Links: Linking Theory to Practice for the Web Case for Support

Philip Wadler
University of Edinburgh

Summary

e-Commerce, e-Government, e-Science — the coining of such words reflects the growing importance of the World Wide Web in all aspects of our lives. Consumer spending on the web in Britain exceeds £12 billion a year.

A typical web program involves three tiers. The front end is the browser running on your computer. The middle tier is a server, executing the logic that controls your interaction with the web site. The back end is a database, providing the information you wish to access, such as a catalog of items for purchase, a collection of government records, or a store of scientific data.

The programmer must master a myriad of languages: the logic is written in a mixture of Java, Perl, PHP, and Python; the forms in HTML, XML, and JavaScript; and the queries are written in SQL or XQuery. There is no easy way to link these — to be sure that a form in HTML or a query in SQL produces data of a type that the logic in Java expects. This is called the *impedance mismatch* problem.

Links will solve the impedance mismatch problem by providing a single language for all three tiers. The system will be responsible for distributing tasks among tiers and translating into suitable languages for each tier — for instance, translating part of a program into JavaScript to run in the browser, Java to run on the server, and SQL to run on the database. Links will incorporate ideas proven in other programming languages: support for database programming from Kleisli, for XML programming from XDuce, for web interaction from PLT Scheme, and for distribution from Erlang. It will be developed by a consortium, as were ML and Haskell. Like all of these languages, it will be functional.

Support is sought for one PhD studentship and one RA for three years, costing £292K for 36 months. Letters of support are attached from two commercial web firms, LShift and Run Deep.

I. Previous Track Record

Applicants

Philip Wadler (Principal Investigator) Wadler is Professor of Theoretical Computer Science in the School of Informatics of the University of Edinburgh. His career spans academia and industry, with degrees from Stanford and Carnegie-Mellon, posts at Oxford, Glasgow, Bell Labs, and Avaya Labs, and guest professorships in Sydney, Copenhagen, and Paris.

Wadler possesses an unmatched track record for converting theory into practice, with major contributions to XQuery, Java, and Haskell. *XQuery* is the forthcoming W3C standard for querying XML data (think of it as ‘SQL for XML’). Wadler, working with Fernández and Siméon, put forward proposals for the syntax, core algebra, formal semantics, and static type system for XQuery, now incorporated into the standard. *Java 5*, released in September 2004, contains generic types — the first change to the core language since 1997. Wadler, together with Bracha, Odersky, and Stoutamire, devised GJ (Generic Java), the basis for Sun’s design. *Haskell* is the most widely used language for lazy functional programming. Wadler served as first editor of the Haskell report, and contributed Haskell’s two main language innovations, type classes (joint with Blott) and monads (joint with Peyton Jones); the latter won the Most Influential POPL Paper Award (2003 for 1993). Languages influenced by Haskell include Isabelle (theorem proving), Mercury (logic programming), Curry and Gödel (logic-functional programming), and Hal (constraint programming).

Wadler ranks 70 on Citeseer’s list of Most-Cited Authors in Computer Science (May 2005). He is a Fellow of the Royal Society of Edinburgh and a Royal Society Wolfson Merit Fellow. Wadler co-founded the *Journal of Functional Programming*, serving as Editor-in-Chief until 2004. He sits on the Executive Committee of ACM SIGPLAN, and served as program chair for ICFP (2000), Plan-X (2002), PADL (2003), Erlang (2004), and FOOL (2005). Invited speaking engagements include POPL (Albuquerque, 1992), ILPS (Portland, 1995), Software Reuse (Victoria, 1998), Domain Specific Languages (Austin, 1999), TCS (New Delhi, 2000), ICDT (London, 2001), VLDB (Rome, 2001), FLOPS (Aizu, 2002), FGUC (London, 2004), and RTA (Nara, 2005).

Peter Buneman (Co-investigator) Buneman is Professor of Database Systems in the School of Informatics at the University of Edinburgh. His work in computer science has focused mainly on databases and programming languages, specifically: active databases, database semantics, approximate information, query languages, types for databases, data integration, bioinformatics and semistructured data — an area in which he has co-authored a book. He has recently worked on issues associated with scientific databases such as data provenance, archiving and annotation. He has served on numerous programme committees, editorial boards and working groups, and has been programme chair for ACM SIGMOD, ACM PODS and ICDT. He is a fellow of the Royal Society of Edinburgh, a fellow of the ACM and the recipient of a Royal Society Wolfson Merit Award. He is research director of the UK Digital Curation Centre.

Before joining the University of Edinburgh, Peter Buneman was a Professor at the Department of Computer and Information Science at the University of Pennsylvania. He received his undergraduate degree from Cambridge University, his graduate degree from the University of Warwick, and he did postdoctoral work at the University of Edinburgh. He has held visiting positions at the University of Glasgow, Imperial College, Kyoto University and INRIA. In addition to computer science he has made contributions to graph theory and to the mathematics of phylogeny.

Don Sannella (Co-investigator) Sannella received his PhD in Computer Science from the University of Edinburgh in 1982 where he has worked ever since, being appointed Professor in 1998. His research interests include the design of algebraic specification languages and functional programming languages, foundations of algebraic specification and formal software development, and more recently global computation and resource certification for mobile code. He has published more than 60 papers in journals and international conferences and has held a series of grants since 1985 for research projects in the area of verification and formal development of programs including an EPSRC Advanced Fellowship in 1992–1997 and an RSE/SOEID Fellowship during 1998. He is editor-in-chief of *Theoretical Computer Science* (responsible for part B: Logic, Semantics and Theory of Programming), founder, former chairman and current treasurer of the ETAPS conference series, and is on the Council of the European Association for Theoretical Computer Science. He is Director of the Informatics Graduate School at Edinburgh University.

