Web Components the future of web development

BTH, (October, 2016)
Jesper Johnsson, jesper.johnsson1995@hotmail.com
Tim Christiansson-Boquist, timc.boquist@hotmail.com

Abstract

In this document we are introducing Web Components and its standards and how it can be used. We are going to show these new features so the reader get a better understanding of this new way of working with the web. We've tried to define our research questions in such a way that'll give the readers a more basic approach to the Web Component world, both in its current and future state. We hope that this sheds some light on the subject and hopefully expands the user base.

Contents

1. Introduction

The web development environment is constantly being updated, with technologies and standards being redefined and changed quite often. We think that the next major change to it is Web Components. Being able to reuse your code is something really basic in the software development world. We've seen it in some languages as libraries, some as modules and some components. The web hasn't had a native way of doing this, it has only been frameworks that has been offering this feature, until a couple of years ago with the introduction of the Web Component standards. This collection of standards offers completely new features and offers a new way of development for the web. The future of the web is already here, in the form of Web Components.

2. Research questions

1. What defines and characterizes a Web Component?
2. Which major browsers is currently supporting Web Components?
3. Why are Web Components needed in web development?
4. How do you create a Web Component?

3. Method

We'll start by gathering information about Web Components and the four standards that are a part of the standard Web Component. Using this information we'll be trying to explain these standards as short and concise as possible, expecting that the readers have some experience in the field of HTML, CSS and JavaScript.

Using the websites caniuse.com and webcomponents.org, we'll find the current support for the standards in five of the most popular browsers, to see how far the support of Web Components currently are.

We'll be finding pros and cons with the Web Components API and give our opinion based on these facts. We will also present some similarities between the Web Components API and the services some of the frameworks are offering today.

We'll show how you could create a Web Component based on the latest semantics, syntax, clarity and structure of the component itself.

4. Literature review

Search criteria

Because this document is an introduction to Web Components our search criteria was broad in terms of just gathering information about the parts of Web Components.
Our search terms consisted:

- HTML Imports
- Custom Elements
- Shadow DOM
- HTML Template
- Web Component

Web Components

All these links are introductions to Web Components to get a broad view over the inner workings of the features it provides. Web Components has its own website which provided a lot of good information. The wikipedia page gave us an base of information and some external references that we could continue our research with.

www.webcomponents.org/
https://en.wikipedia.org/wiki/Web_Components

Developer sites

These links are more in depth over the features that Web Components provides. In these links we started using more of our search terms to get the information we needed. This gave more information about HTML Imports, Templates, Custom Elements and Shadow DOM. These sources are provided by the developers of the different browsers, so we consider them very reliable.

https://developer.mozilla.org/
https://developers.google.com/
http://w3c.github.io/
https://www.w3.org/

Browser status

Here are links that provided us with status of Web Components for the different browsers. Caniuse.com has been very helpfull in this regard and webcomponents.org also provided usefull information and links to the browsers own feature page.

http://caniuse.com/#search=web%20components
http://webcomponents.org/

5. Analysis and discussion

5.1 What defines and characterizes a Web Component?

The reason for us having this question is to introduce the Web Component standards in a fast and basic way. We are excluding most of the specific details and we'll explain the standards in a way that makes it easy to understand, what the they do and what they bring to the Web Component standard as a whole. We believe this part is crucial as the other parts relies on the reader having the basic understanding of the parts of a Web Component.

5.2 Which major browsers is currently supporting Web Components?

We want to present the current status of the Web Component standard in five of the major browsers [29]. Those browsers being chrome, opera, firefox, safari and IE / Edge. Some of the browsers support more than others, chrome and opera are the ones with the best support by supporting all standards related to Web Components. Firefox and safari have postponed all further development related to HTML imports as they seem to think that the new ES6 modules standards will have some sort of better importing functionality. Or as the webkit status pages says, "Multiple browser vendors have raised concerns about how the HTML imports currently works. For example, its dependency model is not fully compatible with ES6 Modules. We are exploring other ways to package Web Components by integrating into ES6 Modules with other browser vendors" [1]. The standards related to Web Components are either planned to be implemented or in development for the browsers. Browsers accepting and implementing completely new standards have often proven to take quite some time and we believe that the Web Component standard won't be implemented fully in all the browsers for another couple of years. This due to that they are in disagreement about some parts mentioned above.

