Declarative Web 2.0

Erik Wilde
School of Information
UC Berkeley

Abstract

Web 2.0 applications have become popular as drivers of new types of Web content, but they have also introduced a new level of interface design in Web development; they are focusing on richer interfaces, user-generated content, and better interworking of Web-based applications. The current foundations of the Web 2.0, however, are strictly imperative in nature, which makes it difficult to develop applications which are robust, interoperable, and backwards compatible. Using a declarative approach for Web 2.0 applications, this new wave of applications can be built on a more robust foundation which is more in line with the Web's style of using declarative methods whenever possible. We show a path how today's imperative Web 2.0 applications can be regarded as a testbed as well as a first implementation for a revised version of Web 2.0 technologies, which will be based on declarative markup rather than imperative code.

1 Introduction

Fueling the second wave of the Web's popularity, the term *Web 2.0* has become highly popular for describing a new breed of Web applications. It is mainly perceived to refer to two major areas, one being user-generated content and the social networks around it, and the other being a set of technologies providing support for richer interfaces for Web applications, bringing the user experience of Web applications closer to that of desktop applications.

In this paper, we refer to both areas; we argue that the contextual nature of Web 2.0 content needs a better representation, and we also argue that the same representation can be used to better describe the rich interfaces for applications building on that contextual content. Our claim is that a *declarative way* of representing Web 2.0 data provides benefits in various areas, overall leading to a *Declarative Web 2.0*, which is superior to the current Web 2.0 in terms of machine-readability and loose coupling. A *declarative* style of describing a task focuses on describing the task itself, rather than describing how the task has to be solved.

While a declarative Web 2.0 represents core information in structured documents, the current Web 2.0 heavily relies on imperative principles, with a multitude of JavaScript libraries providing largely overlapping sets of functionality. Switching between these libraries is hard, because they use different APIs and underlying models and thus tightly couple user code with a particular library.

One of the main reasons why a declarative approach is preferable over imperative programming is that declarative languages provide a better foundation for versioning and extension, as demonstrated by the Web's foundation, the *Hypertext Markup Language (HTML)*. Because HTML is designed to tolerate versioning and extensions, older browsers can be used to render Web pages designed for newer Web browsers; they will render these pages as good as possible[1] within the limitations of the older language.

One area where this issue becomes very visible is in the area of *accessibility*. While today's rich Ajax applications provide a much better user experience for most users, they often completely break down in cases where Ajax's programming-oriented approach does not work, for example when Web pages should be rendered by voice browsers. Again, declarative approaches would be highly superior for this scenario, because a declarative application could be used by two very different clients, one being a visual browser with advanced interface capabilities, and the other being a voice browser, which still could access all relevant application information for rendering it.

Another problematic area of the current imperative Web 2.0 approach is *security*, because everything revolves around code and sometimes very language-specific technologies. Code injection and other security threats are a serious issue, and the number of security bulletins about Ajax-related areas is steadily increasing.

A third area where the current Web 2.0's shortcoming become apparent is the area of *mashups*, which are applications combining content from different sources. Because most mashups are based on API-style approaches rather

[1] This, of course, depends on the page authors to design their pages in a way which is compliant with HTML's idea of graceful degradation, rather than building Web pages that will fail completely in older browsers.

than on declarative markup, they tend to be very brittle, and in fact most current mashups break very easily when any of the mashed up applications changes their interface. In this area, it becomes apparent that API-style coupling is not a good foundation for loosely coupled systems, and a more REST-oriented approach is a more promising way for building loosely coupled Web 2.0 mashups.

In Section 2, we take a closer look on some of the most popular Web 2.0 application and technologies, trying to establish a set of common properties which can be used as starting point for designing declarative structures. The two most important areas where a declarative approach would have very real advantages are interface design, discussed in Section 3, and the loose coupling of distributed applications, discussed in Section 4. Based on these observations, Section 5 contains a proposal for declarative Web 2.0 resources. Section 6 contains references to related work from different application areas, trying to put the "Declarative Web 2.0" approach into the larger context of Web architecture. Section 7 concludes the paper with final remarks and some ideas for further work in this area.

