SmartComposition: Extending Web Applications to Multi-screen Mashups

Michael Krug(✉), Fabian Wiedemann, and Martin Gaedke

Technische Universität Chemnitz, Chemnitz, Germany
{michael.krug,fabian.wiedemann,martin.gaedke}@informatik.tu-chemnitz.de

Abstract. The overall objective of UI mashups is to enable non-experts to create rich web applications. While current approaches focus on creating UI mashups running on a single screen, we propose SmartCompositionto enable local developers to create multi-screen mashups. We present our enhanced SmartComponents, which are based on the latest developments of the family of W3C standards called "Web Components", as part of our SmartCompositionapproach. SmartComponents provide loosely coupling and support both single- and multi-device usage scenarios by extending Web Components with dedicated communication and synchronization features. We support multiple types of SmartComponents, not limiting them to user interface components. In contrast to other approaches, SmartComponents are independent, encapsulated, configurable and programmable, which ensures hassle-free reuse in any HTML5 web application. SmartCompositionprovides an event-based communication infrastructure which enables inter-component communication as well as message exchange across multiple screens utilizing a WebSocket-based synchronization service.

Keywords: Component-based web engineering · Web components · Distributed multi-device web applications · Web application development · Composition · Reusable components · Multi-screen mashup · HTML5

1 Introduction

Within the last years, the amount of tools for creating user interface mashups (UI mashups) significantly increased. The overall objective of UI mashups is to enable non-experts to create rich web applications [2]. For solving complex tasks an UI mashup consists of several components that offer a limited functionality and are combined and aggregated. While other approaches for creating UI mashups focus on automatic or semi-automatic mashup creation and deployment to desktop as well as mobile screens, our approach eases the creation of UI mashups that run distributed across several screens, so called multi-screen mashups.

The purpose of SmartComposition is to enable local developers to create multi-screen mashups. We assume that a local developer is familiar with basic web technologies, such as HTML5 and CSS [1]. Thus, our approach is based on these technologies and does not require advanced knowledge of JavaScript. Furthermore,

F. Daniel and C. Pautasso (Eds.): RMC 2015, CCIS 591, pp. 50–62, 2016.
DOI: 10.1007/978-3-319-28727-0_4

we want to achieve a high level of reuse of the developed components. This requires loosely coupling and a suitable communication infrastructure to minimize the overhead when integrating them. For enhancing existing web applications to multi-screen mashups, SmartComposition needs to be easily integrable.

In the last years, with the rapid advancement of JavaScript, a lot of client-side components were provided as JavaScript libraries or snippets. Mostly, their functionality is added to standard HTML elements by calling special JavaScript functions that extents them. Those elements are then used as containers to host more dynamically created HTML elements. Common examples would be image slideshows, lightboxes, map sections or enhanced user interface components. Well known libraries that support such components are e.g., jQuery and Dojo.

We observed several limitations and problems using those approaches: First of all, those components cannot be used directly in the HTML code. A placeholder element has to be added to the document on which some JavaScript code is applied. Secondly, such solutions require an advanced knowledge of JavaScript and cumbersome configuration and instantiation is needed. Furthermore, since the components are created in the same DOM (Document Object Model) tree as the host document, a high risk of conflicts with existing elements and style definitions is present.

Most of the currently existing JavaScript components are designed to work separately. A mashup-like scenario, where the composition of multiple components that work together forms a new application, is not tackled. In such scenarios a uniform way to exchange data between components is required. Former research dealt with the integration of inter-widget communication in existing widgets approaches, like W3C or OpenSocial Widgets [7]. Unfortunately, these widget types need to be deployed and hosted in portal environments like Apache Rave or Apache Shindig.

The rest of this paper is organized as follows: First, we outline the context and goals of the SmartComposition approach. In the next chapter, we present related work. Section 4 will describe the SmartComposition approach itself. In Sect. 5 the requested feature checklist is provided. Finally, we describe our preparation for the mashup challenge, the demonstration itself and conclude our paper.

