Mockup Driven Web Development

Edward Benson
Supervised by: David Karger
MIT CSAIL
77 Massachusetts Avenue
Cambridge, USA
eob@csail.mit.edu

ABSTRACT

Dynamic web development still borrows heavily from its origins in CGI scripts: modern web applications are largely designed and developed as programs that happen to output HTML. This thesis proposes to investigate the idea taking a mockup-centric approach instead, in which self-contained, full page web mockups are the central artifact driving the application development process. In some cases, these mockups are sufficient to infer the dynamic application structure completely.

This approach to mockup driven development is made possible by the development of a language the thesis develops, called Cascading Tree Sheets (CTS), that enables a mockup to be annotated with enough information so that many common web development tasks and workflows can be eliminated or vastly simplified. CTS describes and encapsulates a web page's design structure the same way CSS describes its styles. This enables mockups to serve as the input of a web application rather than simply a design artifact. Using this capability, I will study the feasibility and usability of mockup driven development for a range of novice and expert authorship tasks. The thesis aims to finish by demonstrating that the functionality of a domain-specific content management system can be inferred automatically from site mockups.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software—*reuse libraries, reuse models*

General Terms

Design, Human Factors, Standarization

1. INTRODUCTION

The early days of the web combined a simple hypertext format with a copy-and-tweak culture. People didn't need to become programmers to create a web site. They could simply copy a file they liked and tweak it to reflect their own content. The HTML markup involved was nearly self evident given an example to modify, akin to today's Markdown syntax. The amazing growth and subsequent success of the web was arguably due, in large part, to the ease of self expression and publication that simplicity afforded.

Programmers quickly developed the trick of using dynamically generated web pages as the interface layer to server-side applications, and the structural complexity of web pages began to rise. But these new sites were harder to copy and modify; their HTML structure was increasingly created by full-time HTML authors, not hobbyists, and often with the help of a server-side program. In response to this growing complexity, CSS was created to separate structure from style. For years, this new addition to family of web languages quelled the complexity crisis, preserving HTML as a format suitable for casual copy-and-tweak publication.

Our growing expectations of form and function have since continued to inflate the complexity of our web documents, and today the hand editability of the average HTML page is questionable once again. We mitigate this complexity with widely deployed applications and libraries such as WordPress and Twitter Bootstrap, but these are patches atop the problem rather than solutions. This thesis is a reaction to the feeling that it is time once again to consider an addition to the family of web languages to extract and simply certain aspects of web authoring to tame complexity and drive HTML back toward an easy copy-and-tweak format.

This thesis focuses specifically on this problem of the relationship between the raw, communicative *content* of a page and the *structure* which scaffolds that content for presentation and storage. In modern web pages, content is commingled with considerable design scaffolding that, from the standpoint of raw source editing, obfuscates and complicates both content authoring and design tasks. Authors must wade through this complex structure to locate the content they wish to edit, and designers have to replicate their design modifications once for each invocation (even, occasionally, when using template systems, such as the HTML conventions of Twitter Bootstrap).

The opportunities for improvement with regard to simplifying the specification of content and structure do not end with static HTML, however: much of the web, and modern authoring workflow, is based on the idea of the Content Management Systems (CMS) as a provider of WYSIWYG web authoring. But modern CMSs are sizeable programming accomplishments with their origins in procedurally defined applications that use HTML as merely the interface definition language. This results in considerable complexity to provide relatively simple content management operations. In the case of WordPress, for example, data state is procedurally replicated separately in HTML, Javascript, PHP, and a database. The second phase of this thesis aims to show that, by providing a language which can describe the relationship between content and structure, we can use HTML mockups as the definition of a simple content management system, rather than simply the output format. Such a capability would enable a new paradigm of web application construction, **mockup driven development**, in which the declarative interface mockup, rather than procedural code, serves as the backbone of the application definition.

The final piece of this work will address the challenge of transitioning the population of CMS users to a mockup-driven ecosys-

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

tem. Any technology that provides a replacement for existing methods faces the reality that switching cost is often high enough to deter users, even if the benefits are clear.

2. STATE OF THE ART

This thesis draws upon a rich body of work seeking to improve the web as a platform for content description and application authoring.

