

Poster: Tierless Programming in JavaScript

Laure Philips*, Wolfgang De Meuter*, Coen De Roover*

*Software Languages Lab,

Vrije Universiteit Brussel, Belgium

Email: lphilips, wdmeuter, cderoove @vub.ac.be

Abstract—Whereas “responsive” web applications already offered a more desktop-like experience, there is an increasing user demand for “rich” web applications (RIAs) that offer collaborative and even off-line functionality. Realizing these qualities requires distributing previously centralized application logic and state vertically from the server to a client tier (e.g., for desktop-like and off-line client functionality), and horizontally between instances of the same tier (e.g., for collaborative client functionality and for scaling of resource-starved services). Both bring about the essential complexity of distributing application assets and maintaining their consistency, along with the accidental complexity of reconciling a myriad of heterogenous tier-specific technology. Tierless programming languages enable developing web applications as a single artefact that is automatically split in tier-specific code —resulting in a development process akin to that of a desktop application. This relieves developers of distribution and consistency concerns, as well as the need to align different tier-specific technologies. However, programmers still have to adopt a new and perhaps esoteric language. We therefore advocate developing tierless programs in a general-purpose language instead. In this poster, we introduce our approach to tierless programming in JavaScript. We expand upon our previous work by identifying development challenges arising from this approach that could be resolved through tool support.

I. INTRODUCTION

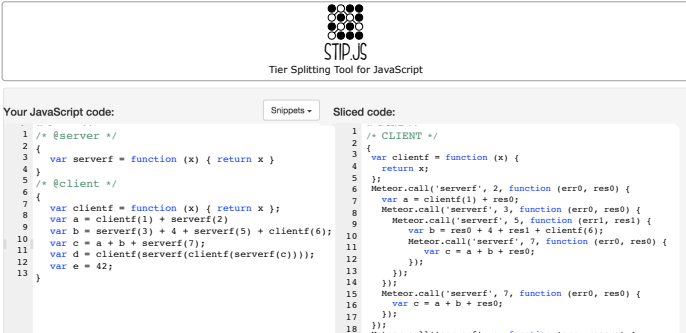
The vision of tierless programming is to develop, test and maintain multi-tier web applications as a single artefact that spans the traditional database, server and client tiers. This eliminates the impedance mismatch between different tier-specific technologies. Different approaches to tierless programming exist: tierless languages (e.g., Links [1], Hop [2], Ur/Web [3], WebDSL [4]) introduce a new language, while tierless frameworks (e.g., Google Web Toolkit (GWT)¹ and Meteor²), enable developing such applications in general-purpose languages. Most of these approaches require the developer to specify *tier demarcation information*: tier annotations are used by a compiler or transpiler to split the program into client, server and database tiers.

While tierless languages reduce essential and accidental complexities of web development, the developer still needs to learn a new language. Moreover, tool support for tierless languages is often lacking. We therefore advocate developing tierless applications in existing general-purpose languages, from which existing tool support can be leveraged. In this poster, we summarize our approach to tierless web development in

JavaScript [5]. We focus on how the initial prototype presented there can be extended to support full-fledged web applications, while identifying challenges arising from this approach that could be resolved through dedicated tool support.

II. TIERLESS PROGRAMMING USING STIP.JS

We introduce our tier splitting process [5] that takes a tierless JavaScript program as input and produces tier-specific code for the server and client tier. A prototype of this process is implemented by the tool STIP.JS³, of which a screenshot in figure 1 is given. This prototype supports tierless web development for the JavaScript language. Many web developers are familiar with the language, ubiquitous on server and client tiers alike, and it has an extensive set of tools and libraries for developing RIAs. Programmers implement the tierless web application as a normal JavaScript program, where all data and functions are defined locally. After testing and debugging the application locally, a minimum of code needs to be annotated with `@client` or `@server` comments. The prototype tool then performs a *tier splitting* process on this code to separate client code from server code. The tier-splitting process relies on a variant of program slicing [6], a technique that can part a valid subset of a program. Our algorithm constructs a distributed dependence graph using the tier annotations in the program, and slices the graph into a subset destined for the client and server tier respectively. These slices are subsequently *transpiled* to add code that implements appropriate distribution (e.g., through remote procedure calls) and data sharing and replication concerns (e.g., through web frameworks such as Meteor).



