Resource-Based Web Applications
Extended Abstract

Sebastian Fischer
sebf@informatik.uni-kiel.de
Christian-Albrechts-University of Kiel, Germany

Abstract

We present an approach to write web applications in the programming languages Haskell [6] and Curry [4]. The web applications we propose are directly based on the Hypertext Transfer Protocol (HTTP) – no additional protocol on top of HTTP is necessary. Although the main ideas are not new and already applied using other programming languages, we believe that features present in modern declarative languages – especially algebraic datatypes – serve well to represent resources on the web.

1 Resource Framework

A *resource* is any piece of information named by a Uniform Resource Identifier (URI). A web application provides access to internally stored information over the Internet or accesses such information provided by other web applications. Usually, (a representation of) this information is transferred using the Hypertext Transfer Protocol (HTTP) [1] – possibly employing some protocol on top of it. Note that we do not primarily consider applications running inside a web browser like [2, 5, 7].

HTTP provides different methods to access resources from which the most important are the GET and the POST method. GET is used to *retrieve* some information, e.g., web browsers use GET to ask for the content of web pages. POST is used to *send* information to be processed by a server, e.g., entries in a web form can be sent to the server using POST. While processing the request, the server can update its internally stored information and usually sends back a result of this process.

We provide a datatype `Resource a b c` and library functions `get` and `update` to create a uniform interface to local and remote resources. Remote resources are directly accessed via HTTP and we provide operations to create a CGI program that makes local resources available over the Internet. Using our library both the client and server side of web applications can be implemented without knowledge of the details of HTTP communication. Hence, the programmer can concentrate on the application logic which is separated from network communication.

The user of our library can control the representation of resources during transfer and it is possible to support different formats in a single web application with content negotiation. We provide type-based combinators to concisely construct XML and JSON converters and integrate them seamlessly into our framework. Moreover, we implement authenticated communication over HTTP and integrate it transparently into our framework, i.e., hiding the details from the programmer of web applications.

We see two main advantages in using a declarative programming language like Haskell or Curry for writing web applications: Algebraic datatypes are tree-like structures and therefore well suited to represent hierarchically structured data, e.g., in XML format. Furthermore, the type systems of Haskell and Curry help to find errors in advance, which is very important in applications publicly available over the Internet. Due to the type-based conversion combinators and the built-in error handling mechanism, the programmer of a web application always gets type-correct inputs from clients. She is therefore released from the burden to write complex and error-prone code, that is not primarily considered with the application itself.

2 Related and Future Work

Hanus [2], Meijer [5] and Thiemann [7] provide frameworks to build server side web applications based on HTML forms that run inside a web browser. Although [5] abstracts from network communication, it uses strings as references to input fields, which is a source of possible errors. [2, 7] abstract also from such references, which not only prevents typing-errors but also enables the programmer to compose different web forms without the risk of name clashes. Recently, Hanus [3] presented an approach to concisely describe type-based web user interfaces (WUIs), based on [2].

Our approach differs from the mentioned approaches because we do not aim at web based *user* interfaces through HTML forms but provide a framework for web based interfaces that transfer *raw data*. Such interfaces can be used for machine-to-machine communication or in the upcoming Web 2.0 applications based on asynchronous HTTP transfer using Javascript. The transferred data can be in a variety of formats (e.g., XML, JSON, or almost anything else) and we provide a mechanism to support multiple formats at once in a single web application. A client could even send a request in XML and receive the response in JSON format. The type-based combinators we presented to specify how data has to be converted are inspired by the WUI combinators presented by Hanus [3]. In fact they are so similar, that we think an integration of both approaches should be possible.

So, for future work we plan to integrate the machine-interface framework with the user-interface framework presented by Hanus [3]. We think, it should be possible to support the manipulation of local data through a web based interface by both humans and machines with a single web application.

Furthermore, the type-based combinators to construct XML converters serve well to model existing datatypes as XML data. However, modeling existing XML data as datatypes in Haskell or Curry is not always possible using our combinators. For example, attributes are not supported. We are working on a more flexible combinator library for XML converters and plan to integrate it into our framework for resource-based web applications.

References

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. June 1999.

[2] M. Hanus. High-Level Server Side Web Scripting in Curry. In *Proc. of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, pages 76–92. Springer LNCS 1990, 2001.

[3] M. Hanus. Type-Oriented Construction of Web User Interfaces. Technical report, Technical Report, CAU Kiel, 2006.

[4] M. Hanus (ed.). Curry: An Integrated Functional Logic Language (Vers. 0.8). Available at `http://www.informatik.uni-kiel.de/~curry`, 2003.

[5] E. Meijer. Server Side Web Scripting in Haskell. *Journal of Functional Programming*, 10(1):1–18, 2000.

[6] S. Peyton Jones, editor. *Haskell 98 Language and Libraries—The Revised Report*. Cambridge University Press, 2003.

[7] P. Thiemann. WASH/CGI: Server-side Web Scripting with Sessions and Typed, Compositional Forms. In *4th International Symposium on Practical Aspects of Declarative Languages (PADL 2002)*, pages 192–208. Springer LNCS 2257, 2002.