Future Trends in Web Development

Simon Heimler, University of Applied Sciences Augsburg

January 17, 2015

Abstract. Where is the web of the future heading? Nobody can predict this for sure, but nevertheless it might be interesting to observe some trends that loom ahead. This article will give a brief overview of some possible technological foundations of the future web: ECMAScript Harmony, Web Components and the Semantic Web.
After that some roles the web of the future might increasingly play are outlined. We seem to be heading toward a ubiquitous web, becoming the most widespread and available platform for applications and increasingly services, too.

Keywords: Web, Internet, JavaScript, Web Components, Semantic Web, Services, API, Internet of Things

Current Trends in Software Systems, Computing, and Information Systems
Prof. Dr.-Ing. Christian Märtin

Faculty of Computer Science
University of Applied Sciences Augsburg

1 Introduction

Writing about the future is a risky, since history can prove the predicting person wrong very easily. Nevertheless the author is convinced that thinking and speculating about the future is ever more important in a world that is changing so fast. It requires some open mindedness about the possibilities where we are going. The future may surprise us in many ways, but some parts of it can and should be anticipated by attentive people.

It is a myth that the future comes from nowhere. It is always rooted in the past and learning about the history can help predicting the future. Even in a time where a lot of technology happens for the first time, they usually have dozens of years of research and development behind. Tablets, for example, date back as far as to the 50s[1].

The Web has an interesting relationship to the future. Since its architecture is decentralized, no one has direct control were the web is heading. Users and developers set the course for the future of the web, for better or worse. Trends nobody originally anticipated, like rich web applications, emerge and scratch hard on the limits on the currently available technology.

After a new trend has established itself or shown enough promise, the official development and standardization process might pick it up, adding better technological support and making it a native part of the web technology stack. This may seem like a slow, sometimes rough process but it is truly "bottom up", fitting the decentralized nature of the web.

One of the obvious disadvantages of this is process is, that technical advancement has to wait for developer and user adoption. Since many people use outdated browsers, the current web is a compromise of what is working for the majority of them. Because of that the current state of the web always lags years behind the new ideas and standards.

Chapter 2 will introduce some key technologies that are currently in development and may build the foundation of the future web. Chapter 3 will focus on roles the future web may increasingly play. Both will not go deep into technical details but outline the main ideas and the implications those trends might have.

2 Technological Foundations of the Future Web

2.1 ECMAScript Harmony

Terms and current state

ECMAScript[2] may not be a familiar term to many people. This is because it is more commonly known as "JavaScript" - the programming language of the browser. The term "Harmony" refers to the two upcoming ECMAScript 6 and ECMAScript 7 standards of the language.

---

1 Dimond 1957

2 ECMAScript community 2014

ECMAScript 6 is currently in working draft[3] state with a feature freeze already in place. The official publication date is expected to be June 2015[4].

Of course even after the official publication it will take months to years for the browsers to implement ES6 completely and many years until the majority of users have their browser updated to support those features. Until then developers can start using parts of ES Harmony through transpilers, like Google Traceur[5]. They allow for writing more modern JavaScript and compiling it to ES3 or ES5 for outdated browsers. Some features of ES6 can be retrofitted to older browsers through small JavaScript libraries, called shims[6]. This process is called feature modernization.

Do not break backward compatibility

ES Harmony has its name for a reason. A definite goal of the new standard is to remain completely backward compatible and introduce no breaking changes. No old features are removed or "repaired". Instead ES Harmony adds new features that can be used in place of the old features. This ensures that old, unmaintained code will continue to work on modern browsers.

A good example for this is the new `let` variable declaration[7] which uses block-scope variable declaration instead of function-scope with `var`. There is a new `for … of` loop[8] which solves some problems of the current for … in loop and has elegant support for iterators. In both cases it was decided not to fix the current implementation but to add a new one to the language.

This choice of keeping deprecated features does even more increase the need of sub-setting JavaScript[9] through defined coding guidelines / style guides, since the old (potentially dangerous) features are still available.

A better language for complex applications

An important goal is to make JavaScript better at writing big, complex applications since this is where the web has grown to the last years. JavaScript was never meant to be used in this scale, but it happened anyway. The most important addition is the new module system[10] which lets developers easily organize their application in a modular fashion. Right now this has to be done through 3rd party libraries, with competing (and incompatible) standards, like AMD[11] or CommonJS[12].