Languages and Ontologies. Many language-driven approaches seek to formalize standard design needs into concise, central vocabularies for reuse. HTML5 and CSS3 for example, are vocabulary expansions (with implementation to back them up) over their previous versions. While HTML and CSS focus on page design, other efforts have provided way to demarcate data within the page. Microformats [12], as well as several prominent Semantic Web offerings such as FOAF [5] and Good Relations [10] embed data conforming to domain-specific vocabularies within HTML. Other efforts, such as RDFa [1] and HTML5 Microdata [11] provide a general-purpose syntax and abstract data model for embedding extensible data within HTML. HTML5 Web Components provide a way to define, resuse, and parameterize custom HTML elements with a strong layer of encapsulation around them [**?**]. Our CTS language provides similar functionality while providing an additional layer of indirection that permits more flexibile content reuse.

Frameworks. The Hilda project [19, 18] proposed a declarative web development model in which web assets and operations on those assets were described by a graphical data structure that could be reasoned about, and appropriate portions offloaded onto the client for processing. This provided one model for achieving an automated solution to the problem of balancing load between clients and servers. The FORWARD (App2You) project[13, 8] propose a model that removes the controller from the traditional MVC design. It adds declarative extensions to XHTML that, along with embedded SQL, enable users to query a remote database and bind the results to pre-packaged widget invocations, like a Google Map, with no application code necessary.

Design Tools. Several projects from the HCI community have begun to explore ways to mitigate the complexity of modern web programming. Sensemaking approaches like WebCrystal [6] and FireCrystal [17] help authors understand why a fragment of a web page appears or behaves as it does so that they can repurpose it. And recently even static analysis methods for HTML and CSS have emerged [9]. We take these as evidence that the current web programming model results in applications whose complexity exceeds our ability to understand them without software assistance.

Other tools take novel approaches to web design tasks. WYSIWYG editors and mockup tools are the canonical example, but more recent work has focused on retargeting, which is a common practice that until recently was only done by hand. CopyStyler [7] interactively helps users retarget entire pages with an interface that places them side-by-side. Bricolage [15] uses machine learning to perform page-level retargeting automatically. Its model is trained to find equivalences between blocks of content based on features about their positioning, style, and text contents.

XML to RDBMS. User interfaces are inherently hierarchical, and thus so are the languages (like HTML) that describe them. Regardless of a data source's source model, by the time it interacts with HTML it must be projected into hierarchical space. Nevertheless, back-end data models are often relational in nature. The literature suggests that XML documents can be used to derive a compatible relational structure to describe them [14, 2]. This suggests that it might be possible to completely automate the translation layer between databases and web pages for web applications with little domain-specific business logic in between.

3. PROPOSED APPROACH

This thesis develops the idea of **mockup-driven web development** (MDD) as an approach through which to improve the authoring and reuse of web content. Mockup driven web development turns the CGI model of web applications on its head: declarative user interface mockups, rather than the procedural application code, are the central conveyor of application structure. Given an HTML mockup with light-weight declarative annotations, an MDD system should be able to provide both static and dynamic web authors with benefits. It should allow static authors to cleanly separate their written content from the mockup scaffold which describes the structure of its presentation. For dynamic offers, the mockup should contain enough information to automatically provide a wide range of functionality that today is implemented by hand, such as basic content management operations, importing and caching dynamic structured data, and theming and data sharing capabilities.

I propose to develop this approach by beginning with basic questions of system and language design to enable this style of development. I will then study how this architecture affects static web authoring usability, develop performance and automation algorithms capable of using mockups to replace CMS functionality, and finally develop a means to transition current CMS users to a mockup-driven environment. Together, these four pieces of work will provide a clear picture of the feasibility of mockup-driven development as a model for web development, what its practical limitations are, and where high-value opportunities lie for future work.

One might ask, "why focus so much effort on low level HTML editability"? First, despite the popularity of content management systems, authoring and maintaining static HTML content is still common: personal home pages, research group and project pages, university course pages, academic resumes, self-hosted content, and so on. Second, I aim to show that many dynamic web applications can express much of their design and functionality in terms carefully constructed, low-level HTML, making it a relevant format even for professional programmers. Finally accessibility of HTML remains important even in a world where WYSIWYG editing systems dominate, because HTML is the persistence and output format of these systems: the "debugging view," so to speak.

