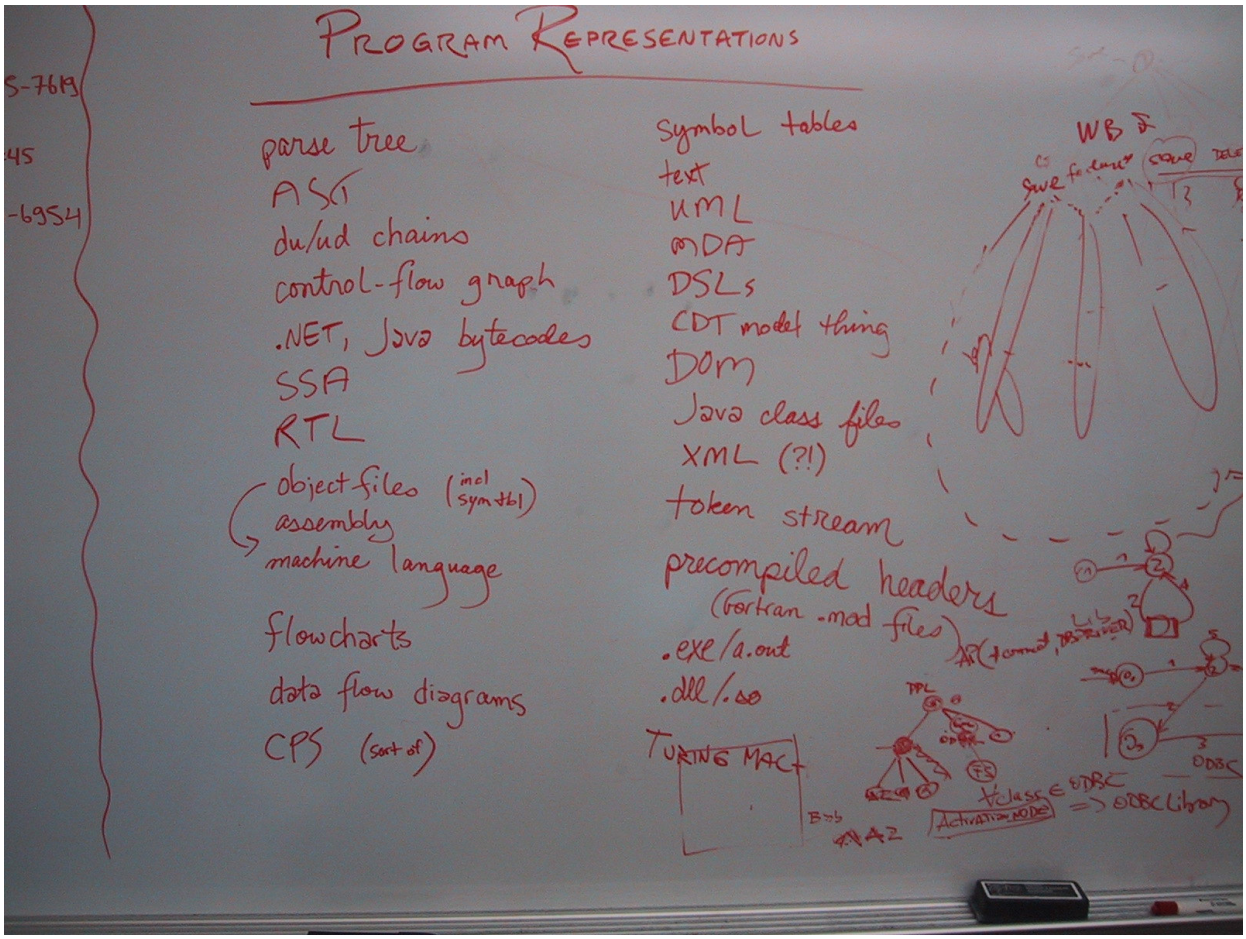
As with the Blind Men and the Elephant, we see the Eclipse community converging on this problem from a number of directions:

- Programmers want TEXT++, updated documentation, etc., not simple ASCII
- Programmers need a parse tree level representation to implement refactoring
- Modeling guys and gals want pictures and programs to linked: round-trip reality, not voodoo computer science where changes to the representation have no effect on that which is being represented
- Metric mongers want a program representation over which they can compute…

Thie Vision: this representation would be somewhere around the AST level. It would be somewhere deeper than the Outline view objects in the JDT and CDT, somewhere around the AST/Parsetree/DOM levels in play as we speak.

It would be naïve to think there will be a single representation, there will be many, it would be suicide to give up on finding out what they have in common, and to not find a way to make them freely convertible.

This area, in my opinion, offers some of the most fertile ground for productive research and practical innovation in all of the realm. May a thousand flowers bloom.



No single model, no single representation, no single framework is likely to emerge. More likely what we will see is architectural balkanization in the post-linguistic era. Whether these representations are stale, isolated fiefdoms, silos, or part of an organic catalytic ecosystem, remains to be seem…

These architectures, these designs, will be far from perfect. This is a worse is better world. A heterogeneous, polyglot world. But freely convertible currencies, and perhaps a lingua franca may not be too much to hope for…