Coupling Server-Side Templates and Client-Side Models

Anders Ingemann, 20052979

Master's Thesis, Computer Science

April 2012

Advisor: Michael Schwartzbach

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

▶in English. . . ◀

Resumé

▶in Danish. . .◀

Acknowledgements

►. . . ◄

Anders Ingemann,
Aarhus, April 26, 2012.

Contents

Chapter 1

Introduction

►. . . ◄

►example of a citation to primary literature: [?], and one to secondary literature: [?]◄

Chapter 2

Developing webapplications

Webapplications are on the rise. Not a day goes by where a new webapp is popping up for uses that were previously reserved for a program locally installed on a computer. Even more so: Previously unimagined uses for any internet enabled device seem to be developed at a rate that surpasses the former.

2.1 What are webapplications?

There are many definitions for a webapplication. What they all have in common is the fact that they communicate with the outside world. Their implementations differ from device to device, desktop PCs included.

In this thesis however, we will focus on webapplications which use a modern web browser and with it HTML as their basis (HTML5 in particular). Interactivity is supplied by JavaScript in this case. There do exist other languages like Dart, CoffeeScript and Google Web Toolkit. However, they are all translated into JavaScript if cross-browser compatibility is a requirement (which it almost always is).

Another way to render a webapplication interactive, is by rendering customized HTML pages on the server. This has both advantages and disadvantages to the client-side scripting method. data-processing speed security concerns client initialization control over application (cross browser stuff)

2.2 The development process

The development process of a webapplication is similar to any other software development process. One starts with the data to be modeled. This might happen on the server and the client simultaneously. A protocol for communication between the two is then established. Even though the design layout for an application may have existed in the beginning of the process it is usually only fully implemented when most other critical components are in place.

2.2.1 Usual design patterns

The model-view-controller design pattern has proven itself to be a sane choice for developing webapplications. Most frameworks today use this pattern or variations thereof. model-view-controller-presenter (/ModelView) Designs of webapplications change with time, features are added or removed and common processes simplified. In light of this, it is desireable to ensure that the "View" part of the model-view-controller pattern is easily modifieable.

Chapter 3

Templating languages

3.1 A survey

3.1.1 Mustache

3.1.2 ►. . .◄

3.2 Limitations

These templating languages are very different in their design. All aim to improve one or more aspects of the templating task. Of those Mustache seems to be specifically tailored for webapplications with interactive JavaScript parts. They do all have a common trait which in some cases can be an advantage but given any specific implementation of a webapplication is a drawback: They are completely oblivious of their surroundings. They draw the line at the "View" part in order to encourage a seperation from the other parts. This comes at the cost of lost information when sending a rendered view to the client.

Chapter 4

Requirements

The information lost after a view is rendered might not be useful and the behavior of existing templating languages therefore inconsequential. This is not the case. Let us consider minimal template used for displaying profile information: ▶... ◀ As you can see, the fields of the user object are printed into the HTML at the appropriate places, leaving us with a normal page which can be displayed in the browser. Add the requirement that this form is not to be submitted via a synchronous browser request but via AJAX. Now the developer has to reverse engineer the generated HTML with JavaScript to obtain the user information that is to be submitted. Any changes to the template will now also require a change in client side code, particularly the code, which finds the values in the form. This example contains a rather obvious loss of information. The position of the field attributes of the user object. At the time of the rendering these positions are known, but as soon as the result is converted into a string that is sent to the client, this information is lost.

4.1 Development workflow

4.2 Initial prototype

4.2.1 Libraries

4.2.2 The application

4.2.3 Results

4.3 Plan for next iteration ▶... ◀

Chapter 5

Implementation

5.1 Templating language syntax

5.1.1 Integrating the parser

5.2 Template-aware clients

5.2.1 Client side architecture

5.2.2 Transmitting meta-data

Chapter 6

Usage

6.1 Application example

Chapter 7

Evaluation

7.1 Performance

7.2 Limitations

7.3 Advantages

7.3.1 Comparison

Chapter 8

Future Work

Chapter 9

Related Work

Chapter 10

Conclusion

►. . . ◄

Primary Bibliography

[A1] Simon Holm Jensen, Anders Møller, and Peter Thiemann. Interprocedural analysis with lazy propagation. In *Proc. 17th International Static Analysis Symposium, SAS '10*, volume 6337 of *LNCS*. Springer-Verlag, September 2010.

Secondary Bibliography

[B2] Claus Brabrand, Robert Giegerich, and Anders Møller. Analyzing ambiguity of context-free grammars. *Science of Computer Programming*, 75(3):176–191, March 2010.