This section describes the approach and high-level intuition that I will take for each step, and Section 4 details specific methodologies I will apply.

3.1 Language and Runtime Development

If we added a langauge to the web stack to enable mockup driven development, what would it look like and what would its features be?

I propose that the key capability necessary for mockup-driven development—but missing from the web family of languages today—is the ability to describe the relationship between content and structure on the web. CSS separates many aspects of HTML content from its presentation, but the HTML page that remains is still a complex combination of raw content and "presentational" HTML scaffolding to display it. This presentational HTML obfuscates the division between content and display. By making this division explicit, we can build tools that exploit knowledge of the true underlying content schema of the web page while integrating with the custom presentation layer.

Rather than take an ontologist's approach, coining a microdata vocabulary as many have done in the past, I propose a relational

language which enables developers to describe how hierarchical structures on the web (such as HTML or JSON) relate to each other. Such a language provides the ability to talk about the relationship between content and structure on the web without constraining the use of the language to a particular domain. Users of such a language could converge around common vocabularies by relating an artifacts to a common "hub" structure, but they also retain the flexibility to construct diverse custom mappings and grass-roots usage.

I propose that a small set of such declarative relational operators can be used to provide sufficient information about a web page and its content to enable mockup-driven development as defined above. I have developed these operators into a declarative language called Cascading Tree Sheets (CTS) and constructed a Javascript runtime around it to test its capability.

Using CTS, for example, an author could remap the content of their page into the style of another's, simply by authoring a tree sheet which relates equivalences and conditionalities between the two pages. In a dynamic web setting, a tree sheet can be used both to combine structured data with a template (in the form of a mockup) and also to scrape data back out of that template. And relations between trees can be joined, like database relations, to hop from one web design to another, while preserving the content.

3.2 Usability Studies

Does mockup driven development offer improved authoring experience to novices, skilled developers, and professional web library maintainers?

The second step of the proposed approach seeks to understand the authoring usability of mockup-driven web development—and CTS as an implementation of it–from the standpoint of writing HTML. This the prior stage can structurally show that CTS and MDD provide simplifications to HTML content documents, this stage will test whether that simplification also results in a net usability gain for authors.

The CTS approach to mockup-driven development will present a tradeoff between learnability and efficiency. On the one hand, CTS should simplify many authoring tasks because it enables a clean separation of concerns within HTML documents: this isolates the task of authoring design from that of authoring content, and it enables even static web authors to take advantage of "themes" that are write-once, apply-everywhere. On the other hand, this approach not only requires that users incur the cost of learning a new language, it also requires them to think abstractly about the structural layout of their web pages. While experts are likely used to thinking on this level, casual HTML authors may be unfamiliar with this way of thinking about web authoring.

To test learnability for a web technology, it is important to consider how web authors learn. Web authoring involves heavy reuse. People find CSS that they like, copy it, and then learn how to make small modifications to customize it. Thus, one can expect that the learning curve for a new language like CTS would follow a similar path: a user's first encounter with CTS would be to link to and reuse some CTS sheet, an intermediate author would further modify it, and a skilled author would write CTS sheets from scratch.

I propose to take a usability testing approach that seeks to understand these three tiers of usage: *resuse by copying*, *reuse with modification*, and *original creation*. Each of these will be examined via a user study described in Section 4. I will supplement these user studies with qualitative interviews of web authors who maintain and write their own HTML.

3.3 Automation and Performance

Can an annotated mockup be used to synthesize the application functionality typically found in content management systems?

Web application authors today spend great effort implementing relatively standard content management features and performance optimizations to their applications. Content management features include the ability add and edit lightly structured "items" of the type managed by the web application: blog posts, pages, recipes, etc. Because these systems are web *applications*, as opposed to static content, CMSs also provide a range of performance optimizations mitigate the costs of dynamic rendering. This stage of the project will explore the range of CMS functionality and optimization that can be derived from the mockup annotations that CTS provides.

Synthesizing a content management application from a declarative mockup can be an arbitrarily complex topic. At heart of the problem is a tradeoff between offering a broad set of capability and requiring only a small amount of annotation. I propose to approach the topic by breaking down the issue into smaller basic questions that focus on the core task of providing authors with back-end persistence of edits made in the web browser. Capabilities such as complex queries and multi-user access control will be considered application-centric features that are outside the scope of research.