5.3 Why are Web Components needed in web development?

The Web Component standards offers features to HTML, CSS or JavaScript. Such as the possibility to encapsulate css with Shadow DOM, enable templating with HTML template, import other HTML documents with HTML Imports and write your own HTML elements that the browsers can interpret. [2] Web Component standards offers features that were only available as workarounds or part of a larger framework or module. One being the encapsulation of css.

Because of css's global nature it's impossible to have 100% encapsulated code natively. The only solution to get a somewhat encapsulated css before was to have some sort of standard when you wrote your own css, like prefixing your classes. So when you're using another developers code or finalized code of some sort, you had to be sure that their code wasn't affecting your code in any way. The Web Component makes this work exceedingly

well with the help of Shadow DOM and the way it hides parts of the DOM making it encapsulated [3][4].

The most valuable feature that the new Web Components standards brings to the web development is the promise of interoperability. The intention for the written Web Component is to work wherever the developer see fit to use them. This means that you as an developer can develop one Web Component that's reusable across all technology stacks and browsers.
There's some constraints to this though, for the Web Component to work within frameworks and browsers they'll need to be compliant with the Web Component standards. Fortunately for us developers, there's workaround for all the major frameworks already written and working [2][5]. There's polyfills for the browsers that hasn't adopted the standard [6].

5.4 How do you create a Web Component?

There's many ways of presenting the creation of a Web Component. We chose a simple approach with a clear structure both in the code and files, as to give the reader the latest and greatest of Web Component creation. The example presented is a simple "Hello World" example as we just wanted to introduce the creation of an Web Component. We chose to write the JavaScript with ES6 in mind, specially the ES6 class reference. This gives the best visual representation of the code and makes it easier to understand and interpret the javascript code.
We felt the need for this last question since it gives the most for those wanting to learn about how to write a Web Component [28].

6. Results

6.1 What defines and characterizes a Web Component?

A Web Component consists of four separate technologies, and is itself only an expression to describe this new way of creating reusable user interface widgets for the web using open web technology. At the time of writing the set of features, that make up a Web Component, is currently being added and updated to the HTML and DOM specifications by the W3C (World Wide Web Consortium). This means that the specifications can change at a future point.

The four technologies that characterize a Web Component are the following.

- Custom Elements
- Shadow DOM
- HTML imports
- HTML templates

These four technologies are what defines a Web Component, but all of the above can be used independently for other purposes than creating a Web Component. These four technologies are part of a larger effort to improve the current platform for HTML [2][7].

6.1.1 Custom Elements

Custom element makes it possible for authors to build their own fully-featured HTML tags and elements. The modern browsers of today mostly supports the use of non-standard elements, but such elements have been hard to use or not very functional. Instead of using the non-standard elements, one can define a custom element and inform the browser's parser how that element should be interpreted, constructed and reacting to changes.

There's two types of Custom Elements currently, autonomous Custom Elements and customized built in elements. Autonomous Custom Elements are the ones that's completely new elements without any connection to an natively available HTML element, extending the interface `HTMLElement`. Whereas customized built in elements are extending a native HTML element such as a button, extending the interface `HTMLElementButton` [8][9].

6.1.2 Shadow DOM

The shadow DOM removes the global nature of CSS. It provides encapsulation and separates the DOM from the main document. Therefore preventing it from leaking into the global scope and creating problems relating to naming conventions in css, id's colliding in the HTML markup and more. Without tools or naming conventions, you can bundle CSS with markup and hide implementation details which makes it hidden in the DOM tree [10][11][12].

6.1.3 HTML Imports

HTML imports provides a way to import a HTML document into a HTML document. Using the same tag as when linking to an external stylesheet, the link tag. It's now possible to use the flag `rel="import"` to import HTML files into a document. It will import everything that works natively inside the HTML files, so any styles, scripts or alike will work correctly. Making it the ideal partner for the Web Component, since you can package your Web Component inside one HTML file and import it [13][14].

6.1.4 HTML Templates

The template element is used to declare inert DOM subtrees, fragments of HTML markup, which are unused by the document when loaded but are parsed as HTML. When rendering, the template element represents nothing since its content are stored in a document fragment correlated with another document. To access the templates content one must access the attribute, "content". This content can be manipulated and instantiated elsewhere with identical contents and are available at runtime for use by the web page. This prevents the templates content to interfere with the main document and avoids scripts from executing and forms from being submitted [15][16][17].