Ian Stark (Co-investigator) Stark is a Lecturer in Computer Science in the School of Informatics at the University of Edinburgh, where he also holds an EPSRC Advanced Research Fellowship. He received his PhD from the University of Cambridge in 1995, and subsequently held a Marie Curie Fellowship in Pisa and worked in the BRICS research institute at the University of Aarhus. His research interests are in foundational models of programming languages and concurrency; in particular types, operational reasoning and mobile processes. He has nineteen published papers in journals and conferences. He has served on the programme committee for the international conferences TACS and MFPS. Previously he worked on the EC-funded EuroFoCS, CLICS-II and APPSEM projects and he has set up a .NET laboratory in Edinburgh with funding from Microsoft. As well as his research fellowship, he holds an EPSRC grant on Reasoning about Names and Identity in Programming Languages, and is Edinburgh site leader for the EC-funded Mobius project on secure mobile code.

Sam Lindley (Research Assistant) Sam Lindley obtained a BA in computation from the University of Oxford in 1998. From 1996 until 2000 he worked for London-based web agency Red Snapper, where he helped design and implement their content management system. This has been used successfully to produce and maintain websites for a range of blue-chip clients. In 2000 he began his PhD in Computer Science at the University of Edinburgh. This was completed at the beginning of 2005. During his PhD, in 2002 he obtained a three-month Marie Curie Fellowship to visit the University of Aarhus, and at the end of 2003 he took up a three-month postgraduate internship at Microsoft Research Cambridge. His research interests include programming language semantics and compilers for functional programming languages. Since completing his PhD he has been working for Red Snapper as a freelance consultant. He has been developing a system for reverse-engineering existing websites for import into their content management system.

Host Organisation

The University of Edinburgh School of Informatics holds top ratings for both teaching and research, including the only 5*A research rating for Computer Science in Britain. The School is organized into institutes, many of which will contribute to Links.

Laboratory for Foundations of Computer Science LFCS studies the theory underlying programming languages. Its members include a Fellow of the Royal Society and four EPSRC Advanced Research Fellows. Wadler's contributions to Haskell, Java, and XQuery draw heavily on the work of LFCS, and so will the design of Links. Particularly relevant are Buneman's work on semi-structured data and XML databases, Plotkin's work on monads, and Sannella's work on safe mobile code.

Institute for Communicating and Collaborative Systems ICCS plays a leading role in developing and applying XML standards. Particularly relevant is Thompson's work on XML Schema and XSLT.

Centre for Intelligent Systems and their Applications CISA expertise includes web agents and the semantic web. Particularly relevant is Robertson's work on programming languages for expressing web protocols.

Edinburgh also hosts two national institutes and an enterprise unit designed to facilitate communication between the computing and science communities, and between academics and industry. They provide perfect venues to disseminate Links to academic and industrial communities and to expose Links to new application areas.

The National e-Science Centre NeSC is developing the infrastructure to enable scientists to share petabytes of data and teraflops of computation over the Internet, initially funded at £5.5M. NeSC expertise in web services and the Grid will be particularly helpful, the former because it is crucial for Links to interface smoothly with web services, the latter because it will help Links broaden its concerns to wider aspects of global computing.

The National Digital Curation Centre NDCC is a national resource for archiving and accessing digital data, initially funded at £3.2M. Buneman of LFCS is Director of Research. NDCC expertise in XML and web publishing will help ground Links in real applications.

Edinburgh Research and Innovation ERI provides advice on exploiting the commercial potential of research, and has been instrumental in finding industrial collaborators for Links.

II. Proposed Research

Background and Programme

Ask a room of researchers whether they have purchased goods or services online. Every hand will go up. Now ask whether they have ever *failed* to complete some online purchase because of a fault in the web site software. Every hand will go up again.

Consumer spending on the web exceeded £12 billion per year in Britain, growing nearly 20 times faster than traditional retail [28]. Yet web programming remains an immature art, expensive and error prone. Even major sites like Orbitz, Apple, Continental, Hertz, and Microsoft suffer from fundamental problems [21].

Not only business, but healthcare, government, education, and science are being transformed by the web. Mobile phones, wireless computing, petabyte data resources, remote sensors, and intelligent devices extend the scope and nature of the Internet. Each system must interact with others, ranging from financial institutions to malicious viruses. One recent policy document refers to the emergence of a Global Ubiquitous Computer, the largest engineered artefact in human history [40].

We propose to design and implement Links, a programming language for web applications and web services. Links will apply techniques pioneered in other languages in order to increase the capability and reliability of web sites while decreasing the cost of development.

A quarter century ago, Burstall, MacQueen, and Sannella at Edinburgh introduced an influential programming language, Hope [12]. Hope was named for Hope Park Square, located near the University on the Meadows. Links is named for the Bruntsfield Links, located at the other end of the the Meadows and site of the world's first public golf course.

Three tiers A typical web system is organized in three tiers, each running on a separate computer. (See figure at top of this page.) Logic on the middle-tier server generates forms to send to a front-end browser and queries to send to a back-end database. The programmer must master a myriad of languages: the logic is written in a mixture of Java, Perl, PHP, and Python; the forms in HTML, XML, and JavaScript; and the queries are written in SQL or XQuery. There is no easy way to link these — to be sure that a form in HTML or a query in SQL produces data of a type that the logic in Java expects. This is called the *impedance mismatch* problem.

