Are Domain-Specific Languages Going Mainstream?

Anwar Ghuloum (Intel)

A subtext of discussions of domain-specific languages (DSLs) is that they are niche-y, with limited appeal beyond a small, devoted following. At the surface, there are a few counter-examples to this kind of thinking, most notably in shader languages for graphics programming, which are among the most widely used programming languages for parallel computing today. Are there other trends that might indicate that DSLs are gaining traction in the mainstream?

Historically, there are some qualified indications to the contrary. Matlab, Mathematica, R, Maple, and other technical computing languages are widely used in the earliest stages of the project lifecycle for algorithmic design and exploration. These rarely make it deployment in production software, however. The fundamental barrier for these languages is performance. They facilitate high-productivity transcription and testing of ideas, but they usually do not provide sufficient levels of performance.

Other aspects of DSLs are also problematic. Their focus on specific problems or data types seems to inherently limit their uptake. Most developers use a general purpose language, typically a C-language variant, managed runtime (Java and .Net), or FORTRAN. These are even beginning to encroach on the aforementioned domain of graphics - perhaps to supersede shader languages as they exist today.

It seems like a pretty bleak outlook for DSLs, but there are several trends worth noting that paint a much rosier picture for DSLs.

Architectural trends inherently narrow developer focus to some extent. That is, while the domain-specificity of DSLs is derides as inherently limiting their appeal, the current focus on identifying scalable parallel portions of programs multi- and many-core architectures is narrowing closing the typically wide aperture of developers choice. For example, scalable data parallelism is necessary (but perhaps not sufficient) condition for the expressiveness of programming languages because of its applicability in a variety of signal, image, and video processing domains (not to mention a variety high-performance computing domains). In particular, image processing and animation APIs are key differentiating features in many SDKs (see the Mac OS X and iPhone SDKs!). The emergence of standards around low-level data parallel computation, such as OpenCL (which primarily supports regular array types), is another indication of this trend.

Despite the conventional wisdom, multi-language development is increasing, especially in cases where user and/or developer customization and adaptation is desired. In these cases, 'mainstream' programming languages are often married to managed and dynamic languages to build applications. This has been driven by productivity concerns like choosing the best tool for the job. But it has fundamentally been enabled by progressively improved interoperability between dynamic, managed, and unmanaged languages. As an example, common Language Runtime implementation (on .Net and Mono) in recent years have hardened support across these language types, with a single runtime capable of

simultaneously supporting managed, functional, unmanaged, and scripting languages. 'Supporting' in this case means both managing execution and promiscuous data exchange.

The final, and perhaps most controversial, trend is the widespread acceptance of dynamic compilation and meta-programming techniques, especially those typically seen in C++ template programming and the Eval constructs of various languages (Lisp, Python, etc.). C++ template programming is somewhat hampered by variation in the standards compliance of C++ compilers and oft-inscrutable template error messages, but has still seen significant growth (Boost, Blitz++, among many). Dynamic compilation of the 'Eval' flavor is emerging thanks to a couple of factors: the growth of web programming languages and infrastructure and recent efforts to overcome limits on static compilation in the face of significant software abstraction and architectural velocity. The latter, in particular, is the rational for efforts in Matlab, C++ (Ct-Rapidmind, Accelerator), and autotuning research (UC Berkeley). In a change from the Java and .Net driven perception, dynamic compilation for performance is amassing a body of evidence and gaining credibility.

This nexus of hardware-driven performance requirements, multi-language acceptance, and performance-driven dynamic compilation will drive an era in which DSLs will become a normal part of everyday development activities.