Functional Programming in the Real World

Editor: *Philip Wadler, University of Edinburg; http://homepages.inf.ed.ac.uk/wadler/*

Report on the First Commerical Users of Functional Programming Workshop

Andrew Moran
Galois
12725 SW Millikan Way, Ste. 290
Beaverton, OR 97005
`moran@galois.com`

Functional languages have been with us for a little over a generation now. Languages like Lisp, Scheme, ML, OCaml, Haskell, and Erlang have well engineered compilers and tools and large user and development communities. Not surprisingly, some of these users work in industry. While there are only a few companies where functional languages are used exclusively, there are many that use functional languages for design exploration, prototyping, modeling, specification and design, building compiler-like tools, and even for developing product.

In recognition of this the first Annual ACM SIGPLAN Workshop for Commercial Users of Functional Programming (CUFP) was held on September 18th, 2004, in Snowbird, Utah. It was co-located with the 2004 ACM SIGPLAN International Conference on Functional Programming (ICFP).

The goals of the workshop were to act as a voice for commercial users of functional programming languages and technology; to help functional programming become increasingly viable as a technology for use in the commercial, industrial, and government space, by providing a forum for functional programming professionals to share their experiences and ideas, whether business, management or engineering, and to enable the formation and cementing of relationships and alliances that further the commercial use of functional languages.

There were 25 attendees, and the workshop was based around 9 short presentations from a diverse cross-section of commercial users, representing Abstrax Inc., Microsoft, Linspire Inc., Beckman Coulter Inc., Cadence Research Systems, Galois Connections, Inc., and Bluespec Inc. Participants were encouraged to view the speakers as leading discussion, as opposed to giving academic presentations, and this led to spirited and fruitful discussions.

In short: the workshop was very successful, and we hope to make it a regular event.

Throughout the day, a number themes emerged from talks and the discussions they engendered. The rest of this article describes those themes in more detail.

1 The Benefits of Using Functional Programming in Industry

All the speakers agreed that their use of a functional language had given them a large advantage over use of a more traditional language (such as Java, C++, C#), and some felt their task would have been almost impossible otherwise.

Some specific benefits that were mentioned by a number of speakers are:

- Powerful type systems;
- Rapid prototyping;
- Ability to code complex algorithms;

- Using continuations for backtracking;
- Scheme's hygienic macros;
- Easy creation of domain-specific languages (DSLs);
- Good memory management;
- High productivity, and
- Portability.

These were familiar to everyone in the audience, and are familiar to everyone in the functional programming research community, as they have long been touted as concrete reasons for using functional languages to solve industrial problems. It was pleasing to see that the research community's intuition was borne out: these factors really do make a difference.

Some new benefits were emphasized by Galois. For Galois, it's also a big bonus that Haskell is close to its mathematical roots, because their clients care about "high assurance" software. High assurance software development is about giving solid (formal or semi-formal) evidence that your product does what it should do. The more functionality provided, the more difficult this gets. The standard approach has been to cut out functionality to make high assurance development possible. But Galois clients want high assurance tools and products with very complex functionality. Without Haskell (or some similar language), Galois said, they wouldn't even be able to attempt to build such tools and products.

2 Problems Encountered

After introducing their particular problem domain, and how functional programming enabled them to succeed, most speakers turned to the barriers to using functional programming in industrial contexts, and issues they faced day to day.

2.1 Technical Issues

Many of the common technical barriers to widespread use of functional programming in industry were quite familiar: lack of libraries, libraries not being tuned for performance, poor performance in general, lack of platform support (particularly tools for Windows), and lack of industrial-grade development and debugging tools. A strong barrier to the use of functional languages in large companies was the fact that internal build and testing infrastructure just isn't designed with functional languages in mind.

A few others concerned the languages themselves: too much abstraction can lead to confusing and hard to maintain code, higher-order functions are difficult to test well, and lazy languages have space behavior that is difficult to predict.

Byron Cook, of Microsoft Research, pointed out a dramatic irony about the state of the art in formal verification and analysis tools: they are many excellent and powerful tools for C, C++, C#, and Java, but nothing comparable exists for functional languages.

2.2 Non-Technical Issues

In spite of this preponderance of technical issues, there was consensus that the biggest problems are non-technical in nature.

Most of these had to do with the fact that many functional language implementations are open source research projects. This means that there's no 24-hour support hotline, or even someone who will guarantee timely bug fixes.

There's no product roadmap, and the long-term viability of a functional language compiler often rests upon the shoulders of one or two individuals — this was referred to as the "A Bus Hits Simon Peyton Jones" problem several times.

Open source projects worry companies from a legal standpoint too. Managers are concerned about using "unproven" technology in product, legal teams are concerned about patent issues (such as those raised by SCO), and no-one wants to touch the GPL[1]. Even the LGPL[2] is seen as too hot to handle.

Existing infrastructure, such as testing and build mechanisms, is almost always geared toward traditional languages, and trying to add support for a functional language is a difficult prospect, technically and culturally. Testers are rarely familiar with functional programming languages, and tend to be extremely reluctant to take on the considerable burden of learning functional programming.