2 Context and Goals of the SmartComposition Approach

The focus of our SmartComposition approach is the client-side composition with an emphasis on user-interface components. To make our approach stand out from other solutions, we set the goal to support and ease the development of multi-screen capable web applications. We want to eliminate the requirement of many other approaches that need a dedicated runtime environment and enable usage in any standard HTML5 website or application. Therefore, we aim to use only client-side JavaScript and standard Web technologies. We do not want to limit the types of components in our approach. Thus, our solution should enable the development and usage of UI as well as data or logic components. A combination of all three types should also be possible. In contrast to existing UI mashup approaches, where components are mostly called *widgets*, we always use the term *component*. This is justified by not limiting our component types to user interface elements. To ease the development of multi-screen capable web applications, we want to provide not only inter-component communication but also a easy to integrate multi-device message exchange functionality.

How we reached our goals will be presented in the following chapters, after we state related approaches for client-side component-based web development.

3 Related Work

Since we especially focus on the component technologies for creating mashups by composition, we will state related work in the field of component technologies in the Web.

jQuery[1] provides a plugin system that enables developers to create extended HTML elements. In most cases the instantiation and configuration is done by selecting the desired element and applying the provided plugin constructor to it. Elements are inserted in the document's DOM and therefore are not encapsulated. Communication features are not included.

Dojo[2] focuses on a more comprehensive approach and provides a UI library called Dijit. Dijit is a widget system layered on top of Dojo. Dojo widgets are instantiated and configured using the "data-dojo-type" and "data-dojo-props" attributes in the HTML markup. The template content in inserted directly in the document's DOM what increases the risk for conflicts. Dojo provides a topic-based publish/subscribe mechanism for communication purposes.

W3C Widgets[3] (also called Packaged Web Apps) and OpenSocial Widgets[4] are open web standards. Since they need to be executed in special platform environments, such as Apache Rave[5] or Apache Shindig[6], the acceptance and usage is limited. They provide encapsulation by running in iFrames and can exploit inter-widget-communication features for composing applications like mashups. The integration of OpenAjax Hub[7] into Apache Rave is an approach to achieve communication between those widgets. The DireWolf framework [4] is one solution that integrates multi-device communication into the Apache Shindig platform.

Another approach is MultiMasher [3]. MultiMasher is a visual tool for multi-device mashups using a direct manipulation interface where a user can select existing UI elements and send them to connected devices. There, the elements will be mashed up with the content that has been sent. Thus, in contrast to the widget approaches, MultiMasher does not support separated components but relies on existing UI elements.

[1] http://jquery.com.
[2] http://dojotoolkit.org.
[3] http://www.w3.org/TR/widgets/.
[4] http://opensocial.atlassian.net/wiki/display/OSD/Specs.
[5] http://rave.apache.org/.
[6] http://shindig.apache.org/.
[7] http://www.openajax.org/member/wiki/OpenAjax_Hub_2.0_Specification.

4 The SmartComposition Approach

The SmartComposition approach is based on the idea of creating mashups by composing loosely coupled components using standard web technologies. In [6], Krug et al. proposed a component-based architecture for multi-screen web applications. We advance the presented ideas by using the *Web Components* technologies for defining and implementing SmartComponents.

We propose a uniform way of defining, implementing and composing loosely coupled independent components by using the new set of W3C standards called *Web Components*. Thus, we support developers in handling those new technologies by providing an extended version of the Polymer framework[8] that wraps the creation of Web Components in an easy-to-use declarative syntax and is enriched with new communication features based on an event-driven architecture. Additionally, we present an optional messaging service that seamlessly integrates into an application developed with the SmartComposition approach and provides message exchange between multiple devices.

The major benefit of using the proposed technologies for creating modern widgets is that no dedicated portal software is needed to host such composed applications. This enables the integration of SmartComponents into common content management systems like WordPress, Drupal or Joomla, as well as in any other HTML5 based website.

