SmartComposition: Enhanced Web Components for a Better Future of Web Development

Michael Krug
Department of Computer Science
Technische Universität Chemnitz
Chemnitz, Germany
michael.krug@informatik.tu-chemnitz.de

Martin Gaedke
Department of Computer Science
Technische Universität Chemnitz
Chemnitz, Germany
martin.gaedke@informatik.tu-chemnitz.de

ABSTRACT
In this paper, we introduce the usage of enhanced Web Components to create web applications with multi-device capabilities by composition. By using the latest developments of the family of W3C standards called "Web Components" that we extent with dedicated communication and synchronization functionality, web developers are enabled to create web applications with ease. We enhance Web Components with an event-based communication channel, which is not limited to a single browser window. With our approach, applications using the extended SmartComponents and an additional synchronization service also support multi-device scenarios. In contrast to other widget-based approaches (W3C Widgets, OpenSocial containers), the usage of SmartComponents does not require a dedicated platform, like Apache Rave. SmartComponents are based on standard web technologies, are natively supported by recent web browsers and loosely coupled using our extension. This ensures a high level of reuse. We show how SmartComponents are structured, can be created and used. Furthermore, we explain how the communication aspect is integrated and multi-device communication is achieved. Finally, we describe our demonstration by outlining two example applications.

Categories and Subject Descriptors
D.2.11 **[Software Engineering]**: Software Architectures. D.2.13 **[Software Engineering]**: Reusable Software. C.2.4 **[Computer-Communication Networks]**: Distributed Systems - *Client/server, distributed applications.*

Keywords
Web Components, HTML5, multi-platform web applications, distributed multi-device web applications, web application development, mashup, composition, reusable components.

1. INTRODUCTION
Web application development lacks simple reuse of often used HTML/JS constructs. In the last years, with the rapid advancement of JavaScript, many widget-like components were provided as JS libraries or snippets that can be applied to various standard HTML elements. Those standard HTML elements (mostly DIV elements) are used as a container to host a lot of dynamically created HTML elements. Common examples would be image slideshows/lightboxes, map sections or AJAX forms. We observed several problems with those approaches: first of all, those components cannot be used directly in the HTML code. You have to add a placeholder element and apply some JavaScript code on it. Secondly, they often have dependencies and are complicated to configure. Configuration in the markup code is often not possible and you have to use a JSON object within a function call as initial configuration. Furthermore, the components are created in the same DOM (Document Object Model) tree as the document. This can lead to a lot of conflicts if there are duplicated IDs or CSS rules that accidentally apply to child elements. In summary all these aspects and issues mean that developing state-of-the-art web applications becomes a difficult and error prone task especially for inexperienced developers. A use case, which most of those JS components do not tackle, is a mashup-like scenario, where composing multiple components that work together creates the desired application. Those mashup scenarios require a way for the components to exchange pieces of data. For other widget approaches, like W3C Widgets[1] or OpenSocial containers[2], there are developments that provide inter-widget communication [4]. The integration of OpenAjax Hub[3] into Apache Rave[4] is such an approach. Unfortunately, these widget types need to be deployed and hosted in platforms like Apache Rave or Apache Shindig[5] and cannot be used in standard HTML websites or applications. To an increasing degree, users are using more than one device on regular basis. Not only single users are working with multiple devices, also multiple users (teams) are working on collaborative tasks using more and more devices in parallel. Therefore, web applications should support the combination of multiple – especially mobile – devices into one consistent user experience. We call these applications distributed web applications. They are defined as applications that can be used on multiple devices and that work on a shared context at the same time. There are existing developments regarding distributing user interfaces like the DireWolf framework [1], which extends Apache Shindig and OpenSocial widgets. Unfortunately, as stated before, they are limited to be used in special portal environments.

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2742832

[1] http://www.w3.org/TR/widgets/

[2] http://opensocial.atlassian.net/wiki/display/OSD/Specs

[3] http://www.openajax.org/member/wiki/OpenAjax_Hub_2.0_Specification

[4] http://rave.apache.org/

[5] http://shindig.apache.org/

Our approach aims to support developers in creating distributed web applications by the composition of predefined platform-independent components that are able to communicate in one application context as well as across multiple devices. Therefore, we propose to use the new set of W3C standards called Web Components as the new basic technique for creating widget-like components. Additionally we provide an extended Web Component prototype with new communication features as well as an optional synchronization service that seamlessly integrates into the application and provides data synchronization between multiple devices.