---

3 Ecma TC39 2014
4 Axel Rauschmayer 2014
5 https://github.com/google/traceur-compiler
6 Example: https://github.com/paulmillr/es6-shim/
7 Ecma TC39 2014, p. 222
8 Ecma TC39 2014, p. 242
9 Crockford 2008
10 Ecma TC39 2014, p. 301
11 http://requirejs.org/docs/whyamd.html
12 http://wiki.commonjs.org/wiki/CommonJS

A better language for code generators

JavaScript is increasingly used as a target language of code generators. Programs that are written in C++, CoffeScript, Dart, etc. are compiled into machine-optimized JavaScript. While there are projects that add additional optimizing hints to the code, like asm.js, this will be increasingly supported by JavaScript natively by features like Typed Arrays[13].

A better language in general

There are a lot of features that make writing JavaScript a more clean and modern experience. Some patterns that are increasingly used by developers through libraries like the promise pattern[14] will go native. Some new ES6 additions only add "syntactic sugar", which means shorter and cleaner ways to write code, without adding a truly new feature. The new class notation[15] is such a feature. It is just a more convenient (and for many programmers more familiar) way to write object oriented code in JavaScript.

2.2 Web Components

Current State

Web Components are another technology that is currently in standardization process. There are a few W3C specs, most of them in working draft state. One part – templates – is already included in the final, recommended HTML5 Spec[16].

Like ES6 the adoption of the feature depends on the browser and user adoption. Meanwhile there are shims (a script that add support for modern features in older browsers) which could speed this up significantly.

Web Components have complete support in Google Chrome and partial in Firefox. However it's currently not entirely sure if Safari and IE start to implement it too. Since Web Components is a rather big extension to the current browsers, those shims are rather heavy and complex in nature – which has significant size and performance implications[17]. That makes the future of Web Components somewhat uncertain, which is a pity since it is one of the most interesting and promising new approaches to web development in the opinion of the author.

What problems Web Components solve

As mentioned in the introduction, the web is (increasingly) developed from bottom up. Web Components empower this approach through giving web-developers the ability to create their own, custom HTML elements or alter the native ones. This is

---

[13] Ecma TC39 2014, pp. 486ff
[14] Ecma TC39 2014, p. 557
[15] Ecma TC39 2014, pp. 286ff
[16] Ian Hickson et al. 2014
[17] TJ VanToll 2014

made possible by allowing web-developers a more low-level access to extend the browser itself.

Since Web Components work like regular HTML elements, they are easy to use for non-technical users. They feel like plugins: The user adds a new component to his website and now he can use additional elements and features, just by adding HTML.

Developing new browser features through Web Components enforces (or at least encourages) a more modular programming style. Unwanted interdependencies can be easily avoided. This makes WebComponents easy to reuse.

Example: Responsive images

Responsive images are a good example where Web Components would come in handy. There is a definite need for responsive images in the browser. The current image tag does not support conditional loading of images. With the great variety of display sizes and resolutions that came up the last years, it became an important feature to support those in a smart way.

The classic "bottom up" way would be to hack around that limitation through libraries and wait for the W3C and browser vendors to add native support. In fact, this is what's currently happening with the proposed <picture> element[18].

With Web Components developers can get a clean solution much faster: They simply create a new HTML tag, e.g. `<x-picture>`[19]. This new image element has support for conditionally loading different resolutions.

It's likely that a lot of different approaches emerge and after one establishes itself more broadly the W3C could use it as a basis for developing a completely native element. If the element happens to be a rather special use case it won't and it doesn't hurt because the Web Component alternative works just as fine.

In any case, users and developers don't have to wait for standardization to get access to clean, new HTML elements and features.

Custom Elements

Creating custom elements[20] is the main feature of Web Components. This approach has some advantages over the current libraries: It not only blends completely into the way the browser works natively, the API of the new component is simple HTML. Behavior or appearance can be altered by adjusting the content and attributes of it. This makes it very easy to use and re-use, since it is basically just an HTML embed code.

Shadow DOM.

Native Elements hide their implementation and inner working from both users and developers. This is done through the Shadow DOM[21], which is a separate, inaccessi-

[18] https://html.spec.whatwg.org/multipage/embedded-content.html#the-picture-element
[19] https://github.com/ResponsiveImagesCG/x-picture
[20] Dimitri Glazkov 2013
[21] Dimitri Glazkov, Hayato Ito 2014

ble DOM tree just for that element. It contains further HTML elements that are rendered and calculated by the browser.

Web Components essentially open up the Shadow DOM for developers, allowing them to use it for hiding complexity and preventing unwanted manipulation from outside, which could mess up the logic or style of the element. That way, different components can't incidentally interfere with each other.