2 Web 2.0

The term "Web 2.0" was coined in 2004 to label some common characteristics of the most successful Web developments after the bursting of the dot.com bubble. Since the term has been invented to describe some ongoing activities in the Web landscape, it does not have a strict definition, but it has some important major facets, such as *seeing the Web as a platform*, *harnessing collective intelligence*, *being data-centered*, *using Web-based applications*, *using lightweight programming models*, *using cross-device applications*, and *providing rich user experiences*.[2]

This of course is a very diverse list of issues, and it is intended to be, because its main goal is to describe a set of properties which are shared by many successful Web companies, and which probably will be important for the future development of the Web. For a more structured view of these issues, we look at Web 2.0 from two different perspectives, one being the perspective of Web 2.0 application discussed in Section 2.1, and the other being the perspective of Web 2.0 technologies, discussed in Section 2.2. Based on these two perspectives, we then continue by describing more specifically what the problems and opportunities of the current state of the art in these areas are.

2.1 Web 2.0 Applications

The four first issues from the above list of Web 2.0 facets can be used to characterize the majority of successful Web 2.0 applications. Many Web 2.0 applications focus on providing a platform for social structures, knowledge, and data which already exists, but so far has not been combined in a useful way. The general principle of Web 2.0 applications is to try to build an application in this design space, which then should attract a sufficient number of users to gain traction. Once this has happened, various business models exist how to turn such an application into a profitable business.

The interesting observation in the area of Web 2.0 applications is that they generally focus on the value of making connections between things which already exist, may it be people or data. This in essence means that Web 2.0 applications often are more about formalizing structures, than creating new content or new structures. By formalizing structures, these applications create a platform which can then be used for interested parties to fill in instances into the structural framework. A classical example are social networking platforms, which create some kind of formalization of how a social network is structured, and the actual social structures are supplied by the users, filling in details about themselves and the connections they have with other things (which may be other users or any other identifiable object[3]).

For this paper, the important observation is that as a common pattern among Web 2.0 applications, they formalize existing connections between existing things and then make them available for previously impossible tasks. By looking at applications from this perspective, the focus shifts from that of *content* to that of *context*: Instead of working with content, they formalize context (where context is a specialized version of content which always connects existing resources, maybe in connection with annotations for a richer description of the context).

Blogs are an interesting example for this kind of shift from content to context: While the general blog model allows the completely detached existence of a blogger, writing entries which consist only of text, the appeal of blogs to many people, and the interesting facet of the blogspace as a technical and social phenomenon, is the fact that blog entries frequently refer to Web pages and other blog entries, thus forming a tightly connected network of contextualized resources [9]. Thus, blog posts can be seen as context, and a blog then is nothing more than a stream of these context objects, each adding a new set of connections between the resources of the blogspace.

At this point it is worth pointing out that there is some correspondence between this general principle of Web 2.0 applications as context-processing platforms, and the Web architecture [7], which states that "to achieve the goal [to build a global community in which any party can share in-

[2] All these points are quoted from a September 2005 article by Tim O'Reilly (`http://www.oreillynet.com/lpt/a/6228`).

[3] In this context it is interesting to see that in a popular application such as *MySpace*, users invent creative "misuses" of the formalization, for example celebrities and even brands assuming the role of a "user" and building "social" connections with large numbers of other users.

formation with any other party] , the Web makes use of a single global identification system: the URI. URIs are a cornerstone of Web architecture, providing identification that is common across the Web." Consequently, to build Web 2.0 applications in a way conforming with the Web architecture, all objects of interest in the context of the application would have to be made identifiable[4] by URIs. As pointed out in Section 4, such a design would also be the best foundation for loose coupling. Today, only few Web 2.0 applications are really based on this Web architecture principle, many use models where important resources are not exposed through URIs, which makes it impossible to reuse these applications in a truly Web-oriented way.

2.2 Web 2.0 Technologies

For the technical side of the Web 2.0 facets, the three remaining points of the previously listed Web 2.0 facets apply. This can be summarized into something which often is being called *Asynchronous JavaScript and XML (Ajax)*, which is a combination of *Dynamic HTML (DHTML)* — the ability of JavaScript to react to HTML events and dynamically change the HTML's DOM structure — and the ability to contact servers via HTTP from within JavaScript code [11]. Many of today's popular Web 2.0 applications make extensive use of Ajax to provide a better user experience than the old click/refresh model of Web applications.