To make multi-screen mashup applications more interactive, SmartComponents can be configured to be easily movable by drag-and-drop. Additionally, SmartComponents can also be moved to other connected screens with their state preserved. SmartComponents are stateful DOM objects and provide script interfaces. Thus, developers are able to influence the behavior of the used components on runtime with standard HTML5 DOM methods. SmartComponents can be added, removed and reconfigured at any time. By making SmartComponents available as HTML elements, users that are familiar with HTML but do not have knowledge in programming are also able to create mashups.

In the following section we will guide through the structure of SmartComponents and their technological background.

4.1 Structure of SmartComponents

SmartComponents exploit a set of new W3C technologies called *Web Components*, consisting of *Templates*, *Shadow DOM*, *Custom Elements* and *HTML Imports*. The first technology called *Templates* (http://w3.org/TR/html5/scripting-1.html) defines chunks of markup that are inert but can be activated for use later. That means, the content of the template element is parsed by the parser, but it is inactive and not rendered. Within the `<template>` tags normal HTML markup is used to describe the structure of the components static content. When creating the component, the template's content is copied to an adjunct DOM tree called *Shadow DOM* (http://w3.org/TR/shadow-dom/). The *Shadow DOM* is the second

[8] http://www.polymer-project.org.

new W3C standard in the set of *Web Components*. This adjunct tree of DOM nodes 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.

The template is followed by an optional style section, where the look of the component's content can be defined. Existing style sheet definitions can be reused by including the CSS `@import` statement. To address the custom element that is hosting the component's content, the new pseudo-class `:host` is provided. Due to the previously mentioned scoping, the developer has not to worry about conflicting style definitions, class names or IDs.

New SmartComponents are defined using a declarative syntax provided by the Polymer framework. An example definition file for a SmartComponent is shown in Listing 1.

```
<dom-module id="wikipedia-extract">
  <style>
    :host { display: inline-block; }
  </style>
  <template>
    <div id="container"></div>
  </template>
  <script>
    Polymer({
      is:'wikipedia-extract',
      behaviors: [Polymer.SmartComponentBehavior],
      properties: {
        query: {
          type: String,
          reflectToAttribute: true,
          observer:'queryChanged'
        }
      },
      queryChanged: function() {
        // Request to fetch data from Wikipedia
      },
      attached: function() {
        this.subscribe('wiki', this.queryReceived);
      },
      detached: function() {
        this.unsubscribe('wiki', this.queryReceived);
      },
      queryReceived: function(message) {
        this.query = message.data;
      }
    });
  </script>
</dom-module>
```

Listing 1. Definition file of a SmartComponent

This declarative description handles all necessary actions to create a custom element and setting up event bindings. A developer can define any number of properties that can be configured with type settings, observer functions and e.g. reflection to attributes. By using the *behaviors* property to inject our *Smart-ComponentBehavior*, we can provide our later described extensions for inter-component communication without modifying the Polymer framework itself. This supports the maintainability of both our extension and the framework.

SmartComponents are new types of DOM elements that can be defined by developers. They are stateful DOM objects and provide script interfaces. New components can be easily integrated in a website by using *HTML Imports* (http://w3.org/TR/html-imports/). The import statement uses the `<link>` tag to load external definition files (see Listing 2). The new custom element tag can be instantly used in the HTML markup after importing the component resource file. SmartComponents are registered as new HTML elements. Thus, they can be used them in the same way as other standard elements. The usage requires no knowledge of JavaScript. Configuration is possible through attributes or child elements.

```
<html>
<head>
  <link rel="import" href="smart-component.html">
</head>
<body>
  <smart-component some-attr="some-value"></smart-component>
</body>
</html>
```

Listing 2. Usage of SmartComponents in HTML5 websites

In the following section we will describe the communication aspect we integrated into the components of the SmartComposition approach.

4.2 Inter-Component Communication

Loosely coupling of components is important to ensure reuse and enable new compositions. To support message exchange between SmartComponents we therefore propose an event-driven communication channel using a topic-based publish/subscribe mechanism. Figure 1 provides a simplified overview of the inter-component and inter-device communication architecture of the SmartComposition approach.