HTML Imports

Currently libraries require the web-developer to add new script and CSS-link tags to their HTML, which adds new requests the browser has to handle. After that some more HTML is usually required to create the frame the library is working in. With the new HTML Imports feature[22] this can be aggregated into one module import that consists of HTML with JavaScript and CSS already embedded. This makes including Web Components even more convenient.

2.3 Semantic Web and Linked Data

Terms and current state

There are two common used Terms that refer to the same ideas and technologies: Semantic Web is the more official name, while Linked Data has a more pragmatic, data focused, connotation. From a technical perspective, the Semantic Web consists of various concepts, technologies and formats, many of them already W3C standardized. This is called the Semantic Web Stack[23].

The idea was introduced by Tim-Berners Lee, the inventor of the web. To quote him, describing the Vision of the Semantic Web:

„The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."[24]

To sum it up, the Semantic Web is not a replacement of the current web, but an extension – an additional layer. That layer defines the actual meaning of the content in a machine-accessible way. The goal is to lay the groundwork for a better human-machine cooperation.

While the technology part has seen over a decade of development and standardization, the Semantic Web is has yet to become an everyday reality for web users. It is currently mainly used in academia and government. However, big web companies like Google, Microsoft and Facebook started to adopt it lately.

One of the problems of the Semantic Web approach is, that it is rather "top-down", meaning a rather small group of people decided that this needs to be done in order to get a better and smarter web. Such top-down approaches can drive the future in a much faster and more profound way, but they are also prone to fail if they don't get the critical mass of adoption.

---

[22] Dimitri Glazkov, Hajime Morrita 2014

[23] Tim Berners-Lee 2007 http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24)

[24] Tim Berners-Lee et al. 2001

Semantic annotation

Machines have a hard time understanding the actual meaning of the content of a website. This is because the web is almost exclusively optimized for humans.

Semantic Annotation is the process of adding additional layer of machine readable and interpretable information about the content and its meaning (semantics). Those additional information are stored in Triples. The Semantic Web standardized this concept of a triple-centric data structure as RDF[25]. Each triple declares a simple grammatical sentence, consisting of a subject, predicate and an object. Each is an URI in nature, which are unique and can also refer to abstract resources and concepts like real-world things and concepts.

To store those information in a decentralized way, they can be embedded into regular webpages or provided under their own URI. RDF is only an abstract concept of a data format, but there are many concrete data serialization formats, supporting HTML, XML, JSON or plain-text as container format.

The most obvious field of application for this is SEO. Big search-machine companies like Google, Microsoft, Yandex and Yahoo joined forces to create the schema.org[26] project, a common vocabulary for talking about things in the web. With the big search machines supporting Semantic Annotations, there is real SEO value in providing it. This may even be the main driver for the adoption of Semantic Web Technologies.

Knowledge aggregation and creation

If RDF statements share the same entity (URI) they get linked together and a graph structure emerges. Graph structures are especially powerful, since they are schemaless. Merging data and knowledge from different sources is possible without having to (manually) provide hints how the computer should do it.

The Semantic Web offers many additional technologies for aggregating, storing, querying and interpreting that data. Ontologies[27] allow to declare a vocabulary along with a schema (concept) of a domain. This allows to do reasoning with the given data, inferring new knowledge through description logics[28].

Sharing the context of the word

Since ontologies can describe specific domains and the model, it makes it an interesting candidate for machine-to-machine communication and APIs/Services. Humans share a common vocabulary and concept of the world, this is called language and culture. Computers do not have that, so they have to be very strict (and thus not smart) about incoming information. If the computer has an ontology it can do its own reasoning and share the logic about how to interpret the data.

---

[25] Eric Miller, Frank Manola 2004

[26] Google Inc. et al.

[27] See OWL (Markus Krötzsch et al. 2012) and RDFS (Dan Brickley, Ramanathan Guha 2004)

[28] Markus Krötzsch et al. 2014

This can lead to machines first exchanging their ontologies and then the actual data. Currently most APIs have only human-readable documentation and programmers have to carefully explain computers how to access and process those APIs. With a shared ontology computers can learn how to do that on their own. Also this allows for auto generated code and documentation, further reducing areas of inconsistency. An interesting Semantic Web approach here is Hydra[29].

3 Roles of the Future Web

3.1 Web as a Platform

The Web is advancing as the most important and widespread application platform. There are several operating systems which are built to be web-first, like FirefoxOS[30], ChromeOS[31] or WebOS[32]. Currently they do not have a big market share, but its increasing[33].

Even Operating Systems that have their own native platform are making big efforts to support the web as a secondary platform. iOS and Android are already there, Microsoft hopped onto this trend with the release of Windows 8[34] and Windows Phone.

