Links

Philip Wadler
University of Edinburgh

January 2004

Introduction Language shapes thought. This is equally true for prose, poetry, and programming. How shall the next generation think about the web?

Business and government are rapidly evolving to exploit the speed, savings, convenience, flexibility, and opportunities made possible by the World-Wide Web. The firm eMarketeer estimates consumers spent €50 billion over the web in 2003, and will spend €200 billion in 2006.

Web programming is in its infancy. Web sites are expensive to deploy, difficult to navigate, or riddled with errors. E-consumers regularly are unable to complete transactions, faced with an unintelligible error message caused by an event the programmer failed to anticipate. Even popular web sites like Orbitz can suffer from fundamental problems, as detailed below.

[This is taken from a funding proposal to the European Union. I thought it more helpful to circulate it sooner rather than later, even though the tone is not right for this audience.]

Links Two decades ago, researchers at the University of Edinburgh introduced Hope, a widely influential programming language. It was named for Hope Park Square, a park near the University. This research proposes to design a new programming language for the web, Links. It is named for Bruntsfield Links, a park near the University (and site of the world's first public golf course).

The design of Links will exploit successful results already achieved in other languages: distributed and reliable programming from Erlang, database programming from Kleisli, XML programming from Xduce, and support for web interactions from Scheme and Haskell. Adoption will be encouraged by forming a consortium to design and implement the language, building on previous successes of ML and Haskell. The research plan exploits Europe's lead in research on *value-oriented* programming (also known as functional programming).

Distributed programming Web services must be distributed and reliable, available everywhere and everytime. An unavailable server can cost a web-based business millions of euros. Building such systems is known to be difficult.

Erlang is a programming language developed by researchers at Ericsson, used in Ericsson products such as AXD301, DWOS, A910, and ANx. It is designed for building phone switches that are complex and reliable — on the order of a million lines of code, and less than 3 minutes of downtime in one year of operation.

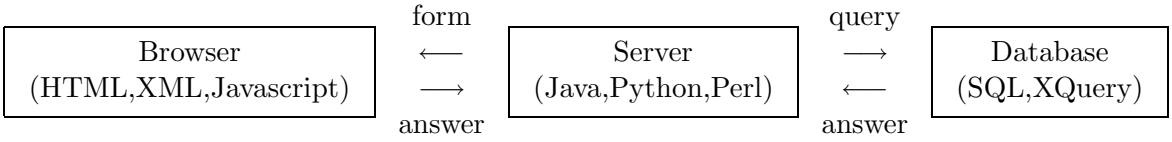
The secret of Erlang's success is a set of design principles that support the creation of distributed and reliable systems. Using this framework, a programmer can code a naive, sequential server, and then instantiate it as a reliable, replicated server. This is far in advance over the support provided by Java/J2EE or C#/.NET.

In *object-oriented* programming, as in Java or C#, each object has a state which may be updated. Care is required to properly synchronize access to objects in a concurrent setting, and to correctly propagate updates to caches in a distributed setting. Errors in such systems are common, and can be difficult to replicate or track down. Such problems are greatly reduced in the *value-oriented* approach used in Erlang, since values are not updated, don't require synchronization, and may be easily cached.

Links will adopt the key feature of the distribution model of Erlang, including the use of value-oriented programming. We will adopt successful Erlang design principles from the published open source implementation and libraries.

Links will differ from Erlang in possessing a static type system. This increases reliability because it rules out large classes of errors and increases efficiency because it supports optimization. To support typing, Links will replace Erlang's "mailbox" model of communication between processes with a more modern "channel" model, probably based on the join calculus developed at INRIA in France.

Database and XML programming A typical web system is organized in three tiers, each running on a separate computer.



Middle-tier logic on the server generates forms to send to a front-end browser and queries to send to a back-end database. The programmer must master many languages: the logic is written in a mixture of Java, Python, and Perl; 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 this problem by incorporating ideas from database programming languages, notably Kleisli, and XML programming languages, notably Xduce.

Over the last decade, Peter Buneman (then at U Penn) and his research group developed *comprehensions* as a basis for database query languages. One outcome of this research is the Kleisli programming language, developed by Limsoon Wong and others. Kleisli has been used to solve database integration problems that otherwise have proved intractable, has been applied to a range of problems in bioinformatics, and is sold as a commercial product. Links will include a comprehension feature and adopt techniques from Kleisli 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 the built-in comprehension feature, from which the Links compiler extracts optimized SQL and XQuery to send to the back-end database.

