Commercial Users of Functional Programming (CUFP) 2006

A Report by John Hughes and Kathleen Fisher

21st September, 2006

The Commercial Users of Functional Programming workshop, held in association with ICFP, attracted a record turn-out this year: 57 attendees came to listen to 9 speakers talk about commercial uses of Scheme, Erlang, Reflect, O'Caml, and Haskell. There was a general sense of optimism at the meeting, reflected in the number of attendees and the announcements by several of the speakers that their companies were actively recruiting. One interesting development was the emergence of functional programming in the financial sector. Concerns for the future included the need to overcome conservatism on the part of companies reluctant to use non-mainstream technologies and to raise the profile of functional programming with students (and their parents)! In the rest of this report, we describe the various talks, with emphasis on the impact of functional programming in the "real world."

Clifford Beshers described the adoption of Haskell at Linspire. Linspire aims to build a Linux version suitable for the consumer market, by automating the job of the system manager. Linspire switched recently from O'Caml to Haskell, citing readability, classes, the IO monad, and more complete libraries as reasons. So far CGI scripts and a package autobuilder have been implemented in Haskell. Static typing, and the use of purely functional data-structures on disk, have helped make for reliable code. Performance has been generally good, although Parsec parsers proved to be slow on 20MB files–a problem that was solved using Data.ByteString. Space leaks have not been a problem, perhaps because frequent IO keeps programs strict. Belying the myth that functional programmers are hard to recruit, Linspire's announcement that they were adopting Haskell generated 5-10 resumes, despite explicitly stating that they were not recruiting. Quotable quote: "When we code in Haskell, we go home on time".

Steve Sims talked about how Reactive Systems, Inc. uses SML, both in implementing their Reactis product and also in their internal business processes, including the company website, intranet, and regression infrastructure. Reactis is a testing and validation package for models of embedded systems developed using Mathworks. A typical example would be a cruise control system on a car. Reactive Systems, which was founded in 1999, has roughly 35 customers from more than ten countries: 29 from the automotive industry, five from aerospace,

and one from heavy equipment. One of the reasons SML has worked well for Reactive Systems is that Reactis requires interpreting both Mathworks Simulink and Stateflow languages as well as C, and SML's support for recursive datatypes and pattern matching facilitates this task. Steve commented that exceptions were useful because customer bug reports almost always contained some information. Steve also cited SML's strong type checking, garbage collection, and module system as relevant to their success, as well as the ability to use imperative features when necessary. Based on comparisons with a similar system developed at Mathworks in C++, Steve estimated SML provided a 5-10 times productivity benefit. He concluded with the observation that SML has been a key factor in recruiting and retaining highly productive employees. No programmer has left the company despite the fact that all have had offers to do so; all of them cited programming in SML as one of their reasons for staying.

Richard Cleis from the Starfire Optical Range, an Air Force Research Laboratory, described how he and others have used Scheme to control the telescope systems at the facility. Such control involves complex calculations to compensate for atmospheric distortions. Originally, the systems were controlled using C code. However, a disappointing 12-hour effort to program the telescopes to track the space shuttle Columbia's landing yielded a single picture. This failure led them to consider alternative languages because they just couldn't write C code fast enough. They now use DrScheme and MzScheme to control the telescopes, interfacing with legacy C code. They have convinced others to use S-expressions to represent data. The fact that Scheme supports hot-code modification is an advantage. Garbage collection times of 10ms can be a problem, but memory requirements are not. To date, they have not been able to persuade management to hire Scheme programmers.

Yaron Minsky from Jane Street Capital talked about the company's use of O'Caml. Jane Street Capital is a small, proprietary trading firm based in New York. They don't have customers; instead, they make money by trading. Originally, they used a combination of Excel and Visual Basic to implement their analyses. Because of the importance of the analyses to their business, they really care about reliability. Excel is a great tool in terms of rapid deployment and it interoperates with VB very well. Unfortunately, on the spreadsheet side, it is easy to see the data but not the logic, while the code shows the logic but not the data, making it difficult to reason about the system. They tried to rewrite the system in C#, but they found the language too verbose and complex, and the non-technical people really complained about inheritance. Yaron had used O'Caml during his Ph.D. studies at Cornell, and had been using it as "throw-away" code at Jane Street. In 2005, the company made a management decision to switch to O'Caml. This switch has been a success: within six months, they had rewritten many of the key systems, obtaining better performance in the process. The code is much shorter and more readable, which is important as the partners in the company review the code personally. Because they can understand the code more easily, they can now use more complex algorithms. They use functional programming as a recruitment tool, observing that it is hard to hire good Java programmers, but easy to hire good functional programmers.