Links will solve the impedance mismatch problem by providing a single language for all three tiers. The system will be responsible for distributing tasks among tiers and translating into suitable languages for each tier — for instance, translating parts of a program into JavaScript to run in the browser and SQL to run on the database. Links will incorporate ideas proven in other programming languages: support for database programming from Kleisli, for XML programming from XDuce, for web interaction from PLT Scheme, and for distribution from Erlang. It will be developed by a consortium, as were ML and Haskell. Like all of these languages, it will be functional.

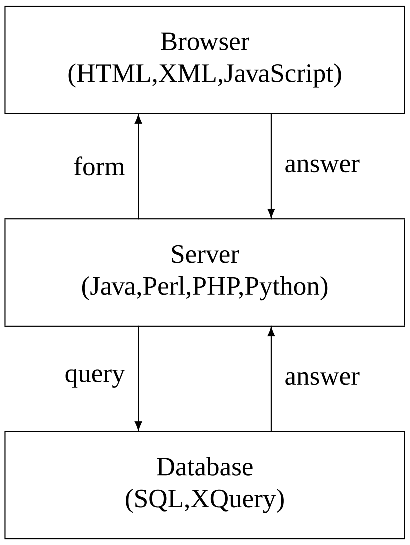


Figure 1. Three-tier model

Database programming Kleisli was developed by Buneman, Wong, and others, based on comprehensions for database query languages (which in turn are based on Wadler's work on monads) [11, 29, 48]. Kleisli has been used to solve database integration problems that otherwise have proven intractable, has been applied to a range of problems in bioinformatics, and is sold as a commercial product. Other database systems that apply comprehensions for querying include Erlang's Mnesia [47] and work by Grust [25].

Links will include a comprehension feature and adopt techniques from Kleisli and Mnesia to compile these into efficient access to SQL and XQuery databases. Programmers writing middle-tier logic in Links need not learn SQL or XQuery: instead they express queries with comprehensions, from which the Links compiler extracts optimized SQL and XQuery to send to back-end databases.

XML programming XDuce was developed by Pierce, Hosoya, and others, based on regular expressions types for XML [26, 27]. XML types were further developed by Wadler and others in XQuery [16, 41, 49], by Schwartzbach and others in Bigwig and JWIG [5, 10, 13, 39], by Castagna and others in CDuce [8], and by Pierce and others in Xtatic [24]; a survey of type systems for XML was published by Møller and Schwartbach [32].

Links will support XML processing using regular expression patterns and types, based on techniques developed for XDuce and other languages. As a special case of this, string pattern matching will be supported as in Perl or Python, with regular expressions providing static types for strings [14, 24]. HTML will be supported as a dialect of XML.

Web interaction Programming web interfaces is tricky and prone to error. For example, Felleisen and others [22] document a problem with comparison shopping on Orbitz: a shopper clicks on a flight and sees its price and details appear in a second window, returns to the first window and clicks on another flight and sees its price and details appear in a third window, returns to the second window and clicks submit — and ends up buying the flight displayed in the *third* window, not the second. They observed similar problems on web sites for Apple, Continental, Hertz, and Microsoft, indicating that this is a fundamental problem rather than a rare bug.

The notion of *continuation* from functional programming has been applied by a number of researchers to improve interaction with a web client, including Quiennec [38], Graham [20] (in a commercial system sold to Yahoo and widely used for building web stores), Felleisen and others [21, 22, 23], and Thiemann [43].

Links will incorporate language features to support web interaction, using *continuations* to capture the state of the interaction, and *types* to ensure consistency between the client and the server. Felleisen and others [21] have proposed a methodology for developing web applications in Java by hand, based on *continuation passing style* and *defunctionalization*. We will apply the same techniques to compile Links programs. This will result in *scalable* programs that maintain session state in the client rather than the server. Only a few commercial web tools (such as ASP.NET) produce code that is scalable in this sense; many commercial web tools (like J2EE) and most research web tools (including current releases of PLT Scheme [23] and Mozart QHTML [15]) are not scalable in this sense.

Rich clients Increasingly, web applications designers are migrating work into the browser. "Rich client" systems include Google Mail, Google Maps, and the Mozilla Amazon Browser, using a new style of interaction recently dubbed "*Ajax*" [19]. A special-purpose language for checking values in the clients was developed by Schwartzbach and others for Bigwig [4], and techniques for transforming code to move computation into clients have been developed by Neubauer and Thiemann in Haskell [33].

Compiling from Links into JavaScript (or other languages available on a client, such as Java or Flash) is a straightforward application of compiling technology. Research is required to determine to what extent computation can be automatically migrated from server to client and to what extent this needs to be under the control of the programmer. In some cases, Links may compile two versions of the code from a single source, one that migrates computation into JavaScript on the client (for improved response), and one that retains computation on the server (for clients that do not support JavaScript, or that have disabled it). In some cases, the JavaScript version may offer better interaction than the server version; research is required into how a single source might specify two levels of interaction.

Concurrency and distribution The functional language Erlang [1, 2] is used in Ericsson's AXD301 ATM phone switch. It is the market leader in its segment, used by British Telecom in their telephony and data backbone, the world's largest telephony over ATM network. The AXD301 has 99.9999999% (nine 9s) reliability, one of the most reliable switches ever made. It contains 1.7 million lines of Erlang, making it the largest functional program ever written. Erlang is used in a number of other market leading products, including Ericsson's GPRS and Alteon's Accelerator.