The main content management questions that I will explore are:

Can we infer a reasonable data schema from a collection of page mockups? XML literature suggests that CTS-annotated HTML documents should be relatively straightforward to transform into possible relational schemas. I believe this is a solved part of the problem that will be necessary for system implementation but not of research value.

Is the view-update problem solved for generated schemas? When updates are made to a relational projection of data, finding the set of data-source modifications that are consistent with those updates is typically problematic. This is known as the *view-update problem* For the class of views and backing schemas possible to create with CTS, is it possible to trace updates to the DOM inside the browser back to the proper tuple—or design mockup—on the server?

How can site maps and page parameterization be expressed? Web sites often consist of many pages, each parameterized by a diverse set of signals, from URL parameters to cookies and HTTP headers. With a mockup language like CTS, what level of annotation is necessary to enable transitioning from a single-page paradigm to a multi-page one, and how should page parameterization be handled?

Can we exploit CTS annotations for better performance? It should possiblei to use knowledge of the content structure provided by CTS to aggressively cache data in the browser, offloading work from the server, reducing the conversation between client and server to simply a data "diff" exchange. This should increase throughput on the server and provide offline access to pages on the client, all without the content author having to provide custom caching code.

3.4 Migrating Existing Systems

To what extent can existing systems that rely on arbitrary code, such as WordPress, be automatically converted to a mockup-driven approach?

An important question for any new paradigm of development is the cost of switching to the new method. Even if a new method exhibits many benefits, if switching cost is high, the method is unlikely to succeed unless it is truly revolutionary. Mockup driven

development has the potential to be revolutionary from the standpoint of a community of users, as it transforms every web page into a reusable mockup[1], but from the standpoint of a single user, the improvements it offers are evolutionary. How, then, to facilitate switching?

I propose that it is possible to convert many existing CMS-powered web sites to declarative, mockup-driven sites using a form of interactive wrapper induction. Ironically, the task of wrapper induction is often applied so that the content can be extracted from a site. In the case of CMS systems, such as WordPress, the content is already available from the database, and the wrapper itself is the extraction target. I will therefore call the task *theme induction*.

WordPress themes are computer programs. It is impossible in the abstract to convert all possible themes to a simple, declarative format. But because of the particular domain of blogging, we can expect that WordPress themes are incredibly simple programs whose behavior we can hope to deduce.

I propose a hybrid method that combines active learning with tree transducer induction. By modeling a WordPress theme as a black box whose input is a data structure and which outputs a web page, we can programmatically generate a set of training examples sufficient to induce–for some subset of themes space–a declarative transformation which mimics the behavior of the theme program. This transformation will then be converted into mockup form, with CTS annotations describing bindings to the data.

This stage has three major goals. First, it will develop this theme induction algorithm for the blogging domain and test it to understand how often it succeeds within this domain. Second, it will provide a path to automated conversion from existing procedural CMS systems to a mockup-driven one, eliminating much of the switching cost. And finally, it will generate a valuable asset to the web community: a library of declarative themes which previously were only available as WordPress plugins.

4. METHODOLOGY

Each stage of the project will be paired with an evaluation methodology suited toward its particular goals.

Language and Runtime Development. The methodology used for this component will be to gather the set of capabilities necessary to enable an MDD system, construct a domain-specific language capable of achieving those capabilities, and then evaluating whether this language achieves those functional goals. The requirements gathering phase will consist of constructing inventory of the capabilities that existing template languages, theming languages, and content management systems provide to authors.

Domain Specific Languages should be evaluated based on their ability to achieve tasks important to their domain, as well as their ease of use, concision, and uniqueness in capability (i.e., why not just use an existing language?). To evaluate Cascading Tree Sheets' ability to perform domain-relevant tasks, I will construct a gallery of usage demonstrations that represent common scenarios in web development, grouped by the expertise-level of the person who might do them: beginners, who copy and modify web content from others; intermediate authors, who write and share content; and professionals, who create content explicitly for others to import and reuse.

To evaluate concision, I will compare the demos constructed with CTS to the alternatives that would have been necessary without CTS. For beginners, emphasis will be placed on labor and skills required versus the state of the art. For intermediate users, emphasis will be placed on labor required and ability to repurpose web code. For expert scenarios, I will demonstrate how CTS provides a layer of structural indirection that would have fixed several publicized encapsulation bugs with web libraries in the past.