From a technical point of view, the Web 2.0 thus is a rather simple, it relies on better JavaScript and DOM implementations in the browsers and the ability to use HTTP from within JavaScript. The biggest problem with this approach, however, is that it violates the principle of "separation of content, presentation, and interaction" [7]. The very practical consequence of this is the fact that today's Web 2.0 application have major accessibility problems, because the interfaces are well-designed for regular users in visually oriented browsers, but because of the presentation and interaction being buried in JavaScript code, non-visual browsers have no way of providing access to the application, for example through a voice-based interface.

This means that even for a simple technical solution on how to build better interfaces for Web applications, the imperative approach is not a good solution. It can be regarded as a useful mechanism for a phase of experimentation, but for a consolidation and better engineered approach, a new mechanism is required. The same thing happened in HTML, which became increasingly interspersed with formatting constructs, until it was decided to factor these out into a dedicated stylesheet language, such as CSS. Newer versions of CSS allow effects which previously were only possible with JavaScript, and they can be used by a much larger class of devices than JavaScript authored specifically for visual browsers, because CSS is purely declarative.

What we propose is a second wave of identifying functionality which is reused by many Web developers and making it available through a declarative language, thus restoring the separation of content, presentation, and interaction, which has been violated by Ajax on a large scale.

3 Rich Internet Applications

Ajax is the technical foundation of what is often referred to as the concept of a *Rich Internet Application (RIA)*, which tries to more closely match the user interface experience of a desktop application. The RIA approach has become feasible because JavaScript and DOM implementations of the major browsers have improved to a point where advanced cross-browser scripting becomes possible, and the remaining compatibility issues are resolved by widely available JavaScript libraries. While this approach has been successful in creating a diverse landscape of RIA applications, these are not easy to maintain. Whenever a new browser version is released, it may break something in a library, and then the library has to be updated to also support this new version. Likewise, when a new library is released, it may introduce changes to its functionality, which then requires applications building on it to be changed. This pattern of propagated changes can make site maintenance very expensive, in particular if a site relies on more than one library.

The main problem is that JavaScript libraries are tightly coupled with the JavaScript implementation of browsers, and Web applications are tightly coupled with the libraries they are using. Together, this creates a ripple-through effect where every change in the browser or library landscape has severe consequences for applications depending on these components. And because the model of interaction between libraries and browsers, and applications and libraries, is based on the API-style of coupling, instead of degrading gracefully, applications break down completely when something in their foundation changes. With the wide-spread adoption of RIA interfaces, this problem has become widespread, and more robust and better designed JavaScript libraries are only a short-time solution.

While the term "declarative Web 2.0" has not been popularized so far, the term "declarative Ajax" has gained some traction. This shows that the community of developers working on RIAs has discovered that the current model of API-style interfaces proves to be not very robust. Versioning issues with new versions of Ajax libraries and the inability to switch between Ajax libraries are an indication of the fact that the coupling between the abstract interface design issues, and the technologies used for implementing

[4] It is important to point out that *identifiable* is not the same as *accessible*, and the Web in fact does not require that URIs must identify accessible resources (XML Namespaces are an excellent example for URI-identified resources which are never accessed).

these designs, is too tight. While we think that the term "declarative Ajax" in a way is a contradiction in itself (if something is tightly bound to a programming language, it is not declarative), we believe that the discussions in this area is a sign that the need for a more robust approach is increasingly being recognized.

From the technology perspective, this means that the Web 2.0 needs a declarative way to represent the Ajax style of building RIA applications. This is primarily a question of representing presentation and interaction within the context of a single Web application, and Section 5 outlines a way how this can be done. Before this, however, Section 4 looks at the different but similar issues that the mashup facet of Web 2.0 applications raises.

4 Loose Coupling

Web 2.0 applications can be regarded as formalizing context, and giving users a platform to capture context, share context, and use and capitalize context by mining it in ways which before the Web 2.0 were not available. The most interesting examples of Web 2.0 applications are *mashups*, which take context from other applications, and then reuse it in a new and different application scenario. Web 2.0 application often explicitly encourage this pattern of application development by publishing some sort of API.