The screenshot shows the Stip.js tool interface. At the top, it says "STIP.JS Tier Splitting Tool for JavaScript". Below that, there are two panels: "Your JavaScript code:" and "Sliced code:". The "Your JavaScript code:" panel shows a JavaScript function `serverf` with a `@server` annotation, and a function `clientf` with a `@client` annotation. The "Sliced code:" panel shows the resulting code, where the `serverf` function is now a Meteor call to a server-side function, and the `clientf` function is now a Meteor call to a client-side function. The code is as follows:

```
1 /* @server */
2 {
3   var serverf = function (x) { return x }
4 }
5
6 /* @client */
7 {
8   var clientf = function (x) { return x };
9   var a = clientf(1) + serverf(2);
10  var b = serverf(3) + 4 + serverf(5) + clientf(6);
11  var c = a + b + serverf(7);
12  var d = clientf(serverf(clientf(serverf(c))));
13  var e = 42;
14 }

1 /* CLIENT */
2 {
3   var clientf = function (x) {
4     return x;
5   };
6   Meteor.call('serverf', 2, function (err0, res0) {
7     var a = clientf(1) + res0;
8     Meteor.call('serverf', 3, function (err1, res1) {
9       Meteor.call('serverf', 5, function (err2, res2) {
10        var b = res0 + 4 + res1 + clientf(6);
11        Meteor.call('serverf', 7, function (err3, res3) {
12          var c = a + b + res0;
13        });
14      });
15    });
16    var c = a + b + res0;
17  });
18  Meteor.call('serverf', c, function (err, resarg) {
```

Fig. 1. Screenshot of the Stip.js tool

¹<http://www.gwtproject.org/>

²<http://www.meteor.com>

³<http://bit.ly/stipjs>

III. DISCUSSION

Our previous work demonstrated that this approach to tierless programming is feasible for small, but representative web applications. To fully realize our vision, however, several challenges remain —some of which could be overcome through dedicated tooling.

a) Exposing Control over Distributed Failure Handling: The tierless program seemingly executes in a synchronous, non-distributed manner. As such, the developer is not aware of the failures that can occur once the tier split program executes in a distributed setting. For example, a local call to a function `foo(42)` gets transpiled to a remote procedure call with a callback `rpc("foo", 42, function (error, result) {})`, where `rpc` is a construct from a distributed programming library. This particular library enables reacting to distributed failures through the `error` callback function. In the tierless call, however, this parameter does not exist. How network errors are handled is different for each application. A solution could be to introduce a new annotation that allows custom failure handling at e.g., the tier level. However, this control should not come at the cost of the tierless illusion.

b) Maintaining the Consistency of Replicated Data: In our previous work, we focused on the design space for distributing and replicating individual variables. Data replication enables collaborative and offline functionality across and between tiers. Our prototype relied on the Meteor framework, which offers an elegant way of replicating data between clients. However, it offers no consistency guarantees when e.g., two clients concurrently change the same variable. Different strategies have been proposed to keep distributed and replicated data consistent. We are looking into the use of annotations to specify such strategies, while the implementation is provided by our runtime. Eventual consistency provides a good consensus between availability and strong consistency of replicated data. We are therefore extending our prototype with an open-source JavaScript implementation [7] of Cloud Types [8]. In addition to consistency mechanisms, database-aware slicing and replication need to be investigated, beyond our current support at the level of individual variables.

