Workshop for Commercial Users of Functional Programming, Tallinn, Estonia, September 2005.

Functional programming as a means not an end.

Many of the slides from this meeting are available from the CUFP website:
http://www.galois.com/cufp/

Notes by Simon Thompson, Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK. s.j.thompson@kent.ac.uk

Draft 2, 16 October 2005

Introduction

The first one-day workshop for Commercial Users of Functional Programming took place in September 2004 in Snowbird, Utah, as a part of ICFP 2004, and was reported in the December 2004 issue of SIGPLAN Notices by Andy Moran. The second meeting, again co-located with ICFP, was held on 24 September in Tallinn, Estonia. Attendance was a healthy 40 plus, doubling last year's numbers. Eight speakers shared their experiences of using functional languages 'in anger' and two wide-ranging discussions completed the day's programme.

The goal of CUFP is to build a community for users of functional programming languages and technology, be they using functional languages in their professional lives, in an open source project (other than implementation of functional languages), as a hobby, or any combination thereof. In short: anyone who uses functional programming as a means, but not an end. In that sense the 'commercial' is a misnomer, and may well be dropped by future workshops.

The emphasis of the workshop is on use rather than research or development, and as such, the focus of the meeting is quite different from the main tracks. Instead of the emphasis on yet another language enhancement or extension, the focus is the complete package of language, libraries, implementations, tools and training, and how these together support or hinder the widespread adoption and use of functional languages in large-scale projects, be they commercial, open source, or indeed in universities.

In the following notes, many of the points made in the individual talks have been addressed in the second half of the report which summarises discussions at the workshop.

Exemplars

High-profile examples of success are one of the most potent marketing tools, and the workshop was able to hear a number of these which have received a lot of publicity over the last twelve months.

Pugs: Perl 6 For Real. Autrijus Tang http://pugscode.org/

From the abstract of the talk**:** Perl is a general-purpose language, known for its vast number of freely available extension libraries. The Perl 6 project was started in 2000 to create a more robust runtime environment, and to improve the language's support for multi-paradigmatic programming. However, the attempt to produce a concrete implementation remains a work in progress after 5 years of development.

This talk presents Pugs, a working implementation of Perl 6, written in Haskell. We review the challenges posed by Perl 6's unusual cultural and technical demands, how Pugs tackles them using Haskell as a host language, and the community that arises around Pugs development.

The Pugs project has generated real excitement in both the Haskell and the Perl communities over the last twelve months as this open source development has unfolded from inception to fully-fledged implementation. Pugs builds on a range of the most advanced features of GHC Haskell, but its main attraction to its users is the fact that it's a full implementation of Perl 6;

its choice of implementation language is immaterial to its use.

The Pugs project is also strong evidence for the effectiveness of functional programming in Haskell within Open Source collaborative projects: Pugs has attracted some 120 committers writing over 7000 commits, and in doing this has converted many of the committers to functional programming. One problem, discussed in more detail below, is that of how such experienced programmers can most effectively learn how to work in Haskell.

The talk also gives me the pretext for reminding readers of the 1993 proposal for Haskerl, a system which combines the best features of Haskell and Perl [Haskerl].

MLDonkey. Fabrice Le Fessant http://mldonkey.org/

MLDonkey is a popular multi-platform, multi-network P2P client written in OCaml (200k lines, with 20k lines of C). As with Pugs, the value of MLDonkey to its users is its functionality, rather than its implementation. The system has provided a platform for research into large-scale distributed algorithms, and provided the first open-source client to access the eDonkey network, with the eDonkey protocol being reverse-engeneered using the protocol sniffer Pandora.

In discussing adoption, Fabrice made the point that number of users will be inversely proportional to the size of the installation document; most users now expect a system to work straight after running the appropriate installer.

Fabrice concluded with a common functional programming dilemma. Many projects star ted to copy or "rewrite" MLDonkey in C, C++, Python or Java during the last 3 years; none has succeeded. But despite this, open-source developers still prefer Python, Perl, Java, etc: why ?