Components can consume and produce events described by a topic and the attached data. The publish/subscribe message bus is implemented using the JavaScript's native eventing system. Messages are sent using a custom event and received by adding an event listener for that custom event. The payload can be structured objects or simple values.

As it is displayed in Listing 1, the *subscribe* method should be called within the life-cycle event *attached*, which means when the component is added to the DOM.

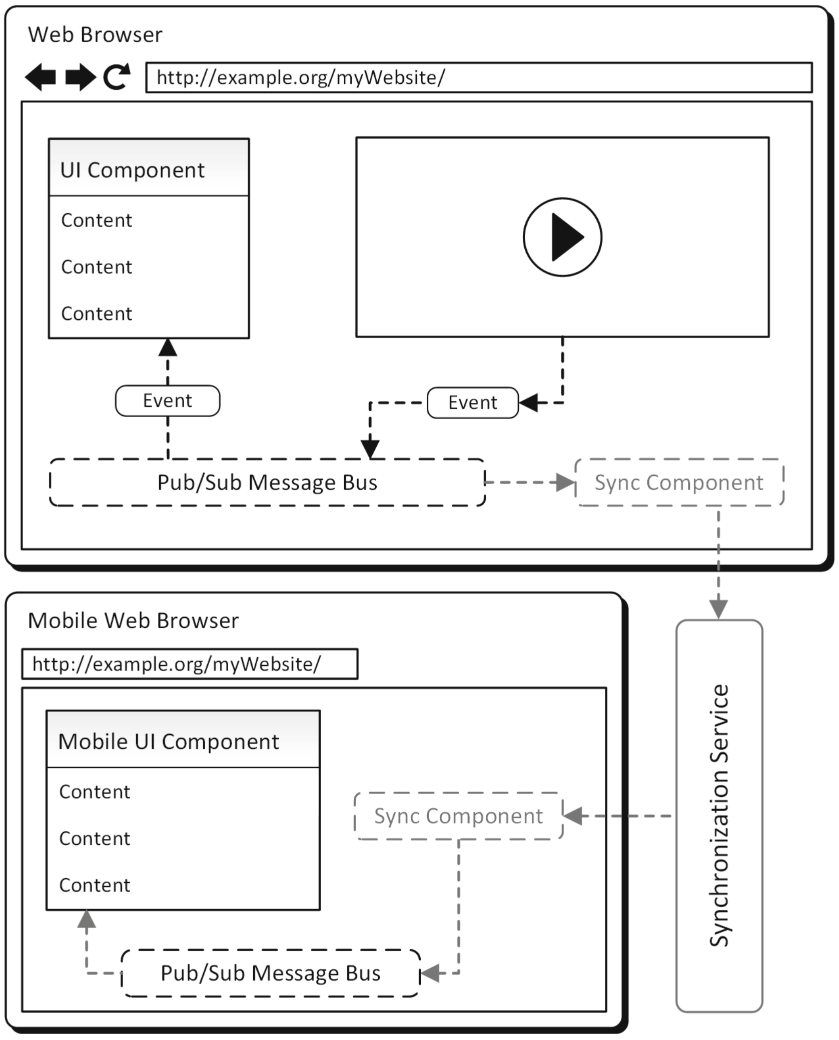


Fig. 1. Simplified inter-component and inter-device communication architecture

But this is not a limitation. In fact, it can be called anytime after attaching. To stop the SmartComponent from consuming events after it is removed from the DOM, the *unsubscribe* method should be called within the *detached* life-cycle event. A SmartComponent can have any number of subscriptions to any topic. Messages can be sent using a *publish(topic, data)* method that is available in the SmartComponent context. All communication functionality is injected using the *behaviors* property and contained in our implemented *SmartComponentBehavior*.

By employing this eventing system and by giving the message a predefined structure, containing the topic and the data, we achieve a topic-based and event-driven communication channel. Without blocking the user interface, we ensure high performant and low latency communication by relying on JavaScript's native event system.

In the following section we show how inter-component communication is extended for multi-device usage.