Usability Studies. I will evaluate the mockup-driven workflow using both web-based user studies and a lines-of-code complexity analysis to perform various web authoring tasks. Three user studies will be performed, for the cases of reuse, reuse with modification, and authoring:

- For reuse, I will measure the ease with which authors can take an existing web structure and reuse it without structural modification. For example, for a university course page, a task might consist of locating, copying, and editing the HTML which represented one course announcement. Methods studied will be a CTS-powered site which separates content from design, an ordinary site which commingles the two, and a WYSIWYG system.

- For reuse with modification, I will measure how long it takes a subject to modify a site's design so that all items (such as course announcements) exhibit a new structural style. I will test the case where the style simply changes, but the content remains the same, and also the case in which new content fields are added. This latter case will require the subject to author new CTS rules, but using existing CTS rules as a guide. Methods studied will be a CTS-powered site, a traditional HTML site, and a dynamic site using an existing "knockout-style" template language.

- For the authoring case, I will first provide subjects with a short tutorial and exercises which teach them how to use CTS to author a web page. I will then ask them to construct a multi-page site which re-uses a shared structure between pages. Methods tested will be HTML with CTS and HTML without CTS.

As a final, qualitative component, I will interview web professionals to understand how they work and gather their feedback about the mockup-driven workflow I am proposing. Academic projects which seek to improve developer workflow in the vacuum of a laboratory run a huge risk of missing real-world factors that can drive development, so these interviews will provide both useful retrospective information and design advice for improvements.

Automation and Performance Automation will be approached from a relational algebra perspective. I will seek to prove a lower bound of the class of content management application that CTS-annotated mockups are capable of describing. This class of applications corresponds to the category of applications that today require procedural code to implement, but could be automatically be synthesized with an annotated mockup.

Performance will be studied using experiments which test the impact of heavy caching of both static and dynamic resources in the client browser. Additional data annotations to provide a data synchronization policy will enable dynamic data to be updated with a diff instead of with full replacement in many circumstances.

Migrating Existing Systems The migration component will consist of algorithm development and testing over a corpus of 1590 blog themes scraped from `Wordpress.org`. I will frame the problem similar to one of tree transduction. Given an example tree of structured data, representing the content of the blog, we can deterministically enumerate through all variants of this tree which represent plausible data configurations. For example, for each tree node which represents an set (such as comments on a post), we can

[1]A key differentiator from the approach offered by FORWARD, for example [**?**]

create a variant with no children, a variant with one child, and a variant with many children. By using the procedural theming engine to render one output HTML page for each variant, we are left with a set of example tree pairs—one JSON and one HTML.

Given these tree pairs, we seek to learn a declarative mapping that relates the JSON to the HTML. Literature has shown that a deterministic, top-down tree transducer can be learned for this task in polynomial time [16]. I plan to draw upon tree transduction literature to create an approach which outputs an *exemplar mockup* and set of declarative, bi-directional CTS relations instead of a transducer. I will test the efficacy of this approach by examining whether the CTS-annotated mockups produced by this process, combined with data, produce the same HTML result as the original procedural process. Themes that appear to have been recovered correctly will additionally be published online in declarative format for reuse.

5. RESULTS

Language and Runtime Development. I have completed several iterations of the Cascading Tree Sheets language, using real world development tasks and interviews with web development firms to guide each iteration. The current implementation of CTS is available for download at `treesheets.org`. I have developed several proof of concepts to evaluate CTS expressiveness as a language: basic templating tasks, converting HTML into rich Javascript widgets (such as D3) using just CSS classes, web scraping, design retargeting, and theming. A paper to be presented at WWW 2013 details the language and these examples [3].

Usability Studies. I have completed preliminary evaluations of mockup-based static content authoring, and these results in CHI 2013's student research competition track. This work tests the "reuse" scenario previously described. When editing CTS-powered web pages, participants could complete reuse tasks tasks significantly faster than with ordinary HTML: 74% faster for copying, 73% faster for pasting, and 51% faster for editing. I plan to redo this experiment with some changes, based on initial feedback from reviewers, and supplement it with the other user studies planned.