6.2 Which major browsers is currently supporting Web Components?

6.2.1 Chrome

As of chrome version 36, chrome is fully supporting the Web Component standards natively without the usage of any kind of polyfill. They are also keeping the standards up to date. The chrome team and google has been the biggest advocates when it comes to supporting Web Components. Google is also behind the popular polyfill Polymer, that is used to get all these functions and compatibility for other browsers [18][19].

6.2.2 Opera

As of opera version 40, opera is fully supporting the Web Component standards natively without the usage of any kind of polyfill. They are also keeping the standards up to date [18][19].

6.2.3 Firefox

As of firefox version 22, firefox supports the HTML template standard. The HTML imports standard isn't planned to be supported by firefox as of late 2014 [20]. The Custom Elements and shadow DOM standards are currently in development. Firefox partially supports the other standards of Web Components but as development flags that needs to be enabled by the user for them to work. These features can't be enabled by a developer, so for a developer to access the later parts of Web Components a "polyfill" is required to support it for the browser [18][19].

6.2.4 Safari

As of safari version 10, safari supports two parts of the Web Component standards natively. These two standards are HTML templates which was introduced in safari version 7.1 and Shadow DOM which was introduced in safari version 10. The Custom Elements standard is currently in development as of late 2015. [21] The HTML imports standard is currently not planned to be supported by safari [18][19].

6.2.5 IE/Edge

IE and Edge only supports HTML templates. The other parts of Web Components is currently under construction or planned for future development [18][19].

6.3 Why are Web Components needed in web development?

Developers today are always looking for efficiency, quality and reusability when developing. The reuse of code between various projects exemplifies efficiency and when done right improves the quality of life for the developers, improves the quality of the code developed and results in less resources being spent overall. Alan W. Brown writes in his book, “Large-Scale, Component-Based Development” from 2000, “Many people in the software industry are beginning to see Component Based Development as an exciting new approach to application development which offers the promise of reducing cycle time for software development, and improving the quality of delivered applications” [22]. Even though it was written over 15 years ago this quote is still applicable to this day.

The idea of componentization and modularization is nothing new in the web development world, it’s something that’s existed for a long time, but in the form of frameworks introducing the idea and implementing it. Some examples are jQuery plugins [23], AngularJS directives [24] and React components [25]. But there hasn’t been something standardized that works across all of these technologies and platforms. That’s where the Web Component standards comes into the picture. Web Components are interoperable, meaning that they can transcend frameworks and be used in multiple projects of different technology stacks. This interoperability decreases the need to re-write the same feature all over again just to be able to use them in a newer or another technology stack, resulting in code with longer lifespan, higher quality and easier maintainability [2][5].

Writing reusable widgets for the web has always required some sort of framework and even then it will often just be possible to reuse these widgets within that framework. The Web Component standard introduces the idea of self encapsulated widgets that will be reusable across projects, frameworks, browsers and platforms [2]. Web Components may not be there yet, but with a devoted community and development ongoing it may just be a matter of time before we have a fully implemented Web Component API that works everywhere. An API that fills the void that currently exists, not being able to write reusable components for the web natively.

Web Components are needed given all the benefits it brings to the web development world. Sure, it's not perfectly implemented right now, you as a developer will have to use polyfills and workarounds to get the interoperability. But imagine when you can write a single Web Component that works everywhere, a Web Component transcending frameworks and browsers.

6.4 How do you create a Web Component?

To show how to create a Web Component we've written a simple one and we will explain the code and some of the semantics that makes it all work. We will be writing our JavaScript with ES6 in mind and specifically the ES6 class definition, we think that it brings clarity to the code and makes it easier to understand and read the Web Component code compared to its ES5 counterpart [27].