The Myth and Reality of using Haskell in the "Real World": experiences from darcs. David Roundy. http://abridgegame.org/darcs/

David talked about darcs, a popular and flexible revision control system well-suited to highly distributed development, written in Haskell. He talked briefly about darcs, but more details of the system itself can be found at the darcs web site; instead he concentrated on the myths and realities of using Haskell in a substantial project. Just as with MLDonkey, darcs is thus far superior to any re-implementations in traditional languages (in this case Python and Java). Roundy's slides give a detailed rebuttal of common myths expressed in quotes from the open source community. He also makes a detailed quantitative comparison of the numbers of contributors to darcs and other non-functional open source projects, with the result that darcs is supported by as diverse and wide a community as others.

Experience

Other speakers discussed their experience of using functional programming languages in an industrial context.

- Michael Sperber; talked about uses of Scheme 48 in the banking industry, and in particular for modelling and pricing financial derivatives. Experience had shown that the programs "were not pretty", but were fitter for purpose than comparable C++ systems, delivering results more quickly
- Jim Grundy of intel [Grundy] talked about his experiences with functional programming at intel's Strategic CAD Labs .Intel has evolved its own functional language tailored with domain specific extensions for hardware verification: reFLect. The high cost of developing and maintaining their own language is seen as unfortunate, and they plan to re-engineer the system on the OCaml infrastructure. Intel has developed significant verification infrastructure in this language, and leverages this infrastructure by putting in the hands of 'regular' C++ (and tcl) developers by making the embedding of reFLect functions and data in C++ as easy to use as native C++ code, and would recommend this route to others who seek a wider user base for their functional systems.
- Robert Boone's experience came from using LISP-based systems within Freescale, http://www.freescale.com/. LISP is an established technology in AI / natural language

applications, and receives particular support in government sponsored projects.

- Jonathan Soebel; talking about lessons the FP community can learn from the OMG's Model Driven Architecture [MDA] and its related marketing efforts. In particular, MDA puts transformation to the fore, and this is a domain in which functional programming is a clear leader.
- Francesco Cesarini; of Erlang Training and Consulting Ltd. [ErlCons] , talked about his experiences using, teaching, and supporting the concurrent functional language Erlang. After discussing how companies can be convinced to go the Erlang route, Francesco addressed the question of raising the profile of Erlang. In particular, the question of an 'Erlang Foundation' to supervise and support language developments had been mooted after the last CUFP meeting; finding the resource (time and money) for this has proved to be a major hurdle. One possibility for funding this is to issue Support Licenses, which give licensees access to additional support and assurance.

Discussions

A summary of the CUFP discussion was fed back into the closing session of the Workshop on Functional and Declarative Programming (FDPE), which followed CUFP. This summary reflects both discussions.

Barriers to acceptance of FP within companies

On the principle of 'know your enemy' we spent a lot of effort in identifying barriers to the use of FP in industry:

- The perception that choosing a functional approach makes the project more risky, simply by the mere fact that it is different.
- Managers (especially middle managers) are a significant barrier; in intel managers would allow programmers to use what they thought would be the best, but this is not the case in all businesses.
- Some thought that programmers could be a barrier; others that a decent C programmer could be taught to be a competent functional programmer in 4 weeks; I guess it depends upon the level of sophistication required.
- Too many factors go against functional languages: the only way that they will be used is when there's no alternative (as when the other ways of solving the problem have failed) – just like the case of alternative medicine?
- Concrete syntax can be a substantial barrier: if one cannot "see through" the concrete syntax to the ideas beneath then it is very difficult indeed to program effectively in the language. [Some advantages are discussed below.]
- The absence of the right books: there's more discussion of this later.
- There will be no breakthrough without a big company behind the language; the examples of C#, Erlang and Java were cited.
- There are problems in putting programmers interested in working with FLs in contact with the hirers who want to find them.

Why it is getting better

There's some reason to be hopeful, however;

- We're now at the point where people who know about FP are at a high enough level to make important decisions.
- Small companies can be the engine for growth. This is the case in Sweden, where there's a strong (and growing) group of small companies using Erlang.
- The intel example: managers in intel have nothing in principle against a functional language being used on a project.
- Syntax: if syntax is modified to look more like C, then people will use the language; in other words, the barriers may well be superficial, rather than semantic.
- Can move things forward by setting up job site or sites, for vacancies and resumes.