4.3 Inter-Device Communication

By providing a WebSocket-based synchronization service, we enable developers to easily create multi-device-capable web applications. Our approach proposes a stand-alone solution with no dependencies and side-effects on other components. The solution consists of a synchronization server and a client-side messaging service. The client-side component is also implemented as SmartComponent that captures all events transmitted 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 message bus where the components will be notified.

We are utilizing the WebSocket protocol (https://tools.ietf.org/html/rfc6455) for the client-server communication. This provides us with a full-duplex, low-latency communication channel based on standard web technologies. The server-side component is implemented as a WebSocket server using Node.js.

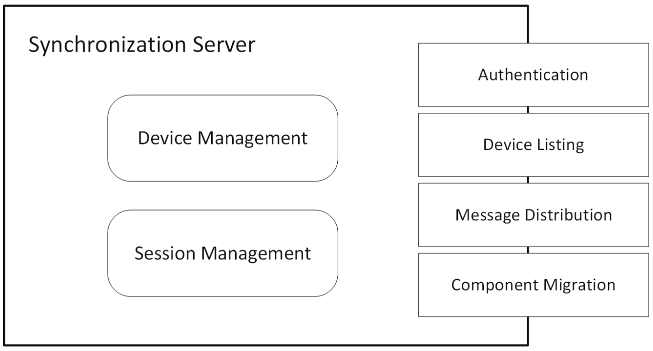


Fig. 2. Basic functionality of the synchronization server

The WebSocket server (see Fig. 2) provides functionality for a set of message types (authentication, clients, ping, data) and can easily be extended. Received messages are analyzed and broadcast to groups (*sessions*) of connected devices. We define the term *connected devices* as: devices with the same synchronization endpoint that share the same session identifier, i.e. context. The session identifier enables the usage of one synchronization endpoints for multiple application contexts. The basic functionality is that messages are only distributed to devices within the same session. Another task of the server-side component is the management of connected devices. On connection, each device will get an up-to-date list of connected devices with their details (name, type, identifier). This list is also updated and distributed if a client connects, disconnects or changes its details.

One major advantage of our synchronization approach is that no reconfiguration of existing components is necessary for multi-device communication. Since the messaging service is working like a hook, all messages sent by the SmartComponents are captured without changing the code or configuration.

5 Feature Checklist

Mashup Type	Hybrid mashups
Component Types	Data components
	Logic components
	UI components
Runtime Location	Both Client and Server
Integration Logic	Choreographed integration
Instantiation Lifecycle	Short-living
Targeted End-User	Local Developers
Automation Degree	Manual
Liveness Level	Level 4 (Dynamic Modificationof Running Mashup)
Interaction Technique	Editable Example
Online User Community	None

6 Mashup Challenge

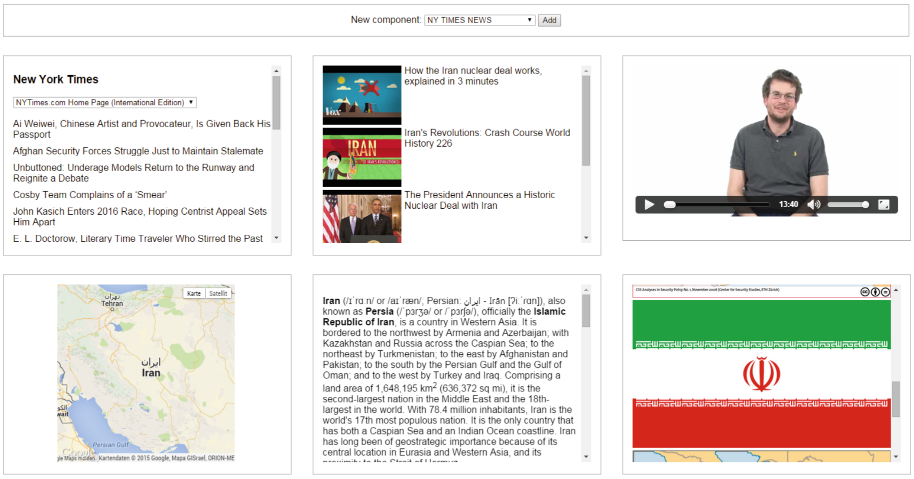
6.1 The Presented Mashup

We demonstrate our SmartComposition approach by presenting a distributed media enrichment application using various SmartComponents to showcase web application development through client-side composition. The application implements a mashup scenario, which was previously discussed and implemented without the usage of *Web Components* in [5]. One possible resulting mashup can be seen in Fig. 3.