Traditional concurrent programs use operating system threads (up to 2000 on a single processor) and shared data protected by semaphore. Erlang supports lightweight processes (up to 30,000 on a single processor) and "share nothing" concurrency where the only way processes can exchange data is by explicit message passing. Avoiding sharing eases both distribution and reliability. Because there is no shared data, it is relatively easy to arrange to move processes to another machine, to support distribution or scaling. Reliability is increased by structuring programs as "worker" and "observer" processes — the observer monitors the worker and performs error recovery if something goes wrong. The worker and observer can run on the same machine, or on separate machines to support fault-tolerance. (Another language that supports lightweight processes and "share nothing" concurrency is Mozart [15, 44], but it does not offer support for reliability through workers and observers.)

Erlang's OTP system is structured around a set of design principles that support the creation of distributed and reliable systems. Using this framework, a programmer can code a naive, sequential server, and then instantiate it as a reliable, replicated server. This is far in advance over the support provided by Sun's Java and its J2EE infrastructure, or Microsoft's C$^\sharp$ and its .NET infrastructure.

Links will adopt the key features of the distribution model of Erlang, and successful Erlang design principles from the published open source implementation and libraries. Wadler designed one of the first type systems for Erlang [31], and served as program chair of the 2004 Erlang workshop.

Links differs from Erlang in that it is statically typed. Numerous process algebras offer a foundational basis for distribution with types. Links will base its design on the Join calculus [17], as successfully used in JoCaml [18] and Polyphonic C$^\sharp$[6], and on Timber [34], a typed language with communicating objects inspired by Erlang.

Functional programming Links will be a typed, functional language. Query optimization in Kleisli, pattern matching in XDuce, continuation-based web interfaces in Scheme, and distribution in Erlang all work better with immutable values rather than with mutable objects. Types ensure consistency between forms in the browser, logic in the server, and queries on the database; and between the two ends of a channel in a distributed program. With the exception of Haskell, all the functional languages listed above are strict; Links will also be strict, with support for lazy evaluation where it is required.

We intend to take ideas from both the Haskell and ML communities. Monads in Haskell delimit the use of side effects, while ML makes it easy to insert a side effect at any point in a program. We hope to get the best of both worlds by permitting side effects at any point but using an effect type system to control their use, building on work relating monads and effects by Wadler and Thiemann [46]. Strict evaluation in ML is better suited to a distributed environment, but lazy evaluation in Haskell is useful for stream processing. We will use a strict language with built-in support for laziness, as suggested by Wadler, Taha, and MacQueen [45].

Related work Other languages for web programming include Xtatic [24], Scala [35, 36], Mozart [15, 44], SML.NET [7], F$^\sharp$[42], and C$^\omega$ (based on Polyphonic C$^\sharp$ [6] and Xen [9]). Like Links, these languages combine ideas from the XML, object-oriented, and functional programming communities, and Links will benefit from fruitful interactions with these efforts. However, none of these languages shares Links' objective of generating code for all three tiers of a web application from a single source — scripts for the front-end client, logic for the middle-tier server, and queries for the back-end database. We expect that providing a single, unified language to replace the current multiplicity of languages will be what attracts users to Links.

Consortium Dozens of new programming languages are introduced each year, and few of these see widespread uptake. To ensure that Links develops a significant user community, we will form a consortium of research groups to develop and promote Links. This strategy has proved effective for programming languages since Algol 60, and was used to good effect by ML and Haskell. The Links consortium will build on these successes, drawing on members of both the ML and Haskell communities. Unlike the previous ML and Haskell efforts, Links is aimed at exploiting the power of functional programming in specific application areas. The goal here is ambitious: not to be the next ML or Haskell, but to be the next Python or Java.

A workshop on Links took place on April 2005, colocated with ETAPS. The call for participation was a collaborative effort, signed by Xavier Leroy (INRIA, Paris), Simon Peyton Jones (Microsoft, Cambridge), Benjamin Pierce (University of Pennsylvania), and Philip Wadler. Despite minimal publicity, forty researchers attended, and many others registered an interest in the project. An initial design team has been formed, consisting of Leroy, Pierce, Martin Odersky (EPFL, Lausanne), and Wadler.

Links has already attracted attention from the developer community (see discussion on Lambda the Ultimate or comp.lang.functional). By seeking early feedback from this community we hope to better suit the design of Links to its target audience. We will use weblogs, web surveys, and web tools such as Bugzilla to seek feedback on design choices — particularly on choice of syntax, which seems to be inordinately important to the success of a language.

Advisory committee We intend to form an advisory committee from members in industry concerned with web development. The committee will explain problems the developer community finds particularly important, give feedback on the Links design, and suggest realistic test cases. For instance, our attention was drawn to the scalability problem by interactions with this community. Letters of support are attached from two commercial web firms, LShift and Run Deep.

Programme and methodology

Current status We already have a partial design for Links, tackling some of the points above, together with a prototype.

Gilles Dubochet, an MSc student from Lausanne sent by Odersky, worked under Wadler's supervision in Edinburgh, August 2004–February 2005. He reimplemented much of the Kleisli system, working from published descriptions. Some explanations are difficult to get in print, so we were greatly aided by a visit from Limsoon Wong, a principal implementor of the original Kleisli system. The original Kleisli system used a non-standard approach to record and sum types. Dubochet's systematically uses row types for records and sum, based on a tutorial description by Pottier and Rémy [37].

Jeremy Yallop, a PhD student, began work under Wadler's supervision in September 2004. He first developed a prototype compiler from Links to JavaScript, then integrated this with Dubochet's code. We now have a rudimentary prototype that can generate JavaScript when all code is to run in the client and can partition code to run in the client or server as appropriate.