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="import" href="hello-world.html">
    <title>Title</title>
</head>
<body>
    <hello-world data-text="World"></hello-world>
</body>
</html>
```

The above is a simple index file with nothing much to it. The only part that stands out is the `hello-world` custom element located inside the `body` tag. This element is our Web Component and to make it work we've had to import it using the HTML imports functionality of the `link` tag. By using the flag `rel="import"` we've specified that we would like to import the file `href="hello-world.html"` by simple referencing it like we would a stylesheet.
Our custom element has an attribute `data-text` that we use to pass data to the component itself. This is similar to how you send data into other elements in HTML.

hello-world.html

```
<script src="hello-world.js"></script>
<template>
    <style>
        strong {
            color: blue;
        }
    </style>

    <p>Hello <strong></strong> :)</p>
</template>
```

The hello-world.html file is our main templating file for our Web Component, it's here that we've defined the structure of our component. This basic example has a `script` tag that will load the JavaScript file connected to this component. The `template` tag is what contains the

markup and styling of our component, since the tag is inert no code inside the tag will be parsed by the browser until we've loaded it with JavaScript.

hello-world.js

```
class HelloWorld extends HTMLElement {
    detachedCallback() {};

    attributeChangedCallback(attr, oldVal, newVal) {};

    attachedCallback() {
        var template = this.owner.querySelector('template');
        var clone = document.importNode(template.content, true);
        this.root = this.createShadowRoot();
        this.root.appendChild(clone);
        this.strong = this.root.querySelector('strong');
        this.strong.textContent = this.getAttribute('data-text');
    }

    createdCallback() {};
}
if(document.createElement('hello-world').constructor !== HelloWorld) {
    HelloWorld.prototype.owner = (document._currentScript ||
document.currentScript).ownerDocument;
    document.registerElement('hello-world', HelloWorld);
}
```

The javascript file, hello-world.js, has two major responsibilities. One being the definition of our custom element and defining the functionality of our element. In our case we're extending the interface `HTMLElement` which makes it possible for the browser to parse our custom element. This interface gives us access to the four lifecycle callbacks otherwise not accessible.
The only lifecycle callback we're using is the `attachedCallback`, that is called whenever an instance was inserted into the document.

In the callback function we get the template from the hello-world.html. Then we create a clone of its markup by importing the templates content, `document.importNode(template.content, true)`.
We create a shadow root that the clone get attached to. Then we do some simple DOM manipulation to add the value of the attribute to the strong tag located inside the shadow root.
At last in the file we check if an instance of the custom element is already created or not and if not it gets registered to the document by using `document.registerElement('hello-world', HelloWorld)`[26][28].

output