To create an application by composition, multiple SmartComponents can be imported and inserted into an HTML website as it is displayed in Listing 3.

We start our mashup with an empty web page that has an option to add new components to the application. Using the New York Times news feed component as the starting point, we add more and more components that work together to form a new interactive experience. For a detailed description of the mashup components and how they work together see Sect. 6.3.

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. Our demos can be


New component: NY TIMES NEWS
Add
New York Times
NYTimes.com Home Page (International Edition)
13:40
Karte
Satellit
تهران
Tehran
ایران
Iran
Persian Gulf

Ai Weiwei, Chinese Artist and Provocateur, Is Given Back His Passport
Afghan Security Forces Struggle Just to Maintain Stalemate
Unbuttoned: Underage Models Return to the Runway and Reignite a Debate
Cosby Team Complains of a 'Smear'
John Kasich Enters 2016 Race, Hoping Centrist Appeal Sets Him Apart
E. L. Doctorow, Literary Time Traveler Who Stirred the Past

How the Iran nuclear deal works, explained in 3 minutes
Iran's Revolutions: Crash Course World History 226
The President Announces a Historic Nuclear Deal with Iran

Iran (/ɪˈrɑːn/ or /aɪˈræn/; Persian: ایران - Īrān [ʔiːˈrɑn]), also known as **Persia** (/ˈpɜrʒə/ or /ˈpɜrʃə/), officially the **Islamic Republic of Iran**, is a country in Western Asia. It is bordered to the northwest by Armenia and Azerbaijan; with Kazakhstan and Russia across the Caspian Sea; to the northeast by Turkmenistan; to the east by Afghanistan and Pakistan; to the south by the Persian Gulf and the Gulf of Oman; and to the west by Turkey and Iraq. Comprising a land area of 1,648,195 km^2 (636,372 sq mi), it is the second-largest nation in the Middle East and the 18th-largest in the world. With 78.4 million inhabitants, Iran is the world's 17th most populous nation. It is the only country that has both a Caspian Sea and an Indian Ocean coastline. Iran has long been of geostrategic importance because of its central location in Eurasia and Western Asia, and its

Fig. 3. Screenshot of the presented mashup

used in any modern web browser without the installation of additional software. Not all technologies we are using are currently implemented in all browsers as most of them are still W3C working drafts. By optionally using the webcomponent.js polyfills (http://webcomponents.org/polyfills/), SmartComponents are also enabled in web browsers that lack native support.

Online Demonstration:
http://vsr-demo.informatik.tu-chemnitz.de/smartcomposition/icwe2015/

```
<html>
 <head>
  <link rel="import" href="nytimes-news-component.html">
  <link rel="import" href="semantic-extraction-component.html">
   ...
 </head>
 <body>
  <nytimes-news></nytimes-news>
  <semantic-extraction></semantic-extraction>
  <youtube-search query="Iran"></youtube-search>
  <smart-video></smart-video>
  <google-geocoder address="Iran"></google-geocoder>
  <google-map lat="51" lng="12" zoom="12"></google-map>
  <twitter-tweets query="Iran"></twitter-tweets>
  <wikipedia-extract query="Iran"></wikipedia-extract>
  <google-images query="Iran"></google-images>
 </body>
</html>
```