As noted above, recently a design team consisting of Leroy, Odersky, Pierce, and Wadler has been formed. The design team will work in close consultation with others, including Yallop at Edinburgh and Frisch at INRIA.

Research plan Yallop will continue to work on Links through his second and third year (he is referred to as PhD 1 in the plan below). The proposal calls for funding one additional PhD student (referred to as PhD 2) and one RA. We have lined up likely candidates for these posts, Ezra Cooper for the PhD, and Sam Lindley for the RA.

The tasks listed below are diagrammed in the workplan at the end of this submission. We expect to adapt the plan flexibly as work proceeds.

Interaction. What form of interaction model should Links support? JavaScript treats DOM trees as mutable structures. We have devised a simple model where DOM trees are immutable. This requires additional computation but preliminary work suggests this has adequate performance and expressiveness. We need to perform a more thorough evaluation, and to extend the model to deal with Ajax-style interaction. For a case study, we may attempt to re-build the Google Maps interface in Links. [PhD 1, work currently in progress.]

Partitioning and security. How should computation be distributed in a web application? How does this interact with security? The prototype allocates as much work to the JavaScript front end and the SQL back end as possible, leaving relatively little work in the middle-tier server. We need to examine whether automatic partitioning across the three tiers is adequate, or whether finer control is required. We also need to examine the relationship between partitioning and security, since data sent to the front end may be subject to tampering. Security issues may be dealt with by automatic encryption or by giving finer user control over data migration. [PhD 1.]

Concurrency and distribution. Can the concurrency and distribution model of Erlang be adapted to Links? We need to incorporate support for Erlang's lightweight processes into the Links interpreter and the JavaScript front end, including devising a suitable type system, perhaps basing this on ideas from JoCaml or Timber. The interaction model with immutable DOM trees, mentioned above, should fit well with concurrency. We need to consider whether concurrency should alter the JavaScript interaction model, which currently depends heavily on callbacks. [PhD 2.]

Databases and transactions. Can database query and update be integrated into Links? Our prototype, like Kleisli, is a special purpose language without general recursive structures or modules. We need to add support for recursion and modules. Kleisli achieved acceptable performance for bioinformatics applications, we need to evaluate performance on server-intensive web applications. SQL queries are often tuned to work with vendor-specific implementations; can we tune the queries generated by Links, perhaps by providing generation templates? Kleisli only generated queries. Our prototype handles insertion and deletion of relational tuples, we need to evaluate the adequacy of our design, and extend it to handle transactions. [PhD 2.]

XML support. How should previous work on XML typing and manipulation be adapted to Links? The prototype supports creation of XML data, but treats all XML data as having a single type, "xml", and has limited support for manipulation. We need to support regular-expression typing for XML, one question being how to integrate this with Hindley-Milner style type inference. (We may build on recent, unpublished work by Frisch in this area.) And we need to add constructs for manipulating XML, one question being whether to use XDuce-style pattern matching, XPath-style selection, Hosoya's recent work on XML filters, or some combination. Our prototype only generates queries in SQL, and should be extended to also generate XQuery. [RA.]

Web services and interlanguage working. How should Links integrate with other systems? We need to support web services, including import and export of XML Schema and WSDL, and support for SOAP. Experience in the functional programming community shows the importance of interlanguage working, and we plan to implement an interface with Java or $C^\sharp$. The type system of Links will need to be adjusted to integrate well with object-oriented languages, perhaps by adapting ideas from O'Caml or Scala. [RA.]

Development The RA will be in overall charge of the structure of the compiler, while the PhD students will build prototypes for new features. To enable three developers (or more) to work flexibly and reliably in concert, we will adopt techniques from the Extreme Programming and Agile Processes community [3]. We will use pair programming to ensure all programmers are familiar with all code; unit test to increase reliability, with writing of unit tests preceding rather than following writing of code; and frequent refactoring to keep the code easy to maintain. We will use storycards to focus on what feature to implement next, and a daily fifteen minutes stand-up meeting to facilitate communication. We will make a major release of a new compiler once every six months. We will review and adjust our use of agile methods as the project continues. [RA and PhDs, throughout.]

Case studies We will need to implement case studies to evaluate the language design. One candidate is the Java Petshop (or TPC-W), which has been used to compare Java, $C^\sharp$, and other web frameworks. Another is to see if we can emulate in Links the design of Topsl, a domain-specific language for on-line surveys embedded in Scheme [30]. We expect to solicit additional candidates for case studies from our advisory board. [RA and PhDs, throughout.]

Library A organized, documented, and large library is one factor contributing to the success of languages such as Perl, PHP, and Python. The RA will maintain a library web site, encouraging our user community to develop and share libraries. Analogous to the well-known CPAN (Comprehensive Perl Archive Network), our library site will be called CLAN (Comprehensive Links Archive Network), giving a fortuitous Scottish slant. [RA, throughout.]

Write up, distribution and documentation Time is allocated at the end for write up of the dissertations, and for producing a final distribution with complete documentation.

Management Wadler will meet with each doctoral student for one hour each week. The entire project team will meet once each week, plus additional meetings as needed. All team members will record their progress in blogs, a strategy that has proved effective when trialed this year. At the end of each year an interim progress report will be produced, with each team member contributing a separate section. We will also aim to have a yearly retreat, to enable deeper review and to build team spirit.