Most applications exposing APIs either provide a SOAP [5] or a REST [3] interface. SOAP interfaces have significantly declined in popularity, because the API-style of SOAP introduces all the problems of tight coupling mentioned earlier, and also SOAP requires a lot of tools to be used, and it makes it hard to build pure client-side mashups.[5] For this reason, REST-style interfaces have become more popular, but only few of these "REST interfaces" are well designed, and most importantly, they expose the data of the application in a proprietary format.

For successful communications between cooperating applications, it is necessary to share a common understanding of the data that is being exchanged, and the current model of REST interfaces makes it necessary to achieve this common understanding individually for each pair of communicating applications, because there is no common data model across Web 2.0 applications. We propose such a model in the following section, which would provide a minimal data model being shared across applications, so that cooperation among applications requires less coordination and has less dependencies on proprietary data formats.

[5]For client-side integration, SOAP is a significant hurdle, because a complete client-side implementation of SOAP is a rather heavyweight component.

5 Web 2.0 Resources

For a declarative Web 2.0, there must be data format which captures the essential facets of the resources being used, and since we identify *context* to be the underlying common concept of Web 2.0 data, it must be a data format for expressing the associations of multiple resources. The W3C has standardized a language for this, which is called the *XML Linking Language (XLink)* [2]. The XLink specification was unsuccessful for a variety of reasons, but in the new context of the Web 2.0, it provides a very convenient way of expressing the contextual nature of data. For example, Figure 1 shows how a blog post can be expressed as an XLink, we call such a resource a *blog link (blink)*.

All `xlink`-prefixed attributes are XLink standard attributes (the XLink vocabulary uses attributes only), with the `xlink:type` attribute determining the XLink type of each element. For link structures, two major categories of elements can be distinguished:

- *Resources:* These elements represent resources participating in a link. Resources can be either referenced using a URI (which then appears in the `xlink:href` attribute) with `locator` elements, or they can be embedded into the link by using `resource` elements. Resources participating in a link are simply part of the link, and for blinks, we classify them using the `xlink:label` attribute, which is an indication of the role of the resource in the context of the blink.

- *Arcs:* Elements being of type `arc` represent connections between resources of a link; these are the traversable paths between blink resources. Arcs connect pairs of resources based on their `label`s, and each arc specifies `from` which label it is going `to` which label. If there are more resources tagged with an arc's labels, then the arc represents more than one connection between a pair of resources.

The resources in the blink have been extracted from the blog post (which means that the blink simply is a different representation of the blog post). They are a representation of blog post content and metadata. The resources are the blog post as permalink (`<permalink>`) and as HTML fragment (`<post>`), links to the blog's home page (`<blog>`), the blog's author (`<author>`), and the blog's feed (`<feed>`), as well as a number of URIs which have been mentioned in the blog post (`<posturi>`), and a time stamp indicating the post date (`<timestamp>`). Only the blog post and the time stamp are embedded resources, all other participating resources are URI references to Web resources.

The important observation is that the majority of Web 2.0 resources can be represented as XLinks, for example data

```
<blink xlink:type="extended" xmlns:xlink="http://www.w3.org/1999/xlink" xlink:title="Sorry Pluto...">
 <permalink xlink:type="locator" xlink:title="Sorry Pluto..." xlink:label="permalink"
   xlink:href="http://docordie.blogspot.com/2006/08/sorry-pluto-and-some-thoughts-about.html"/>
 <author xlink:type="locator" xlink:title="Bob Glushko" xlink:label="author"
   xlink:href="http://www.ischool.berkeley.edu/~glushko/"/>
 <blog xlink:type="locator" xlink:title="Doc Or Die" xlink:label="blog"
   xlink:href="http://docordie.blogspot.com/"/>
 <feed xlink:type="locator" xlink:title="'Doc Or Die' Atom Feed" xlink:label="feed"
   xlink:href="http://docordie.blogspot.com/atom.xml"/>
 <posturi xlink:type="locator" xlink:label="site"
   xlink:href="http://en.wikipedia.org/wiki/International_Astronomical_Union"/>
 <posturi xlink:type="locator" xlink:label="site"
   xlink:href="http://www.modbee.com/local/story/12638580p-13341408c.html"/>
 <posturi xlink:type="locator" xlink:label="site"
   xlink:href="http://www.freenewmexican.com/news/48332.html"/>
 <posturi xlink:type="locator" xlink:label="site"
   xlink:href="http://www.miami.com/mld/miamiherald/living/15365590.htm"/>
 <post xlink:type="resource" xlink:label="post" xlink:title="Sorry Pluto..." type="text/html">
  <h2 xmlns="http://www.w3.org/1999/xhtml">Sorry Pluto...</h2>
  <!-- complete HTML snippet of blog post -->
 </post>
 <timestamp xlink:type="resource" xlink:label="timestamp" type="xs:date">2006-08-27</timestamp>
 <arc xlink:type="arc" xlink:from="permalink" xlink:to="timestamp" xlink:title="Post Time"/>
 <arc xlink:type="arc" xlink:from="site" xlink:to="post" xlink:title="Blog Post"/>
 <arc xlink:type="arc" xlink:from="permalink" xlink:to="author" xlink:title="Post Author"/>
 <arc xlink:type="arc" xlink:from="permalink" xlink:to="blog" xlink:title="Blog Home"/>
 <arc xlink:type="arc" xlink:from="blog" xlink:to="author" xlink:title="Blog Author"/>
 <arc xlink:type="arc" xlink:from="blog" xlink:to="feed" xlink:title="Blog Feed"/>
 <arc xlink:type="arc" xlink:from="author" xlink:to="blog" xlink:title="Authored Blogs"/>
</blink>
```