Over the last five years, Benjamin Pierce (also at U Penn) and his research group developed *regular expression types* as a basis for programming languages for XML. One outcome of this research is the Xduce programming language, developed by Haruo Hasoya, and others. Links will include regular expression types and support for XML processing. Programmers writing middle-tier logic in Links will easily manipulate XML, using generalizations of the techniques that programmers in Perl and Python use to manipulate strings.

Web interaction Matthias Felleisen encountered a problem while comparison shopping on Orbitz. He clicks on a flight, sees its price and details appear in a second window, clicks on another flight, sees its price and details appear in a third window, returns to the second window, clicks submit — and ends up buying the flight displayed in the *third* window, not the second.

Links will solve this problem by incorporating continuation-based support for web interactions, based on ideas developed in Scheme and Haskell.

Solutions to this problem were developed, independently, by Christian Quenniec (Paris VI) in Scheme, and John Hughes (Chalmers) in Haskell. The solutions are based on capturing a *continuation* — a representation of the remainder of a computation past the current program point. The Scheme work has been further developed by Felleisen (Northeastern), and Peter Thieman (Freiburg).

Value-oriented programming All of these languages exhibit *value-oriented programming*, where data structures are built by composition, and the role of update is severely restricted. Contrast this with object-oriented programming, where classes are created by updating fields. For instance, here is a method to return a list of three successive integer values in Java:

```
List count(int n) {
  List l = new LinkedList();
  l.add(new Integer(n));
  l.add(new Integer(n+1));
  l.add(new Integer(n+2)));
  return l;
}
```

And here is a function to do the same thing in Haskell.

```
count(n) = [n,n+1,n+2]
```

Both the Java and Haskell programs are statically typed; in Haskell, the types are inferred and declarations are optional. Building data structures by composition is more flexible and more compact. Restricting the use of update is essential: for distribution (as in Erlang) this means caching can be used extensively; for databases (as in Kleisli) it supports the optimization of queries.

Consortium Dozens of new programming languages are introduced each year. Few of these see widespread uptake. In order to ensure that Links develops a significant user community, we will adopt the strategy of forming a consortium of research groups to develop and promote Links. This strategy has proved effective for programming languages since Algol 60.

In particular, the value-oriented languages ML and Haskell both benefited from the consortium strategy. Both languages have spawned variants, Haskell having influenced Clean and Mercury (among others), and ML having spawned Standard ML, Moscow ML, and O'Caml. The latter integrates object-oriented and value-oriented styles, and has become quite popular in its own right.

We plan to build on these successes by founding a new consortium, drawing on members of both the ML and Haskell communities. We may explore forming a Network of Excellence to support the consortium activities. Members of the consortium will cooperate on the design and implementation of the language. There will also be an emphasis on building integrated development environments, and developing applications to test and further refine the language design and implementation.

Much legacy code is written in object-oriented languages, and Links will adopt ideas from Haskell and ML for smooth interworking with languages such as C, C++, Java, and C#.

Possible contributors to such an effort include: Philip Wadler, Peter Buneman, Don Sannella, and Ian Stark, Edinburgh (Haskell, ML, Erlang, Kleisli, databases, XML, distribution); Simon Peyton Jones and Andrew Gordon, Cambridge (Haskell, compilers, distribution, XML, security); Philippa Gardner, London (distribution, XML); Joe Armstrong, Stockholm (Erlang, distribution); John Hughes, Göteborg (Haskell, Erlang, web interaction); Giuseppe Castagna and Veronique Benzaken, Paris (ML, XML, databases); Xavier Leroy, Paris (ML, O'Caml, compilers); Giorgio Ghelli, Pisa (distribution, databases); Peter Thiemannn, Freiburg (Haskell, web interaction); Martin Odersky, Lausanne (object-oriented languages, compilers); Fritz Henglein, Copenhagen (ML, distribution, XML). We will also collaborate with relevant researchers in the US, including Greg Morrisett, Boston (ML, compilers, proof-carrying code) and Benjamin Pierce, Philadelphia (ML, distribution, XML).

Related work Other languages aimed at middle-tier programming include Xtatic from Pierce at U Penn, Xen from Eric Meijer at Microsoft, and Scala from Odersky at Lausanne. Links overlaps with these, and we expect fruitful interactions with those teams. One difference of Links is that it places more emphasis on applications, focussing on adopting ideas for distributed systems from Erlang, and ideas for databases integration from Kleisli.