Future plans The support sought here is minimal, in order to establish that our approach is sound. Establishing Links as a widely used language is an ambitious plan and will require additional support. As the work progresses, we expect to apply for additional grants, within the UK and from the EU, to test the design on larger applications (perhaps in e-Science), to build a debugger and profiler, and to integrate with an interactive development environment (such as Eclipse).

Relevance to Beneficiaries

Web applications are widely used in commerce, healthcare, government, education, and science. A programming language that allows one to develop web applications with increased productivity and greater reliability could have an enormous impact on all areas of society. Consumers, patients, citizens, students, and scientists will benefit from more reliable and more flexible web applications. Companies, hospitals, ministries, schools, and universities will benefit from increased productivity of web application development.

As a particular example, we expect Links to benefit e-Science and e-Research, which depend crucially on distribution and databases, two of the essential elements in Links. We have already been approached by members of the e-Science community interested in Links. With Amanda Clare of the Computational Biology group at Aberystwyth and Werner Dubitzky of the Bioinformatics Research Group at Ulster, we prepared a small travel grant proposal, “Exploring Links as a language for e-Science”, to facilitate communication between Links and the e-Science community. Note that the Links team was approached by the e-Science practitioners, not the reverse — they learned about Links from the announcement of the Links workshop, and it sounded like a match to their needs. We also expect Links to benefit from the location in Edinburgh of the National e-Science Centre.

We also expect a less direct benefit from Links. Britain has a long tradition of leading the world in developing programming language theory and in putting it into practice, beginning with Turing and extending through Strachey, Burstall, Milner, and Hoare. But as the software industry has grown its inertia has increased, and the gap between theory and practice has widened. Although Britain continues to produce the best programming language theorists in the world, that lead is no longer feeding through sufficiently to innovation in practice. We hope Links will take the lead in showing how to reduce this gap, and reopening the productive interchange between computing theory and industrial practice.

Dissemination and Exploitation

We will make the Links software available to the community, under an open-source license. Edinburgh Research and Innovation maintains a web site and management system to support open-source distribution of software. CLAN, the Comprehensive Links Archive Network, as described above, will act as a means for Links users to develop and share libraries.

The industrial advisory board and web community described above will be important avenues both for understanding the needs of the developer community and for disseminating the results to that community.

Developing Links as a consortium will further enhance the dissemination of the results. We have already organized one Links workshop, and expect to hold one Links workshop per year.

We will present research at workshops and conferences and publish in academic journals.

Justification of Resources

Manpower We request funding for one additional PhD studentship, and one Research Assistant on AR1A at point 5 (£23,643). We also request 25% of a computing officer on AD3 at point 3; 5% of a secretary on CN3 at point 3.

It is desirable to attract for the RA post someone with experience of web application development. This is a highly saleable skill, requiring the RA post to be funded at a competitive level.

The computing officer will be responsible for maintaining the web server for testing Links applications, a Links website acting as a source for documentation, code, and libraries (via CLAN), and for ensuring the School of Informatics computing environment supports implementations needed to act as a basis for Links and for comparison with Links, including Erlang, Haskell, O’Caml, PHP, and Apache.

The secretary will be responsible for coordinating meetings with our industrial advisors; coordinating meetings with academic partners in the UK, Europe, and worldwide; organizing workshops and meetings to promote Links; travel arrangements for conferences and workshops; and document preparation.

Travel We request support for a total of 12 visits of one–three days within the UK (e.g., Cambridge, London, Oxford, York) at £250 per trip; 10 visits of one week within Europe (e.g., Aarhus, Chalmers, Copenhagen, Freiburg, Lausanne, Paris) at £850 per trip; 8 visits of one–two weeks to the US (e.g., Harvard, Northeastern, Pennsylvania, Stanford, Yale), Australasia (e.g., Sydney, Kyoto, Tokyo, Singapore) at £1600 per trip; and 15 conference attendance fees of £350 each [1].

Equipment The School of Informatics has an integrated computing environment (DICE) that provides high reliability and comprehensive facilities. Faculty and students are supplied with DICE PCs by the school.

We request one desktop Linux PC running DICE for the RA, and one high-end PC to act as a web server, for use in testing the performance of Links.

Portable machines are necessary for presentations, demonstrations, and working effectively during travel. Apples running OS X provide the best usability. We request two Apple PowerBooks to be shared among the researchers.

We need to compare Links with competing software, much of which is available only under Windows. We request one Windows laptop to be shared among the researchers.

We require both laptops and desktops because we need to be both portable (presentations, travel) and fixed (development, server).

We request one large display screen for demonstrations; this can be attached to both desktops and laptops as required.

Consumables To facilitate comparison with competing systems, we request £1000 for commercial software of relevant systems (such as Visual Studio) and £500 for books describing such systems (which are not available from the University library).

We request £1000 each for organization of three workshops. We expect to organize some workshops in Edinburgh and some elsewhere, in order to emphasize that Links is a collaborative project with participation from a number of sites.

[1] To attend conferences such as: AiML, AMAST, APLAS, ATVA, BCTCS, CADE, CATS, CAV, CC, CFP, CLAPF, CLIMA, CMCIM, CMCS, CMSB, Concur, Coordination, CoOrg, CSL, CTCS, DBPL, ECOOP, Erlang, ESSLLI, ESOP, ETAPS, FLOPS, FMCO, FME, FOAL, FOOL, Forte, FroCos, FSE, FSTTCS, GCSE, Haskell, ICALP, ICCL, ICDCS, ICFEM, ICDCIT, ICDT, ICFP, ICWE, ICWS, ICLP, ICTAC, ISDT, IWFM, LICS, MFCS, MFPS, MPC, MTCOORD, OOPSLA, PADL, PEPM, Plan-X, PLDI, PLI, PODS, POPL, PPDP, PROCOMET, RTA, SAS, Scheme, SPIN, TACAS, TACS, TFP, TGC, TIC, TLCA, TLDI, TPHol, TYPES, VMCAI, VLDB, WebDB, WFLP, WOLLIC, WSFM, WWV, WWW, XML.