Figure 1. Blog Post Represented as XLink

from a social networking site could represent a person's profile by mapping the networking facets (the connections with other persons, groups, and other identifiable resources) to XLink resources, in effect turning the profile into an XLink. This link would use different `labels` and `arcs`, but the framework (the fact that it is an XLink) remains the same.

The RIA support outlined in Section 3 could also be represented by XLinks, because XLink supports *behavior attributes* for controlling user interaction with a link. This means that a significant subset of RIA interactions (those which, based on user interaction, update the DOM and/or use HTTP requests) could also be modeled by XLinks, even though XLink's current set of behaviors is too restricted and most likely would have to be extended to better serve the demands of RIA developers. But given a sufficiently powerful XLink, many parts of RIA interactions could be specified declaratively, not relying on any specific JavaScript library.

The biggest disadvantage of XLink in its current state is that the specification has a number of technical shortcomings, and that there are no complementary specifications for how to retrieve XLinks over HTTP, or how to render them. While the *Atom Publishing Protocol (APP)* [4] is a very promising candidate for the first scenario, the presentation problem so far is unsolved.

It could be argued that structures such as the blink shown in Figure 1 are clearly establishing relationships between URI-identified resources and thus should be represented based on the *Resource Description Framework (RDF)* [8], which is the W3C's generic format for metadata. However, we argue that RDF so far has failed to deliver real benefits on a large scale, and that specialized semantics are easier to understand and use and thus more likely to be adopted by Web developers. The *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)* [1] approach could be an easy way to transform XLinks to RDF, though, if for some reason applications wanted to get an RDF representation of declarative Web 2.0 resources using XLink syntax.

6 Related Work

The declarative Web 2.0 approach presented here has connections with many areas of Web and information integration technologies. From the Web technology perspective, the question is which abstractions are widely used so that they warrant representation in a declarative format. CSS is the example which demonstrated that given enough users and interest from the developer side, it is possible to create new abstractions on the Web for frequently occurring design patterns. We believe that the Web 2.0 has shown emerging patterns, and that the approach of capturing these in a resource format based on context representation is a promising approach. The ability to use XLink as the

foundation for the Web as an open hypermedia system [10] (which is not discussed in this paper) also addresses many issues which are typical for Web 2.0 applications.

From the information integration perspective, the problem described in this paper can be regarded as being similar to the problem of *Enterprise Information Integration (EII)* [6], which deals with the problem of how to answer queries over distributed heterogeneous sources. However, EII has a number of different focuses, most notably it (a) can afford to build on some fairly complex infrastructure, because EII problems typically are solved in large companies which can afford to deploy expensive tools, and (b) has the goal of integrating all heterogeneous sources into one unified model, whereas the approach discussed in this paper is more targeted at exposing a minimal set of shared structures among Web 2.0 applications, without the claim to provide an exhaustive model for these applications.