This trend does not stop at classical computing devices: Smart TVs, Cars and even fridges start to support the web platform. It is hard to find systems which don't. The web seems to establish itself as the ubiquitous platform, making JavaScript the most universal, cross-platform language.

JavaScript as a Compile Target.

JavaScript is increasingly used as a compile target of different programming languages. The performance of JavaScript had been dramatically improved in the last years. Mozilla reached a major breakthrough with asm.js[35], which is "an extraordinary optimizable, low-level subset of JavaScript"[36] that supports performance optimization through source code annotations. There are more projects that aim for making JavaScript a better compile target, including parts of ES Harmony.

Those efforts allow for high performance ports of foreign applications to the web browser. It is getting more common to write web applications in a language of choice (like Google Dart[37]), and compile it to optimized JavaScript afterwards.

---

[29] Lanthaler 2013
[30] https://www.mozilla.org/de/firefox/os/
[31] http://www.chromium.org/chromium-os
[32] http://www.openwebosproject.org/
[33] http://www.gartner.com/newsroom/id/2819917
[34] http://msdn.microsoft.com/en-us//library/windows/apps/br211385.aspx
[35] David Herman et al. 2014
[36] http://asmjs.org/
[37] https://www.dartlang.org/

An impressive example of a port of already existing native code is Linux running in the browser[38] or the Unreal Engine 4 ported to HTML5[39].

This trend is establishing JavaScript the default, cross-platform "virtual machine byte code". Ironically JavaScript succeeds, where JAVA once failed.

3.2 Web of Applications

The current web is dominated by Content Management Systems (CMS). All the widespread CMS are based on the traditional, stateless request-response model. They usually do provide not much abstraction: Either you get the whole site containing all information or you get nothing. This is very inefficient, since that whole site has to be calculated, sent and rendered every time the user moves to a new page.

Now that the architectural possibilities have dramatically improved, this can be done much more efficient and performant for both the server and client.

Single Page Applications (SPA) are much smarter about this, but currently they are almost exclusively applications. Now that Google has better support for indexing dynamic, AJAX oriented websites[40], the biggest hindrance from switching to a more modern architecture is gone. Isomorphic JavaScript[41], sharing the same codebase between client and server, may be a good alternative approach for solving the SEO problem. The server could always deliver the full page on a direct request, and if the client supports it, dynamically load and render all future request asynchronally.

Since the advantages of this approach are obvious, it is likely we'll see some new type of CMS systems emerge which feel more like applications than traditional websites. They will likely be API / Service centric and have a loosely coupled architecture. The front-end could be multiple, completely separate projects – supporting browsers, (native) mobile apps and desktop applications. Service / API centric CMSs can be useful in the context of Internet of Things, since they would have excellent machine accessibility by default through its API architecture.

If this approach should establish itself, the web experience will be more like that of application and apps. Users will become accustomed to a uninterrupted, nearly immediate browsing experience.

3.3 Web of Services

The Web of Services[42] is a natural trend that comes out of the growing field of application of the Web. It acknowledges the fact that the web is not only the traditional browsing of websites anymore. Behind the curtain a lot of machine-to-machine com-

[38] http://bellard.org/jslinux/
[39] https://www.unrealengine.com/html5/
[40] Google Webmaster Central Blog 2014
[41] Spike Brehm 2013
[42] http://www.w3.org/standards/webofservices

munication happens to make the web a smarter and more interconnected experience. The growth of (RESTful) APIs over the last years is tremendous[43].

Internet of Things.

The Internet of Things is a topic on its own, so this article will focus on how web-development might play an important part in the IoT and vice versa.

IoT devices are always connected to the internet, building a bridge between the physical world and the internet. It is important to disambiguate between the terms internet and web. The former is the low-level infrastructure and platform the later runs on. There are a lot of new protocols and standards in development that are geared rather toward IoT and not Web use.

Many of the here mentioned technologies may play an important part in the IoT. The Semantic Web already has put a lot of research and development into machine-to-machine communication which is an essential part of making the IoT happen. Since the web as a platform has become nearly ubiquitous it could become the (human-facing) platform of the IoT as well. In order that the IoT can benefit from the collected information and intelligence that is already existing in the web, a Web of Services becomes even more important.

Since the IoT and the web share the same platform it might be possible that we see an even greater fusion of both, IoT influencing the web and vice versa. Maybe people in the future even won't make a big distinction here anymore.

4 Outlook

We're heading toward a ubiquitous web[44] that escapes the browser window and enters devices and services of all kind. The technological foundations to make this happen have already been laid or are in the process of doing so. With the ever increasing importance of the web, the pace of future development is likely to increase. For sure it's an exciting time to be a web-developer these days.