```
Hello World :)
```

This is the result of all the code above.

7. Conclusion

Our modern web development world requires a way of creating reusable components that can transcend framework and browsers. The solution is here and it's Web Components. With this report we wanted to give you the reader a quick introduction to the world of Web Components and it's parts for you to better understand the separate techniques that makes it possible to write Web Components. The Web Component standard consists of four standards and them being Custom Elements, HTML templates, HTML imports and Shadow DOM. We think that all of these parts are interesting and brings equally important features to the table, but it is together they work at their best.

The current native support in modern browsers are not the greatest. But with the help of polyfills you can add Web Components to your developing cycle today. For you the reader to get a better understanding of how a Web Component actually works we wanted to show a simple example of the classic hello world example. We wanted to show how the different parts works together and what they all contribute. The benefits of Web Components exceed the initial cost of learning and integration in your development cycle, it can even be applied to existing projects. Web Components will change web development as we know it, if it already hasn't done just that.

8. Future work

We would like to explore the possibility of creating an application solely based on Web Components. Since we like the concept of Web Components and the features included, this would be both a proof of concept and a way for us to learn more about developing using Web Components and really explore the pros and cons of Web Components.

Another possibility is that we would research and take a more in depth look on one or more of the standards of the Web Components API. It would probably be either HTML imports or Shadow DOM since they, in our opinion, bring the most interesting features to the table.

Another field that interest us are the interoperability of the Web Components API. Is it possible to use the Web Components API with other frameworks. How fluent is the use, does it require work arounds or does everything work right out of the box. This would be a comparison of frameworks, to see how well the Web Components API really works.

Do a comparison of the most popular Web Components frameworks. See what differs between them and compare the features of the different frameworks. The frameworks in

mind are Polymer, X-tag and Bosonic, that currently are the three most popular Web Components frameworks out there.

We would like to try changing an already existing application to use Web Components. This would tackle the idea of replacing existing code with Web Components. Here it’s about finding out how easy of a transition it is and if you should do it.

9. References

[1] Current status of HTML imports
https://webkit.org/status/#feature-html-imports
(Accessed 2016-10-20)

[2] English wikipedia page for Web Components
https://en.wikipedia.org/wiki/Web_Components
(Accessed 2016-10-10)

[3] Shadow DOM encapsulates CSS
https://www.html5rocks.com/en/tutorials/webcomponents/shadowdom-201/
(Accessed 2016-10-10)

[4] English wikipedia page for cascading style sheets (CSS)
https://en.wikipedia.org/wiki/Cascading_Style_Sheets
(Accessed 2016-10-20)

[5] Why invest in Web Components?
http://webcomponents.org/articles/interview-with-michael-bleigh/
(Accessed 2016-10-10)

[6] The main page of webcomponents.org, [POLYFILL] box, the polyfill webcomponentjs
http://webcomponents.org
(Accessed 2016-10-10)

[7] Mozilla Developer Network page for Web Components
https://developer.mozilla.org/en-US/docs/Web/Web_Components
(Accessed 2016-10-10)

[8] The w3c working draft of Custom Elements
https://www.w3.org/TR/custom-elements/
(Accessed 2016-10-10)

[9] Mozilla Developer Network page for Custom Elements
https://developer.mozilla.org/en-US/docs/Web/Web_Components/Custom_Elements
(Accessed 2016-10-10)

[10] The w3c working draft of Shadow DOM
https://www.w3.org/TR/shadow-dom/
(Accessed 2016-10-10)

[11] Shadow DOM v1: Self-Contained Web Components
https://developers.google.com/web/fundamentals/getting-started/primers/shadowdom

(Accessed 2016-10-10)
[12] Mozilla Developer Network page for Shadow DOM
https://developer.mozilla.org/en-US/docs/Web/Web_Components/Shadow_DOM
(Accessed 2016-10-10)

[13] The w3c working draft of HTML Imports
https://www.w3.org/TR/html-imports/
(Accessed 2016-10-10)

[14] Mozilla Developer Network page for HTML Imports
https://developer.mozilla.org/en-US/docs/Web/Web_Components/HTML_Imports
(Accessed 2016-10-10)

[15] Mozilla Developer Network page for HTML Template
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template
(Accessed 2016-10-10)

[16] HTML's new template tag
https://www.html5rocks.com/en/tutorials/webcomponents/template/
(Accessed 2016-10-10)

[17] HTML specification of the template tag
https://html.spec.whatwg.org/multipage/scripting.html#the-template-element
(Accessed 2016-10-10)

[18] The main page of webcomponents.org, Browser support
www.webcomponents.org/
(Accessed 2016-10-19)

[19] Browser support
http://caniuse.com/#search=web%20components
(Accessed 2016-10-19)

[20] Mozilla browser support
https://hacks.mozilla.org/2014/12/mozilla-and-web-components/
(Accessed 2016-10-19)

[21] Safari browser support, Custom Elements
https://bugs.webkit.org/show_bug.cgi?id=150225
(Accessed 2016-10-19)

[22] Large-Scale, Component-Based Development
Alan W. Brown
ISBN: 0-13-088720-X
http://armstrong.craig.free.fr/eBooks/Prentice%20Hall/Prentice%20Hall%20Large-Scale%20Component-Based%20Development.pdf

[23] jQuery plugins
https://learn.jquery.com/plugins/
(Accessed 2016-10-20)

[24] Angularjs Directives
https://docs.angularjs.org/guide/directive
(Accessed 2016-10-20)

[25] React Components
https://facebook.github.io/react/docs/react-component.html
(Accessed 2016-10-20)

[26] Example of Custom Elements definition, by Mozilla Developer Networks
https://developer.mozilla.org/en-US/docs/Web/Web_Components/Custom_Elements/Custom_Elements_with_Classes
(Accessed 2016-10-18)

[27] ES6 standard for Web Components
http://www.benfarrell.com/2015/10/26/es6-web-components-part-3-making-an-es6-component-class/
(Accessed 2016-10-18)

[28] GitHub hello world code, old standard
https://github.com/webcomponents/hello-world-element
(Accessed 2016-10-18)

[29] The most popular browsers
http://www.w3schools.com/browsers/
(Accessed 2016-10-25)