2. SMARTCOMPOSITION APPROACH

Our approach is based on the idea of creating web applications by composition. In [3] we proposed a component-based architecture for multi-device web applications. We use the presented ideas to create a prototypical implementation by exploiting the Web Components technologies. Our implementation uses only client-side JavaScript. The major benefit of using the proposed technologies for creating modern widgets is that no runtime environment or portal software is needed to host such composed applications. This could have a large impact since developers can also easily integrate such components in common content management systems like WordPress, Drupal or Joomla, as well as in any other HTML5 based website.

Web Components consist of four new W3C web standards:

- Templates[6], which define chunks of markup that are inert but can be activated for use later.
- Custom Elements[7], which let authors define their own elements, with new tag names and new script interfaces.
- Shadow DOM[8], which encapsulates a DOM subtree for more reliable composition of user interface elements.
- HTML Imports[9], which define how templates and custom elements are packaged and loaded as a resource.

New components can be easily integrated in a website by using the new HTML5 import statement, which uses the <link> tag to load external definition files (see Listing 1). The structure and content of the definition file is stated later. After the import of the component resource file you can instantly use the custom element tag in your HTML markup. Web Components are registered as new HTML elements. Thus, you can use them in the same ways as other standard elements. Configuration is possible through attributes or child elements. In our SmartComposition approach, a SmartComponent is a mixture of W3C Web Components technologies and extensions for a systematic composition transcending device and platform boundaries. There are the following three parts defining a SmartComponent (cf. Listing 2): Firstly, the template, which specifies the basic look and HTML structure of the new component. The template element contains markup that is defined for later usage. The content of the template element is parsed by the parser, but it is inactive and not rendered. The content of the template can be accessed as a document fragment through its "content" property. Using a <content> node in the template definition, you can set up an insertion point for child elements of the component markup that otherwise would be hidden by the shadow DOM. This is used by the later shown image slideshow example. The content of the component itself is inserted into a shadow DOM. The shadow DOM is an adjunct tree of DOM nodes. This subtree can be associated with an element, but does not appear as a child node of the element. Instead the subtree forms its own scope. Due to the different scope of the shadow DOM, the styles, names or IDs of elements in the root document do not interfere with the definitions in the component. Secondly, there is an optional style section, where you can define the look of the elements of the component. The CSS @import statement is supported to reuse existing style sheet definitions. To address the custom element that is hosting the component's content, a new pseudo-class called ":host" is provided.

```
<html>
 <head>
  <link rel="import" href="component.html">
 </head>
<body>
 <smart-component some-attr="some-value">
  <shadow-root> /* Not visible as child */
   <h1>Some Widget</h1>
   <p></p>
  </shadow-root>
 </smart-component>
</body>
</html>
```

Listing 1: Usage of Web Components in HTML5 websites

```
/* Template definition */
<template>
  <h1>Some Widget</h1>
  <p></p>
</template>
/* Style rules */
<style type="text/css">
  h1 { … }
</style>
/* Web Components Extension */
<script src="Helper.js"></script>
/* Optional for multi-device sync * /
<script src="SynchronizationService.js">
</script>
<script>
(function() {
  var smartComponent =
      createWebComponent('smart-component');
  smartComponent.created = function() { … };
  smartComponent.attached = function() { … };
  smartComponent.detached = function() { … };
  smartComponent.handleMessage = function() {
    /* Gets called when message on subscribed
       topic was published */
  };
}());
</script>
```

Listing 2: Definition file of an enhanced Web Component

The third part covers the dependencies and actual program code. New components are defined using a helper method that we created to automate all necessary steps to create a custom element and setting up event bindings. Custom elements are new types of DOM elements that can be defined by authors, are stateful DOM objects and provide script interfaces. We are using prototype inheritance to extend the *HTMLElement* with additional functionality. After the call of the method the element with the stated name is registered and can be used in the markup. Loosely coupling of components is very important to ensure