Listing 3. Application development by composition

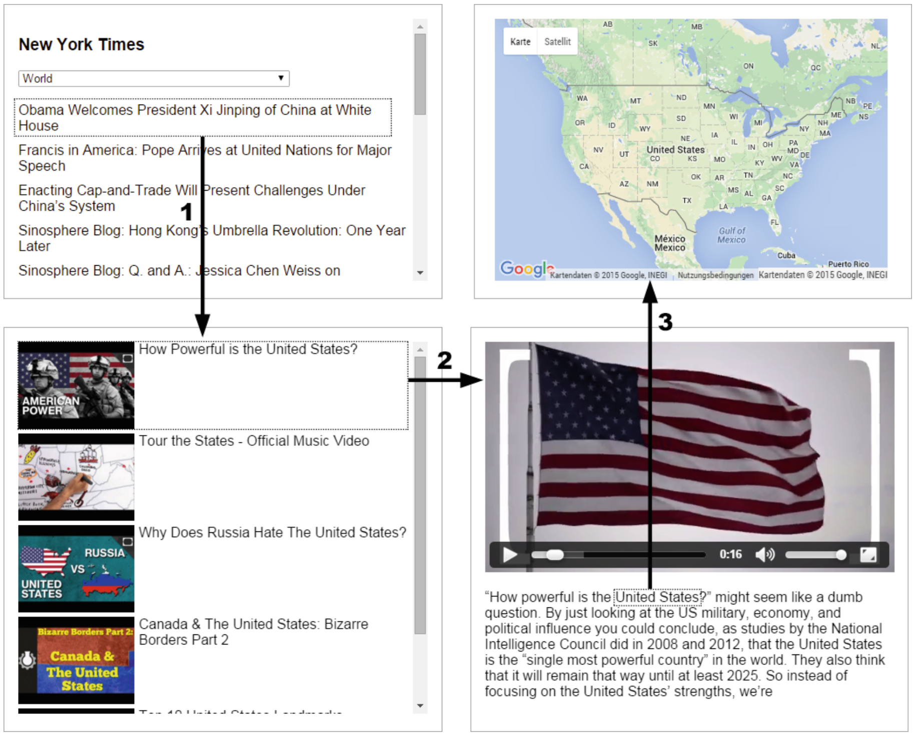
6.2 Preparation of the Challenge

In preparation of the challenge, 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. Firstly, we implemented a component that retrieves the New York Times RSS feed and extracts the news entries separated by categories. It provides a selection of the category and displays all matching news headlines. When the user clicks on one entry, the news text is published to the message bus of the application. Secondly, we reused a component that uses the AlchemyAPI to extract keywords (entities) from text by applying natural language processing technologies. These keywords are categorized and again published to our message bus.

Additionally, we created a YouTube search component. This component takes a search phrase as input and displays a list of videos that are related to that phrase. Furthermore, we implemented a special video component that publishes messages at specific timestamps - in this case parts of the transcript - while playing a video. This is done by exploiting the TextTrack-API and an attached VTT subtitles file containing time-based metadata. The video component works with local videos as input as well as with YouTube URLs that are automatically resolved. To visualize information about different entities, we implemented components that catch data from web sources, like Twitter, Google Maps, Images and Wikipedia. A drawback that needs to mentioned is that the topic names and data formats of connected components have to be known by the developers.

6.3 The Demo Flow

In general, the first source of information can be any component. In our specific demonstration, we use the New York Times news feed as an entry point. New components can be easily added to the application by either stating them in the markup or adding them dynamically using the given select box and button. When the user clicks on one of the displayed news headline entries, the component publishes the corresponding news text. By adding the semantic extraction component, the mashup is able to obtain different entities from published text content. They are annotated with categories and can be used by other components to retrieve related information. If there is e.g., an entity categorized as *location*, the Google Maps Geocoder component is using the entity to convert it to geographical coordinates that can be again consumed by the Google Maps component to display this place on a map. Furthermore, entities of the type *person* can be e.g., visualized by the Google Images or Wikipedia component. To make the mashup more interactive, not only the news feed is used as information source. The gathered entities are also used by the YouTube search component to retrieve related videos with subtitles. If the user clicks on one of the listed videos, it will be passed to the video component that is able to play the video and at the same time publish time-based metadata. The metadata - in this case the transcript - is also used for semantic extraction and will trigger the display of different kind of information visualizations. An example message flow can be seen in Fig. 4.