References

[1] J. Armstrong, M. Williams, R. Virding. *Concurrent Programming in Erlang*. Prentice-Hall, 1993.

[2] J. Armstrong. Concurrency oriented programming in Erlang. Invited talk, FFG 2003.

[3] K. Beck. *Extreme Programming Explained*. Addison Wesley, 2000.

[4] C. Brabrand, A. Møller, M. Ricky, M. Schwartzbach. PowerForms: Declarative client-side form field validation. *World Wide Web Journal* 3(4), 2000.

[5] C. Brabrand, A. Møller, M. Schwartzbach. The Bigwig project. *TOIT*, 2(2), 2002.

[6] N. Benton, L. Cardelli, C. Fournet. Modern concurrency abstractions for C$^\sharp$. *TOPLAS*, 26(5), 2004.

[7] N. Benton, A. Kennedy, and C. Russo. Adventures in interoperability: the SML.NET experience. *PPDP*, 2004.

[8] V. Benzaken, G. Castagna, and A. Frisch. CDuce: An XML-centric general-purpose language. *ICFP*, 2003.

[9] G. Bierman, E. Meijer, W. Schulte. Programming with rectangles, triangles, and circles. *XML Conference*, 2003.

[10] H. Böttger, A. Møller, M. Schwartzbach. Contracts for cooperation between web service programmers and HTML designers. Tech report, BRICS, 2003.

[11] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of programming with complex objects and collection types. *TCS*, 149(1), 1995.

[12] R. Burstall, D. MacQueen, and D. Sannella. Hope: An experimental applicative language. *Lisp Conference*, 1980.

[13] A. Christensen, A. Møller, M. Schwartzbach. Extending Java for high-level web service construction. *TOPLAS*, 25(6), 2003.

[14] A. Christensen, A. Møller, M. Schwartzbach. Precise analysis of string expressions. *SAS*, 2003.

[15] S. El-Ansary, D. Grolaux, P. Van Roy, M. Rafea. Overcoming the multiplicity of languages and technologies for web-based development. *Mozart/Oz Conference*, LNCS 3389, 2005.

[16] M. Fernández, J. Siméon, P. Wadler. Introduction to the XQuery Formal Semantics. Katz, ed, *XQuery for Experts*, Addison-Wesley, 2004.

[17] C. Fournet, G. Gonthier. The Join Calculus: a language for distributed mobile programming. *Applied Semantics*, LNCS 2395, 2002.

[18] C. Fournet, F. Le Fessant, L. Maranget, A. Schmitt. JoCaml: a language for concurrent distributed and mobile programming. *Advanced Functional Programming*, LNCS 2638, 2003.

[19] J. Garret. Ajax: a new approach to web applications. 2005.

[20] P. Graham. Beating the averages. 2001.

[21] P. Graunke, R. B. Findler, S. Krishnamurthi, M. Felleisen. Automatically restructuring programs for the web. *ASE*, 2001.

[22] P. Graunke, R. B. Findler, S. Krishnamurthi, M. Felleisen. Modeling web interactions. *ESOP*, 2003.

[23] P. Graunke, S. Krishnamurthi, S. van der Hoeven, M. Felleisen. Programming the web with high-level programming languages. *ESOP*, 2001.

[24] V. Gapeyev, M. Levin, B. Pierce, A. Schmitt. The Xtatic experience. *PLAN-X*, 2005.

[25] T. Grust. Monad comprehensions, a versatile representation for queries. In P. Gray, *et al* (editors), *The Functional Approach to Data Management*, Springer Verlag, 2003.

[26] H. Hosoya, B. Pierce. XDuce: A typed XML processing language. *WebDB*, LNCS 1997, 2000.

[27] Haruo Hosoya, Benjamin C. Pierce. XDuce: A typed XML processing language. *TOIT*, 3(2), 2003.

[28] IMRG. Retail's New Order. Press release, 2003.

[29] L. Libkin, L. Wong. Query languages for bags and aggregate functions. *JCSS*, 55(2):241–272, 1997.

[30] M. MacHenry, J. Matthews. Topsl: a Domain-Specific Language for On-Line Surveys. *Scheme Workshop*, 2004.

[31] S. Marlow, P. Wadler. A practical subtyping system for Erlang. *ICFP*, 1997.

[32] A. Møller, M. Schwartzbach. The design space of type checkers for XML transformation languages. *ICDT*, LNCS 3363, 2005.

[33] M. Neubauer and P. Thiemann. From sequential programs to multi-tier applications by program transformation. *POPL*, 2005.

[34] M. Carlsson, J. Nordlander, D. Kieburtz. The semantic layers of Timber. *ASPLAS*, 2003.

[35] M. Odersky, V. Cremet, C. Rockl, M. Zenger. A nominal theory of objects with dependent types. *ECOOP*, 2003.

[36] M. Odersky *et al*. An overview of the Scala programming language. Technical report, EPFL Lausanne, 2004.

[37] F. Pottier, D. Rémy. The essence of ML type inference. In B. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 10, pages 389–489. MIT Press, 2005.

[38] C. Queinnec. Continuations to program web servers. *ICFP*, 2000.