Performance. I previously published a performance study of in-browser dynamic data caching and processing in WWW 2010 [4]. This work shows that using declarative annotation, combined with a client-side templating language and data store, up to a four-fold throughput improvement can be achieved for queue-heavy workloads, such as blogs, and a doubling of throughput can be achieved by simply offloading data rendering tasks to the browser. While this work was not implemented with the CTS language, the findings apply to CTS.

Migration. I have completed scraping the dataset to use for this piece and have constructed an evaluation harness.

Remaining Work. Three chief units of work remain: completing the usability studies, proving the category of applications for which content management operations can be inferred, and developing the theme induction algorithm.

6. CONCLUSIONS

The web is the modern digital commons, a vital platform that enables everyone from authors to expressing themselves to engineers deploying applications. But web development technologies today are focused too much on process and not enough on the artifact we're trying to make. I intend to evaluate how we might construct tools which enable authors to program the web by stating *what they want* instead of *how to get what they want*.

This thesis proposes to push the edges of mockup driven web development, develop a common model for mockup annotation, and explore its feasibility for a variety of web tasks. The approach I propose is balanced: part systems building, part usability studies, and part database and applications research. These efforts will yield valuable data—as well as software artifacts—about how we might construct a next-generation web programming model that broadens the base of those able to contribute to the web and provides web experts with an even more powerful baseline.

7. REFERENCES

[1] B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and processing. *W3C Recommendation*, 2008.

[2] S. Amer-Yahia, F. Du, and J. Freire. A Comprehensive Solution to the XML-to-Relational Mapping Problem , Nov. 2004.

[3] E. Benson and D. R. Karger. Cascading Tree Sheets and Recombinant HTML: Better Reusability and Encapsulation of Web Content. In *In submission to WWW '13: Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013.

[4] E. Benson, A. Marcus, D. Karger, and S. Madden. Sync kit: a persistent client-side database caching toolkit for data intensive websites. In *WWW 2010*.

[5] D. Brickley and L. Miller. FOAF Vocabulary Specification 0.98. *Namespace Document*, 2010.

[6] K. S.-P. Chang and B. A. Myers. WebCrystal: understanding and reusing examples in web authoring. In *CHI 2012*.

[7] M. J. Fitzgerald. CopyStyler : Web design by example. *MIT Masters Thesis*, 2008.

[8] Y. Fu, K. W. Ong, Y. Papakonstantinou, and M. Petropoulos. The SQL-based all-declarative FORWARD web application development framework. *5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, 2011.

[9] P. Geneves, N. Layaida, and V. Quint. On the analysis of cascading style sheets. In *WWW '12: Proceedings of the 21st international conference on World Wide Web*. ACM, Apr. 2012.

[10] M. Hepp. *GoodRelations: An Ontology for Describing Products and Services Offers on the Web*. 2008.

[11] I. Hickson and D. Hyatt. HTML5. *W3C Working Draft*, 2011.

[12] R. Khare and T. Çelik. Microformats: A Pragmatic Path to the Semantic Web.

[13] K. Kowalzcykowski, K. W. Ong, K. K. Zhao, A. Deutsch, Y. Papakonstantinou, and M. Petropoulos. Do-It-Yourself Database-Driven Web Applications , Jan. 2009.

[14] R. Krishnamurthy, R. Kaushik, and J. F. Naughton. *XML-to-SQL Query Translation Literature: The State of the Art and Open Problems*. Lecture Notes in Computer Science, 2003.

[15] R. Kumar, J. O. Talton, S. Ahmad, and S. R. Klemmer. Bricolage: example-based retargeting for web design. In *CHI 2011*.

[16] A. Lemay, S. Maneth, and J. Niehren. A learning algorithm for top-down xml transformations. *PODS 2009*.

[17] S. Oney and B. Myers. FireCrystal: Understanding interactive behaviors in dynamic web pages. In *VLHCC 2009*.

[18] F. Yang, N. Gupta, N. Gerner, X. Qi, A. Demers, J. Gehrke, and J. Shanmugasundaram. A unified platform for data driven web applications with automatic client-server partitioning. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 341–350, New York, NY, USA, 2007. ACM.

[19] F. Yang, J. Shanmugasundaram, M. Riedewald, and J. Gehrke. Hilda: A high-level language for data-drivenweb applications. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 32, Washington, DC, USA, 2006. IEEE Computer Society.