They are currently hiring.

Howard Mansell talked about Haskell programming at Credit Suisse in New York. This group's business is derivative trading for clients (a stock option is a simple case of a derivative). Valuing complex derivatives is computationally costly, requiring nightly runs on thousands of CPUs. There is a real competitive advantage in being able to build models quickly, since some exotic derivatives may only be traded 10-100 times in total. Previously models were built using Excel, with heavy computations delegated to C++ plug-ins. Most of the Excel code has now been replaced by Haskell, with Excel just providing the user interface (which in turn is generated by DSELs). The work has been done by Lennart Augustsson and Gabriele Keller, but Credit Suisse need more Haskell programmers–they're hiring.

Roope Kaivola described how Intel uses an internally developed lazy functional language called reFlect to formally verify chip designs. He explained how Moore's law has led to larger structures on chips with greatly increased complexity. A typical CPU design has more than 500 design engineers and requires roughly two years from design start to the first silicon fabrication. A full chip specification is one to ten million lines of code and costs roughly five billion dollars from R&D to fabrication, so Intel needs to have high volume sales to recover the initial investment. However, high volume increases the financial risk from errors. Testing is not enough to mitigate against this risk, because the size of the state space means that exhaustive testing is impossible. As a result, most CPU designs have a formal verification team. Such verification has been a huge success, proving designs correct with respect to models such as the IEEE floating point standard and finding many high quality bugs in the process. Engineers at Intel use the reFlect language to carry out this verification. ReFlect is an interpreted "ML-like" lazy functional language that supports scripting, rapid prototyping, and the development of libraries and formal tools. Engineers write system specifications, verification strategies, and analysis code in reFlect. The language is customized for formal verification, including support for binary decision diagrams (BDDs) and symbolic simulation/trajectory evaluation. It supports reflection to interface with a theorem prover to reason about the code itself. The language has made the group that uses it one of the most effective formal verification groups at Intel. They report that the laziness of the language is essential in avoiding unnecessary computations but that it confuses novice users.

Rishiyur Nikhil talked about Bluespec Inc, which offers System Verilog tools implemented in about 90KLOC of Haskell. Bluespec's tools are based on MIT research in the 90s into generating hardware from term rewriting systems; using Bluespec's "rules" complex hardware can be described at a high level, improving productivity and reducing errors. However, it is not easy to explain briefly why Bluespec's tool is better than the competition: many vendors claim to improve productivity. Selling the tool requires short, punchy examples to convince old hands that there is a better way than writing RTL by hand. Bluespec have made four sales, and many more trials are under way.

Garret Morris talked about Haskell applications at Aetion, which is a defence

contractor offering AI applications based on Bayesian nets. Rapidly changing priorities (as a result, for example, of the invasion of Iraq) make it important to minimize the code impact of changes, which suits Haskell well. Aetion have developed three main projects in Haskell, all successful. Haskell's concise code was perhaps most important for rewriting: it made it practicable to throw away old code occasionally. DSELs allowed the AI to be specified very declaratively. Less successful was the decision to link Haskell to a GUI in Java: the need to serialize data made it impossible to use functions as representations, which was unfortunate. Problems include getting libraries to work together, patchy Windows support, and a need for more debugging tools.

Erik Stenman described how the Swedish company Kreditor used Erlang to build a system to serve as a mediator between on-line businesses and their customers. The system allows customers to obtain merchandise before paying for it and allows businesses to outsource invoicing and other standard business practices. Of course, for such a system to be successful, it must be secure, robust, error-free, and have high availability. Erik discussed how Erlang allowed Kreditor to build such a system quickly and economically, requiring less than four months and a hundred thousand dollar investment to get the initial system working. Their main competitor went bankrupt after burning through ninety million Swedish Kroner building a similar system with .net and PHP.

Finally, there was a discussion on "Making the case for new technology." Telling customers "we're smart, and our functional language is great" is not a successful approach! We tend to show customers the things that impress us, which is a mistake. It is important to show humility, and to explain benefits in the customer's terms. What can be done to help individuals persuade managers to let them use functional languages? Better and more libraries perhaps? Explaining that a system being built in a functional language is "just a prototype" may allow a project to go forward. If it succeeds, it can become the deployed system. Interoperability is important: if others can call our code from their favourite language, then our work will be more valued. If companies using functional languages could release components that are not core IP as open source, that will help everyone solve the library-creation problem.

The next CUFP will take place on October 4th, 2007 in Freiburg, Germany, again co-located with ICFP.