[39] A. Sandholm, M. I. Schwartzbach. A type system for dynamic web documents. *POPL*, 2000.

[40] Science for global ubiquitous computing. A fifteen-year grand challenge for computing research. UKCRC, 2003.

[41] J. Simeon, P. Wadler. The essence of XML. *POPL*, 2003.

[42] D. Syme. F$^\sharp$web page.

[43] P. Thiemann. WASH/CGI: server-side web scripting with sessions and typed, compositional forms. *PADL*, 2002.

[44] P. Van Roy, *et al*. Logic programming in the context of multiparadigm programming: the Oz experience. *TLP* 3(6):717–763, November 2003.

[45] P. Wadler, W. Taha, and D. MacQueen. How to add laziness to a strict language, without even being odd. *Workshop on Standard ML*, 1998.

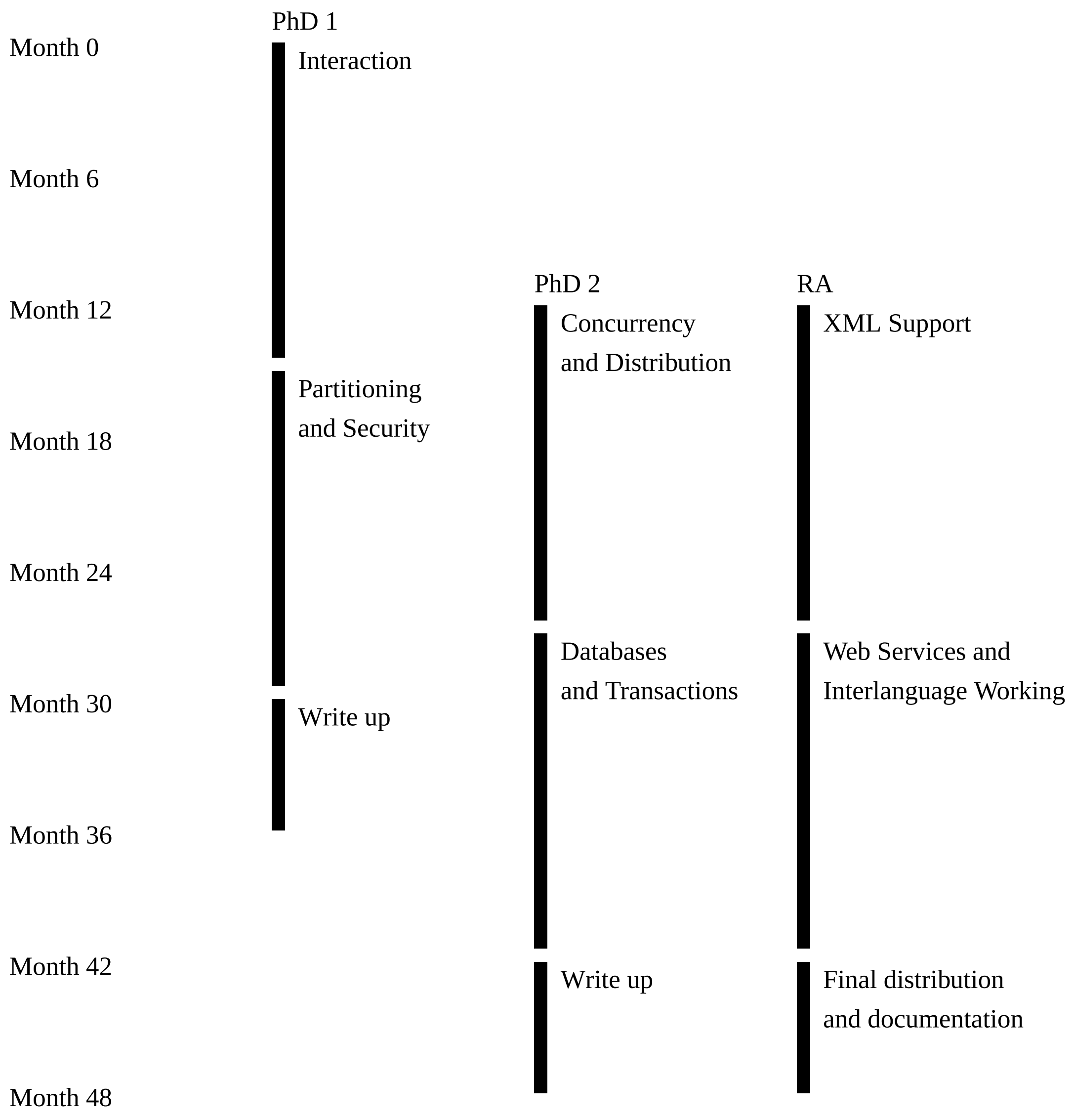
[46] P. Wadler and P. Thiemann. The marriage of effects and monads. *TOCL*, 4(1), 2003.

[47] C. Wikström and H. Nilsson. Mnesia – an industrial DBMS with transactions, distribution, and a logical query language. *CODAS*, 1996.

[48] L. Wong. Kleisli, a functional query system. *JFP*, 10(1), 2000.

[49] XML Query and XSL Working Groups. XQuery 1.0: An XML Query Language, W3C Working Draft, 2005.

Diagrammatic Workplan


PhD 1
PhD 2
RA
Month 0
Month 6
Month 12
Month 18
Month 24
Month 30
Month 36
Month 42
Month 48
Interaction
Partitioning and Security
Write up
Concurrency and Distribution
Databases and Transactions
Write up
XML Support
Web Services and Interlanguage Working
Final distribution and documentation