[6] http://w3.org/TR/html5/scripting-1.html#the-template-element

[7] http://w3.org/TR/custom-elements/

[8] http://w3.org/TR/shadow-dom/

[9] http://w3.org/TR/html-imports/

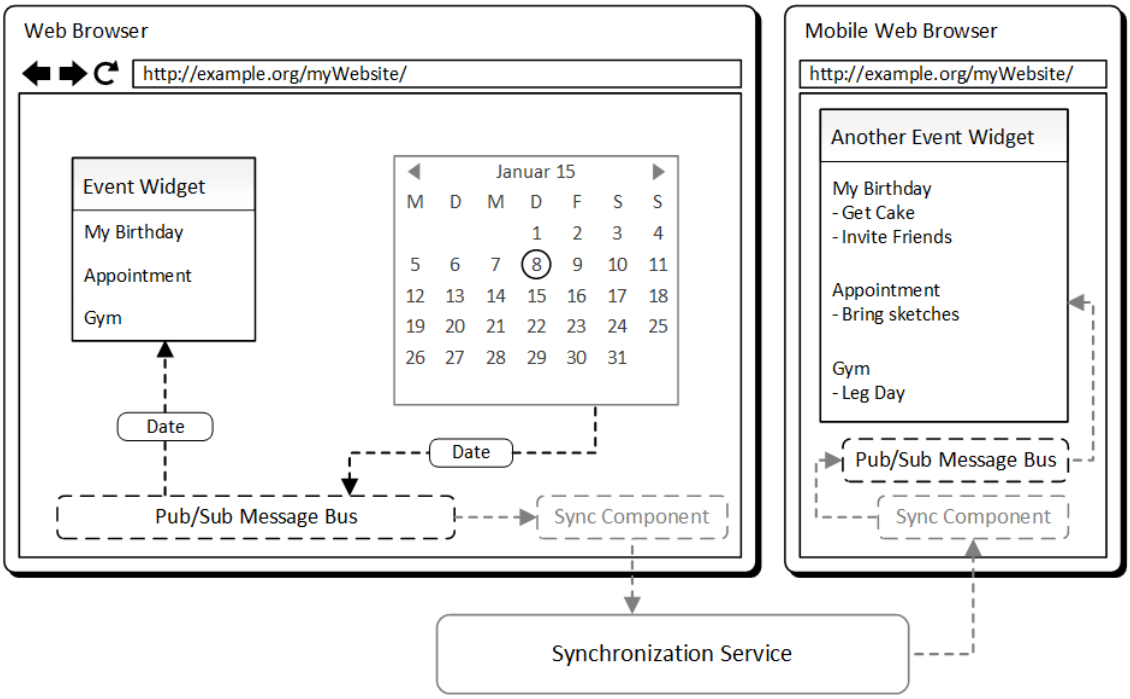

Figure 1: Simplified inter-component and inter-device communication architecture

simple reuse and enable new compositions. Therefore, to enable message exchange between our SmartComponents we propose an event-driven communication channel using a topic-based publish/subscribe mechanism. A simplified overview of the inter-component and inter-device communication architecture of the SmartComposition approach can be seen in Figure 1. Components can subscribe to topics they are interested in and can publish information by posting messages to the event-driven bus. The publish/subscribe message bus is implemented using the HTML5 Web Messaging API. The API provides a function to post a message to a given window. Messages can be structured objects, e.g. nested objects and arrays, can contain JavaScript values (strings, numbers, dates, etc.) and certain data objects. To receive a message the component has to bind an event listener to the "message" event. By using those two methods and by giving the message a predefined structure, containing the topic and the data, we achieve a topic-based and event-driven communication channel, which sets the basis for inter-component communication and enables loosely coupled composition.

Furthermore, to provide developers with the opportunities to create multi-device-enabled web applications, we present a WebSocket-based synchronization service. To ensure a hassle-free reuse of our components, our approach proposes a stand-alone solution with no dependencies and side-effects on other components. The synchronization service consists of two parts. One part on the client side and one part hosted at a web server. The client-side component is implemented as a custom JavaScript object. It captures all events on the previously described publish/subscribe message bus and sends them to the server-side component. When the client-side component receives a message from the server, it sends it back to the local publish/subscribe message bus and the local components will receive the events. We are using the WebSocket protocol[10] for the client-server communication. This provides us with a full-duplex, low-latency communication channel using standard web technologies. The server-side component is a WebSocket server, which analyses the received messages and broadcasts them to groups of connected devices. We call those groups "sessions". The term "connected devices" is defined as: devices using the same synchronization endpoint that share the same session identifier, i.e. context. One major advantage of our synchronization service is that no reconfiguration of existing components is necessary for multi-device communication. Since the service is working like a hook, all messages sent by the components are captured without changing the code or configuration. In the next chapter we present the demonstration of our proposed implementation.

3. DEMONSTRATION

To demonstrate our approach we show how easy it is to create interactive, multi-device web applications with the use of our enhanced Web Components. We show a working prototype of the SmartComposition approach using two demos: The first one covers a common use case. We show an image slideshow that is implemented as a SmartComponent and is able to exchange data with other components. This, the slideshow could be controlled by another SmartComponent. By using our presented synchronization service, this controlling component can also be used on another device. You can even have the same slideshow in

```
<link rel="import" href="slider.html">
<link rel="import" href="control.html">

<smart-slider-component>
  <img id="img1" src="sunset.jpg">
  <img id="img2" src="arch.jpg">
  <img id="img3" src="grooves.jpg">
  <img id="img4" src="rock.jpg">
</smart-slider-component>
<smart-ctrl-component></smart-crtl-component>
```

Listing 3: Slideshow implemented with Web Components

[10] https://tools.ietf.org/html/rfc6455

a synchronized state on a second device. As you can see in Listing 3, the markup of the image slideshow is noticeably clean and readable, without any cluttering from inline JavaScript or any HTML elements that are used for implementing the sliding functionality. You only add the images that should become part of the slideshow. The smart component for control has an even simpler markup.