c) Supporting Existing Tooling in the Presence of Tier-specific Code: We demonstrated in [5] that our approach allows reusing existing development and validation tooling for JavaScript. However, certain tier-specific tasks like DOM manipulation need to be reflected in the tierless variant of the program. To this end, we currently provide tierless primitives for common concerns in web applications. For instance, there are communication primitives such as `broadcast` and `publish`. The actual implementation of these primitives is provided by the transpiler and depends on the chosen target platform. Research is needed to see how existing tools can cope with tier-specific code in a tierless program. Emulating the browser through libraries such as `JsDom` might still enable testing the tierless application as a whole on a Node.js server.

d) Dedicated Tooling to Maintain the Tierless Illusion: A downside of transpiling a tierless program into several tiers, is that developers may witness code at run-time that differs

from the code that they had implemented. The transpiled code is at odds with their tierless model of the program. This problem becomes apparent when developers need to debug their program, as existing debuggers will be oblivious to the transpilation process. Research is therefore needed on a specialized debugger capable of maintaining the tierless illusion across tier-specific runtime artefacts. Along the same lines, research is needed on other tier-specific tooling. A feedback tool could warn developers about inconsistent tier-demarcating annotations. For instance, when a variable is declared both on the server tier and the client tier. We also believe that the programmer can benefit from a graphical representation of our distributed program dependence graph in which the different tier-specific nodes and edges are marked appropriately.

e) Exploring Automatic Scaling of Server Tiers: Finally, different application domains deserve exploring for our solution. The current tier splitting process focuses on the vertical distribution of code and data across the server and client tiers. Horizontal distribution, between instances of the same tier, is required for collaborative client functionality and for scaling of resource-starved server tiers. We believe that horizontal distribution brings about problems similar to those that arise from vertical distribution. Research is needed to transpose our solution to this related problem.

IV. CONCLUSION

Our approach to tierless programming enables developing web applications as JavaScript programs that span multiple tiers. This while allowing existing JavaScript tooling to be reused. While a prototype implementation has demonstrated its feasibility, several challenges preclude us from fully realizing this vision. This poster abstract explicates these challenges and proposes potential strategies to overcome them, some of which amount to dedicated tooling.

ACKNOWLEDGMENTS

Laure Philips is supported by a doctoral scholarship granted by the Agency for Innovation by Science and Technology in Flanders, Belgium (IWT).

REFERENCES

- [1] E. Cooper, S. Lindley, P. Wadler, and J. Yallop, "Links: Web programming Without Tiers," in *FMCO*. Springer-Verlag, 2006.
- [2] M. Serrano, E. Gallesio, and F. Loitsch, "Hop: a Language for Programming the Web 2.0," in *OOPSLA Companion*, 2006, pp. 975–985.
- [3] A. Chlipala, "Ur: Statically-typed Metaprogramming with Type-level Record Computation," in *PLDI '10*. New York, NY, USA: ACM, 2010, pp. 122–133.
- [4] Z. Hemel, D. M. Groenewegen, L. C. L. Kats, and E. Visser, "Static Consistency Checking of Web Applications with WebDSL," *J. Symb. Comput.*, vol. 46, no. 2, pp. 150–182, Feb. 2011.
- [5] L. Philips, C. De Roover, T. Van Cutsem, and W. De Meuter, "Towards Tierless Web Development Without Tierless Languages," ser. Onward! 2014. New York, NY, USA: ACM, 2014, pp. 69–81.
- [6] M. Weiser, "Program slicing," *IEEE Trans. Software Eng.*, vol. 10, no. 4, pp. 352–357, 1984.
- [7] T. Coppieters, L. Philips, W. De Meuter, and T. Van Cutsem, "An Open Implementation of Cloud Types for the Web," ser. PaPEC '14. New York, NY, USA: ACM, 2014, pp. 2:1–2:2.
- [8] S. Burckhardt, M. Fähndrich, D. Leijen, and B. P. Wood, "Cloud Types for Eventual Consistency," in *ECOOP'12*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 283–307.