Another area which has many connections with the work discussed here is that of *Web Services* and the *Service Oriented Architecture (SOA)* in general. The current debate around SOAP/WSDL-based Web services or the REST style closely mirrors the API-style of the current Web 2.0 applications, and the resource-oriented style for the declarative Web 2.0. In both scenarios, API-style technologies tend to work at least in an acceptable way as long as they are used in a scenario which is governed by one single controlling entity, but they easily fail in a loosely coupled scenario with independent versioning and extensions [12].

One of the interesting facets of the work presented here is that it is located in the area where the Web as an information system and Web-based applications as traditionally engineered software systems are now merging. Since its beginning, the Web often has been regarded as a new interface to applications, but not really as *one application*. The SOAP/WSDL Web services area also shows this rather clearly, with SOAP using HTTP as a transport protocol and then simply re-creating all the middleware functionality from the traditional discipline of distributed systems on top of this. The Web *as an application* may still need some more time to become widely accepted, and Web 2.0 is an important development along this path.

7 Conclusions

In the same way as HTML experimentation and unintended uses, such as using tables for page layout, paved the path for CSS, enabling a design of a language which supported the functionality required by real-world Web users, we believe the current wave of Ajax experimentation paves the path for a new way of Web applications, which ultimately should be based on declarative technologies. Factoring out the commonalities of Web 2.0 and coming up with a model that covers a large percentage of Web 2.0 use cases without being overly complicated is not an easy task, but it should be done to reestablish one of the Web's fundamental principles, without abandoning the exciting applications and technologies which have fueled the Web 2.0 wave.

While XLink has been standardized for a while now and has various problems associated with it, its general approach looks promising as an approach of how to represent common structures for Web 2.0 applications. We do think, however, that the current state of the specification calls for a major upgrade, since many of the standard's technical shortcomings cannot be fixed in a backwards compatible way. An improved XLink in conjunction with well-defined link retrieval and link presentation could then become the foundation for the declarative Web 2.0.

References

[1] DAN CONNOLLY. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). World Wide Web Consortium, Candidate Recommendation CR-grddl-20070502, May 2007.

[2] STEVEN J. DEROSE, EVE MALER, and DAVID ORCHARD. XML Linking Language (XLink) Version 1.0. World Wide Web Consortium, Recommendation REC-xlink-20010627, June 2001.

[3] ROY T. FIELDING. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, Irvine, California, 2000.

[4] JOE GREGORIO and BILL DE HÓRA. The Atom Publishing Protocol. Internet Draft draft-ietf-atompub-protocol-15, May 2007.

[5] MARTIN GUDGIN, MARC HADLEY, NOAH MENDELSOHN, JEAN-JACQUES MOREAU, HENRIK FRYSTYK NIELSEN, ANISH KARMARKAR, and YVES LAFON. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). World Wide Web Consortium, Recommendation REC-soap12-part1-20070427, April 2007.

[6] ALON Y. HALEVY, NAVEEN ASHISH, DINA BITTON, MICHAEL J. CAREY, DENISE DRAPER, JEFF POLLOCK, ARNON ROSENTHAL, and VISHAL SIKKA. Enterprise Information Integration: Successes, Challenges and Controversies. In FATMA ÖZCAN, editor, *Proceedings of the ACM SIGMOD 2005 International Conference on Management of Data*, pages 778–787, Baltimore, Maryland, June 2005. ACM Press.

[7] IAN JACOBS and NORMAN WALSH. Architecture of the World Wide Web, Volume One. World Wide Web Consortium, Recommendation REC-webarch-20041215, December 2004.

[8] GRAHAM KLYNE and JEREMY J. CARROLL. Resource Description Framework (RDF): Concepts and Abstract Syntax. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210, February 2004.

[9] RAVI KUMAR, JASMINE NOVAK, PRABHAKAR RAGHAVAN, and ANDREW TOMKINS. Structure and Evolution of Blogspace. *Communications of the ACM*, 47(12):35–39, December 2004.

[10] DAVID LOWE and ERIK WILDE. Improving Web Linking Using XLink. In *Proceedings of Open Publish 2001*, Sydney, Australia, July 2001.

[11] ANNE VAN KESTEREN. The XMLHttpRequest Object. World Wide Web Consortium, Working Draft WD-XMLHttpRequest-20070618, June 2007.

[12] ERIK WILDE. What are you talking about? In *Proceedings of the 2007 IEEE International Conference on Services Computing*, Salt Lake City, Utah, July 2007.