[43] http://www.programmableweb.com/news/9000-apis-mobile-gets-serious/2013/04/30

[44] Andreas Hotho, Gerd Stumme 2010

Publication bibliography

Andreas Hotho; Gerd Stumme (2010): Towards the Ubiquitous Web.

Axel Rauschmayer (2014): The ECMAScript 6 schedule changes. Available online at http://www.2ality.com/2014/06/es6-schedule.html, updated on 6/6/2014, checked on 11/6/2014.

Crockford, Douglas (2008): The good parts. Working with the shallow grain of JavaScript. Farnham: O'Reilly.

Dan Brickley; Ramanathan Guha (2004): RDF Vocabulary Description Language 1.0: RDF Schema. W3C. Available online at http://www.w3.org/TR/2004/REC-rdf-schema-20040210/, updated on 2004, checked on 10/14/2013.

David Herman; Luke Wagner; Alon Zakai (2014): asm.js. Working Draft — 18 August 2014. Available online at http://asmjs.org/spec/latest/, updated on 8/18/2014, checked on 11/7/2014.

Dimitri Glazkov (2013): Custom Elements. W3C Last Call Working Draft 24 October 2013. Edited by W3C. Available online at http://www.w3.org/TR/2013/WD-custom-elements-20131024/, updated on 10/24/2013, checked on 11/6/2014.

Dimitri Glazkov; Hajime Morrita (2014): HTML Imports. W3C Working Draft 11 March 2014. Edited by W3C. Available online at http://www.w3.org/TR/2014/WD-html-imports-20140311/, updated on 3/11/2014, checked on 11/6/2014.

Dimitri Glazkov; Hayato Ito (2014): Shadow DOM. W3C Working Draft 17 June 2014. Edited by W3C. Available online at http://www.w3.org/TR/2014/WD-shadow-dom-20140617/, updated on 6/17/2014, checked on 11/6/2014.

Dimond, Tom (1957): Devices for reading handwritten characters, pp. 232–237.

Ecma TC39 (2014): ECMAScript Language Specifications, 6th Edition. Draft October 14, 2014.

ECMAScript community (2014): ECMAScript. the language of the web. Available online at http://www.ecmascript.org/, updated on 10/14/2014, checked on 11/5/2014.

Eric Miller; Frank Manola (2004): RDF Primer. W3C. Available online at http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, updated on 2004, checked on 10/14/2013.

Google Inc.; Yahoo Inc.; Microsoft Corporation; Yandex: schema.org. Available online at http://schema.org/, checked on 24.10.201407.11.2014.

Google Webmaster Central Blog (2014): Understanding web pages better. Available online at http://googlewebmastercentral.blogspot.de/2014/05/understanding-web-pages-better.html, updated on 4/23/2014, checked on 11/7/2014.

Ian Hickson; Robin Berjon; Steve Faulkner; Travis Leithead; Erika Doyle Navara; Edward O'Connor; Silvia Pfeiffer (2014): HTML5. W3C Recommendation 28 Octo-

ber 2014. Edited by W3C. Available online at http://www.w3.org/TR/html5/, updated on 10/24/2014, checked on 11/7/2014.

Lanthaler, Markus (2013): Creating 3rd Generation Web APIs with Hydra.

Markus Krötzsch; František Simancík; Ian Horrocks (2014): Description Logics.

Markus Krötzsch; Pascal Hitzler; Bijan Parsia; Peter Patel-Schneider; Sebastian Rudolph (2012): OWL 2 Web Ontology Language Primer (Second Edition). W3C. Available online at http://www.w3.org/TR/2012/REC-owl2-primer-20121211/, updated on 2012, checked on 10/14/2013.

Spike Brehm (2013): Isomorphic JavaScript: The Future of Web Apps. Available online at http://nerds.airbnb.com/isomorphic-javascript-future-web-apps/, updated on 11/11/2013, checked on 11/10/2014.

Tim Berners-Lee (2007): Semantic Web, and Other Technologies to Watch. Available online at http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24), updated on 3/3/2007, checked on 10/24/2014.

Tim Berners-Lee; James Hendler; Ora Lassila (2001): The semantic web. In *Scientific american* 284 (5), pp. 28–37, checked on 10/18/2013.

TJ VanToll (2014): Why Web Components Aren't Ready for Production… Yet -. Telerik. Available online at http://developer.telerik.com/featured/web-components-arent-ready-production-yet/, updated on 7/17/2014, checked on 11/11/2014.