Learning Functional Programming

A thread of the discussion in both workshops talked about resources for learning functional programming. Whilst there are a number of introductory texts for each of the main languages, other books are non-existent. Specifically missing are

- More books on more advanced topics, either in specific languages, or for FP in general; a good reference, but not ideal is the series of 'Advanced Functional Programming' Springer Lecture Notes. These might cover, in the case of Haskell, things like performance estimation, designing using monad transformers, and generally 'cool stuff' in FP and so forth.
- Books which will introduce functional programming, and its particular idioms (in specific languages), to the experienced programmer in C, C++ and so on. This would be something that would naturally fit into the O'Reilly series.
- Books on advanced academic topics, such as fault tolerant distributed computing, using a functional language, in Erlang for instance.

If books are not around, what do newcomers use? Haskell IRC and various mailing lists are seen as valuable, and there was also the observation that 'instead of documentation there are papers'. This might not be ideal, but the papers are often readable.

Other communities have managed to build large bodies of programmer-oriented resources on the web. One example for Perl is http://www.perlmonks.org/ which has a vast range of tutorials on introductory through to advanced topics;

Particular Languages versus Functional Programming

Whilst we take it for granted that experience of functional programming in one of Erlang, Haskell, ML or scheme will be transferable to another functional language, we arguably hurt ourselves as a community by keeping the differences between different languages. There was a strong plea in the FDPE discussion that we do more to draw together and compromise on particular features in favour of overall visibility.

Organising ourselves

One way of building interest is by making things available as widely as possible. In particular, libraries of code are popular with everyone. Java has this, there is CPAN, the Comprehensive Perl Archive Network ' the gateway to all things Perl' and the OTP Library for Perl. This is a way of making programming in FP more attractive. If we can wrap useful libraries in FFIs, then there's a chance of getting imperative programmers using our libraries: this is the case with the FL libraries at intel.

Particularly for commercial users there is a need for stability. In the case of Haskell, some notion of “Stable Haskell” or “Industrial Haskell”, which formalises a small set of extensions to Haskell 98 (such as the FFI) which can be agreed to be stable through developments in the medium term.Languages are getting integrated into tools: Haskell is integrated into Visual Studio; other languages are bound into eclipse,

Meetings where users and potential users can hear about what each other are doing give a means of strengthening communities. This certainly works with Erlang, which has two annual meetings: the Workshop, with an academic focus, and the User Conference which looks at the products being developed.

John Launchbury offered to start building a Functional Programming Community web site and general resource, which can keep information about jobs, about 'good news' stories about FP and so on.

Consortia have formed or have been discussed for Erlang, Haskell and OCaml. The purpose of these is to give continuity of support, with the hope that cash will be raised from selling commercial licenses and support.

Functional programming by stealth

Perhaps we make most progress by stealth. One mechanism is through an initiative like Model-Driven Architecture (http://www.omg.org/mda/) which talks about building systems by modelling them first, and then transforming the results into implementations on particular paltforms. This sounds familiar to functional prorammers.

Another way is by 'infecting the brains" of language designers (C# has a lot of functional influences), DSL people and the people who write W3C standards who make a lot of the 'declarative' approach of standards like SVG and SMIL.

The cloud and the sliver lining

Paul Hudak exactly captured our situation in reminding the discussion that over the last fifteen years he'd had many discussions, with some of the same people. about these issues and returning to the same themes. Nevertheless, we should be upbeat. Looking at the example of darcs (http://abridgegame.org/darcs/), he reminded us that it was inconceivable for there to be a fully-fledged version control system written in Haskell even a few years ago.

References

ErlCons] Erlang Training and Consulting Ltd. http://www.erlang-consulting.com/

[Grundy] http://www.intel.com/technology/techresearch/people/bios/grundy_j.htm

[Haskerl] A note on the Haskerl extension to Haskell, Will Partain, 1 April 1993. http://www.dcs.gla.ac.uk/~partain/haskerl.html

[MDA] Model Driven Architecture. Object Management Group. http://www.omg.org/mda/