New York Times

World

Obama Welcomes President Xi Jinping of China at White House

Francis in America: Pope Arrives at United Nations for Major Speech

Enacting Cap-and-Trade Will Present Challenges Under China's System

Sinosphere Blog: Hong Kong's Umbrella Revolution: One Year Later

Sinosphere Blog: Q. and A.: Jessica Chen Weiss on

How Powerful is the United States?

Tour the States - Official Music Video

Why Does Russia Hate The United States?

Canada & The United States: Bizarre Borders Part 2

"How powerful is the United States?" might seem like a dumb question. By just looking at the US military, economy, and political influence you could conclude, as studies by the National Intelligence Council did in 2008 and 2012, that the United States is the "single most powerful country" in the world. They also think that it will remain that way until at least 2025. So instead of focusing on the United States' strengths, we're

Fig. 4. Message flow of an example mashup

The extension of the mashup to use multi-device communication is straightforward. First of all, a synchronization server has to be set up, which is a Node.js WebSocket server. Additionally, the messaging service has to be included in the web application and configured with the endpoint address as displayed in Listing 4.

```
<link rel="import" href="MessagingService.html">

<messaging-service
  endpoint="http://example.org:1234" session="SessionID">
</messaging-service>
```

Listing 4. Code snippet of the messaging-service

Any reconfiguration or even altering of code of existing SmartComponents is not necessary. All published events will now be synchronized between multiple connected devices. Applying it to the mashup application, it then can be used on different devices in parallel with synchronized state without touching the code of the components. Thus, the user can display e.g., the Google Map on his mobile device while watching the video on his laptop.

7 Conclusion

In this paper we presented extended Web Components, called SmartComponents, as a part of the SmartComposition approach. We support developers in creating multi-screen-enabled mashups and other complex, distributed web applications. Using the Polymer framework that wraps necessary functionality lowers the barrier of using the new W3C Web Components technologies. Since SmartComponents are custom elements that become first-class HTML elements, you can 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 SmartComponents into an adjunct shadow DOM subtree prevents CSS rules and IDs of elements from conflicting. Our extension of adding an event-based communication channel as well as the provision of a WebSocket-based synchronization service enables the composition of mashups for usage across distributed platforms and multiple devices. Further research will address how to provide a repository to store and distribute reusable SmartComponents and the description of communication interfaces and topic names to ensure hassle-free composition of SmartComponents.

References

1. Aghaee, S., Nowak, M., Pautasso, C.: Reusable decision space for mashup tool design. In: 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 211–220, Copenhagen, Denmark, June 2012
2. Chudnovskyy, O., Fischer, C., Gaedke, M., Pietschmann, S.: Inter-widget communication by demonstration in user interface mashups. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 502–505. Springer, Heidelberg (2013)
3. Husmann, M., Nebeling, M., Norrie, M.C.: Multimasher: a visual tool for multi-device mashups. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 27–38. Springer, Heidelberg (2013)
4. Kovachev, D., Renzel, D., Nicolaescu, P., Klamma, R.: Direwolf - distributing and migrating user interfaces for widget-based web applications. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 99–113. Springer, Heidelberg (2013)
5. Krug, M., Wiedemann, F., Gaedke, M.: Enhancing media enrichment by semantic extraction. In: Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion, WWW Companion 2014, pp. 111–114. International World Wide Web Conferences Steering Committee (2014)
6. Krug, M., Wiedemann, F., Gaedke, M.: Smartcomposition: a component-based approach for creating multi-screen mashups. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 236–253. Springer, Heidelberg (2014)
7. Wilson, S., Daniel, F., Jugel, U., Soi, S.: Orchestrated user interface mashups using W3C widgets. In: Harth, A., Koch, N. (eds.) ICWE 2011. LNCS, vol. 7059, pp. 49–61. Springer, Heidelberg (2012)