And then we have broader cultural problems. Many speakers talked of the difficulty of convincing their managers to make the switch to functional programming. Kent Dybvig, of Cadence Research Systems, ran through a long list of myths about functional programming, detailing out ways in which one can debunk those myths that have no basis in reality, and pointing the way forward for dealing with those still have some truth in them[3]. For example, the fact that, with one or two exceptions, companies cannot pay for support often makes functional languages unpalatable. This was seen a red herring, as even when a company can pay for support for a compiler, they cannot expect to have show-stopping bugs fixed in a timely manner.

Related to this is the fact that there simply aren't that many skilled functional programmers out there. This poses a problem for companies wanting to develop product with functional languages, but is also of concern to those companies clients: who will support and maintain the code base on the long-term? Clients often ask for product source code to be put in escrow, as a mitigation against the risk that the developer folds, but this is of no help if the client doesn't have any functional programmers.

Educators at the workshop complained that it was hard for them to convince students that functional programmer was more than an amusing diversion, and that they could find jobs using languages other than C and Java.

3 Some Solutions

Not surprisingly, most discussions during the workshop focused on problems and issues with the use of functional languages in industry. A few lessons were shared, and a few ideas to help ease the adoption of functional programming were forthcoming.

Sell the capabilities, not the paradigm Or, market functional programming as a powerful suite of tools, rather than a new or competing paradigm. Taking advantage of powerful tools is a far less risky venture than betting on a whole new paradigm.

Get domain-specific This is related to the first point: focus on an application domain where functional programming ideas and techniques can be particularly effective. Success has already been seen in cryptography (Cryptol, a language for specifying and implementing crypto algorithms), hardwire design (Bluespec, a declarative hardware description language), industrial chemistry experimentation (Beckman Coulter's graphical language for describing experiments), build-to-order manufacturing (Abstrax, domain-specific languages and optimizers), and applied formal methods (the SLAM project at Microsoft, for analyzing protocol adherence in Windows drivers).

[1]GPL, or GNU General Public License, is a free software license and a strong "copyleft" license. Basically, any released improved versions of GPL software must also be GPL, as must any released software making use of GPL software. See `http://www.gnu.org/copyleft/copyleft.html` and `http://www.gnu.org/licenses/gpl.html` for more detail.

[2]LGPL, or GNU Lesser General Public License, is a free software license but lacks the strong "copyleft" nature of the GPL, in that it allows linking with non-free software. See `http://www.gnu.org/licenses/lgpl.html` for more detail.

[3]The CUFP web-site, `http://www.galois.com/cufp/`, has links to a number of the talks presented, including Dybvig's.

Hire smart people; don't worry about FP Look for developers who think abstractly, who take great joy is creating elegant solutions, and who see the value of mathematics to programming; they'll pick up functional programming readily.

Initial success is crucial Be conservative and successful in the beginning — always deliver high quality, on time. Management and clients will respond positively to this, but if you fail, even once, even where traditional approaches have failed worse, that failure will be used as an argument against functional programming.

As for long-term tool support, two approaches were proposed: do it in-house, or form a consortium with other companies to develop tools and provide support. Each of these has it own problems: the former risks balkanization and isolation, the latter requires that a critical mass of participants be reached quickly.

4 Lessons Learned

At the close of the day, Workshop Chair John Launchbury led the participants in an analysis of the workshop: what worked, and what didn't. Happily, the workshop has more positive aspects than negative.

There was room for improvement in a number of dimensions, chiefly that there workshop was attended by a small group of industrial practitioners (about 20 in total), and that there were some notable absences: there were no representatives from Ericsson (which has of the order of 2000 Erlang programmers) nor from Yahoo (some of whose tools are implemented in Lisp, based on the work of Paul Graham). Also, while many problems were discussed, not many concrete solutions were offered.

But these problems did not detract from the experience for the participants. Dialog was productive and interesting, and the flexible schedule and discussion-oriented talks were greatly appreciated. Meeting other practitioners, and learning more about their experiences with industrial application of functional programming, was highlight for all concerned. There was an interesting diversity in talks, and everyone was exposed to a variety of perspectives on how to handle the reactions of management and clients and how to tackle problems as they arise.

Plans are in place for a second CUFP next year, and hopes are high that it prove just as valuable, not only to the participants, and to industrial users of functional programming in general, but also to the functional programming community as a whole.

5 Vital Statistics

The members of the Program Committee were:

- Koen Claessen, *Chalmers University of Technology*
- Byron Cook, *Microsoft Research*
- David Fox, *Linspire*
- John Lalonde, *Abstrax*
- John Launchbury, *Galois* (Chair)

Slides for most of the talks are available at the Workshop web page, `http://www.galois.com/cufp`.

Andy Moran is a member of Senior Technical Staff at Galois, a software contracting company based in Beaverton, Oregon. Moran founded Galois with several colleagues from the Oregon Graduate Institute, with the purpose of applying programming language research techniques to government and industrial problems. He earned his PhD in the Semantics of Programming Languages from the Chalmers University of Technology, Gothenburg, Sweden, and an MSc and BSc from the University of Queensland, Brisbane, Australia.