Online Demonstration – Image Slideshow: http://vsr-demo.informatik.tu-chemnitz.de/smartcomposition/slideshow/

Figure 2: Image slideshow exploiting multi-device capabilities

The second demo is more complex. We used our SmartComposition approach to implement a distributed media enrichment application. This demo shows a mashup-like scenario, which was previously discussed and implemented but without the usage of Web Components in [2]. We created different kinds of new SmartComponents. Most of them gather data from various web services regarding a topic or keyword to display information that can be useful while watching a video. The video is played by a special video component that posts messages containing metadata at specific timestamps. This is done by exploiting the TextTrack-API and an attached VTT subtitles file containing time-based metadata – in this case a transcript of the video. Parts of the transcript are published using a specific topic that is subscribed by a component that extracts keywords using statistical algorithms and natural language processing technologies. The extracted keywords are again published to the bus with different topics based on determined categories. To visualize information about those topics, we implemented components that catch data from, e.g. Twitter, Flickr, Google Maps, Google Images and Wikipedia. The mashup of components is as simple as adding new elements to the HTML document (see Listing 4).

```
<video-component width="550" height="300">
  <source src="video.mp4" type="video/mp4" />
  <track src="metadata.vtt" kind="metadata" />
</video-component>
<maps-component lat="52" lng="12" width="200"
 height="300"></maps-component>
<wiki-component keyword="Europe" width="600"
 height="300"></wiki-component>
<twitter-component keyword="Europe"
width="400" height="300"></twitter-component>
```

Listing 4: Web Components used for media enrichment by visualizing information provided by time-based metadata

To proof the multi-device capabilities of our solution, we show that SmartComponents can display different kinds of information synchronized on multiple devices, and that they can even be moved between devices.

Online Demonstration – Media Enrichment: http://vsr-demo.informatik.tu-chemnitz.de/smartcomposition/

Our demos can be used in any modern web browser without the installation of plugins or server-side runtime environments. If the distributed application requires any multi-device synchronization, a WebSocket server has to be deployed. Unfortunately, not all technologies we are using are yet implemented in all browsers. Most of them are still W3C working drafts. By optionally using the webcomponent.js[11] polyfills, SmartComponents are also enabled in web browsers that lack native support.

4. CONCLUSION

In this paper we demonstrated how to extend the new W3C Web Components to build SmartComponents with event-based communication channels and multi-device data exchange capabilities. We are able to support web developers creating complex distributed web applications with a high level of reuse. The core of our approach, Web Components, can be used in any modern web browser without the installation of additional server-side or client-side software. Since SmartComponents are custom elements that become first-class HTML elements, you can basically add and configure new parts of you web application directly in your HTML markup. The import is done with only one line of code. Inserting the content of Web Components into an adjunct shadow DOM subtree prevents CSS rules and IDs of elements from conflicting. Our extension of adding WebSocket-based communication functionality for loosely coupling of SmartComponents to compose rich web applications experiences across distributed platforms and between multiple devices.

Further research will address how to provide a repository to store and distribute reusable SmartComponents. We are also working on approaches to describe the communication interfaces and topic names to ensure hassle-free composition of SmartComponents.

5. REFERENCES

[1] Dejan Kovachev, Dominik Renzel, Petru Nicolaescu, István Koren, and Ralf Klamma. 2014. Direwolf: a framework for widget-based distributed user interfaces. *J. Web Eng.* 13, 3-4 (July 2014), 203-222.

[2] Michael Krug, Fabian Wiedemann, and Martin Gaedke. 2014. Enhancing media enrichment by semantic extraction. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion* (WWW Companion '14). International WWW Conferences Steering Committee, 111-114. DOI=10.1145/2567948.

[3] Michael Krug, Fabian Wiedemann, and Martin Gaedke. 2014. SmartComposition: A Component-Based Approach for Creating Multi-screen Mashups. In *Web Engineering - Lecture Notes in Computer Science* (ICWE'14), Springer International Publishing, 236-253. DOI=10.1007/978-3-319-08245-5_14.

[4] Scott Wilson, Florian Daniel, Uwe Jugel, and Stefano Soi. 2011. Orchestrated User Interface Mashups Using W3C Widgets. In *Proceedings of the 11th international conference on Current Trends in Web Engineering* (ICWE'11), Springer-Verlag, Berlin, Heidelberg, 49-61. DOI=10.1007/978-3-642-27997-3_5.

[11] http://webcomponents.org/polyfills/