eWEEK ENTERPRISE NEWS & REVIEWS

LISP Deserves a Fresh Look

February 7, 2006

By Peter Coffee

Development tools that are programmer-friendly but hardware-intensive, notably LISP, have long been a staple of research domains such as artificial intelligence—but have not been thought suitable for mass-market applications due to difficulties in packaging the executable bits and running them on end-user hardware.

Web-facing applications, whose running code resides on servers, shift those trade-offs and invite "research" languages such as LISP to contend for mainstream roles.

RELATED LINKS

- Someone's Got to Take Charge
- Security May Not Be Part of the Service
- Making Code Perfectly Clear
- SOA Governance Gets Real
- Software Makes Uncommon Sense

Formal studies of programmer productivity and program performance—with multiple programmers doing side-by-side implementations of multiple tasks—suggest that functions can be performed with fewer lines of code, with less variability in development time and with acceptable memory use and execution speed using LISP and LISP-based systems.

Developers handicap themselves if they play by old rules on the Web's quite different field.

That which was

The current generation of application developers has been imprinted with a business model of mass-market software as frozen bits, packaged as executable binary files, delivered on inexpensive media units—floppy disks or CDs—to run on a PC.

This model is merely an artifact, though, of one brief stage in the evolution of technologies and markets.

Click here to read about the importance of making source code clear.

The model of "bits in a baggie," as one might call it in homage to the Apple II era, defined its own criteria for programming languages: ease of compilation, minimum code size for ease of distribution and minimum memory footprint for acceptable performance on low-cost PC configurations.

When product-cycle durations are measured in years—or, at a minimum, in quarters—a cumbersome process of turning source code into saleable bits is tolerable; when a mass-market application is delivered as function across a network, not as raw bits in a licensee's hands, the equilibrium state is far more friendly to developers.

That which is

Imagine the perspective of the proverbial man from Mars, the hypothetical observer with no preconceived ideas about what makes sense.

The man from Mars sees development languages being chosen as if processor cycles are expensive, when a growing fraction of today's computing workload runs on vast farms of cheap boxes whose cost is spread across entire communities of users.

Martian question 1: Can't you use more processor-intensive methods if they deliver more powerful applications more quickly?

The man from Mars sees development languages being chosen for their ability to condense ideas into executable bits today—not for ease of incorporating new ideas tomorrow.

Martian question 2: If the language you're using requires you to be as clever as you can just to get something working in the first place, how will you ever be clever enough to improve it?

Next page: Reintroducing LISP.

The man from Mars sees development languages being chosen for the convenience of machines, despite attendant productivity penalties and difficulty of delivering high-quality code, instead of being chosen for the convenience of the developers who are the actual scarce resource.

Martian question 3: Aren't your machines getting faster much more quickly than your programmers are getting smarter?

Symbolic acts

At this point, one could say, "Enter LISP," except that LISP has already been on the stage longer than any extant programming language other than FORTRAN.

With a name that is a contraction of "LISt Processing," LISP represents its programs in exactly the same manner as its data—that is, as linked lists of symbols. It can therefore be used to write programs that write new programs as readily as conventional languages construct a vector or array of data.

A subset of such programs includes those that define new programming languages, specifically tailored to solve new problems. Although LISP originated in AI (artificial intelligence), being used to write programs that extended themselves in models of machine learning, LISP has also proved useful in building other customizable systems.

For example, Autodesk's AutoCAD drafting tool has always incorporated a LISP-based language and a LISP run-time environment that enable powerful extensions.

Autodesk recently tapped operations chief Carl Bass to be the new CEO. Click here to read more.

LISP's power in manipulating complex lists of lists has also made it the tool of choice for developing symbolic reasoning systems such as OPS5 and PROLOG (both of which have many commercial and public-domain implementations).

OPS5 found fame as the language of XCON, a minicomputer configuration tool that greatly reduced deployment costs for Digital Equipment, while PROLOG engines from providers such as the Swedish Institute of Computer Science are embedded in a wide range of applications.

Both OPS5 and PROLOG are actually incorporated as utility languages in the Allegro CL (Common Lisp) 8.0 development suite, released in January by Franz.

Click here to read more about Allegro CL 8.0.

In addition to the robust and fully supported Franz product, there are numerous open-source and public-domain LISP implementations enabling developers to experiment with the language.

That which is no more

Like anything that's been around for several decades, LISP carries the baggage of what "everyone knows" about it that is no longer true.

"Everyone knows," for example, that LISP is an interpreted language and, therefore, too slow for production applications—except that modern LISPs can compile functions for run-time speeds competitive with those of C or C++ programs in algorithmically complex tasks.

Attendees of the C++ Connections conference got a glimpse of the language's future. Click here to read more.

The trade-offs are clear. In a study performed in 2000 by Erann Gat, a researcher at the California Institute of Technology's Jet Propulsion Laboratory, programmers writing in LISP produced programs with less variability in performance than more experienced programmers writing in C and C++.

The fastest versions of C and C++ programs were faster than most LISP implementations, but the median performance of the LISP implementations was actually twice as good as the median performance of the C and C++ code performing typical tasks (more at www.flownet.com/gat/papers/lisp-java.pdf).

For real-world teams, such reduction of technical risk and improved worst-case scenarios arguably outweigh best-case results.

The LISP implementations in Gat's study were more memory-intensive than those in C and C++, although LISP's memory use was comparable to that of Java while performance was much better. A key point, though, is that applications are increasingly being delivered to users via the Web, and that developers are therefore freer to use tools that maximize their own flexibility at the supply end.

"In the near future, when everything you encounter is programmable and everyone wants to program [those devices and environments] the ability to create new programming languages will be paramount," said computational biology researcher Jeff Shrager, in Stanford, Calif., in a conversation with eWEEK Labs.

"The combination of AI capability with the ability to create entirely new programming languages make LISP uniquely situated for the near future of computation."

Technology Editor Peter Coffee can be reached at peter_coffee@ziffdavis.com.

Check out eWEEK.com's Application Development Center for the latest news, reviews and analysis in programming environments and developer tools.