AttributeLinking: Exploiting Attributes for Inter-component Communication

Michael Krug(✉) and Martin Gaedke

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

Abstract. In this paper, we propose exploiting attributes of client-side web components to provide inter-component communication by external configuration. With the standardization of WebComponents, the Web is finally getting a uniform way to define and use client-side components. We determined that DOM elements already provide a standard configuration interface: attributes. Using the WebComponents technologies for state-of-the-art user-interface components, attributes can also act as output interfaces. By providing an Attribute-Link component, new web applications can be composed directly in the markup without knowledge of JavaScript. With the integration of a multi-device supporting MessagingService, components can be even linked across multiple connected devices. This enables the development of distributed user interfaces.

Keywords: Web components · Web application development · Composition · Distributed user interfaces · Reusable components

1 Introduction

Component-based application development is based on the composition of multiple components that are reused and combined to form new applications. Applying this to the Web creates new challenges, like the definition of independent components and loosely coupling. A lot of approaches were developed in the last years, but a standardized way of defining and using components for client-side Web application development was missing. Recently, the W3C is working on creating a new set of standards for components for the Web called *WebComponents*[1]. With those standards developers are now able to define new types of DOM elements with custom functionality. Similar to UI mashups, where user-interface components were mostly referred to as widgets, inter-component communication, which is required for application development by composition, is a central problem. This is unfortunately not covered by the W3C specifications. We analyzed different DOM elements and determined that most of them provide an accessible (input) interface, which is mainly used for configuration: *attributes*. Consequently, this interface is also available for WebComponents.

[1] WebComponents - W3C Wiki https://www.w3.org/wiki/WebComponents/.

S. Casteleyn et al. (Eds.): ICWE 2016 Workshops, LNCS 9881, pp. 157–161, 2016.
DOI: 10.1007/978-3-319-46963-8_13

```
<geo-coder address="Chemnitz"// input interface
           lat="50.83"       // output interface
           lng="12.92">       // output interface
</geo-coder>
```

Listing 1. Geo-Coder component with multiple attributes

With an accordingly configuration, attributes of WebComponents can also be used as output interfaces. To explain this idea, an example in Listing 1 is stated, where a *Geo-Coder* component is shown that converts an address into geographic coordinates (latitude, longitude), e.g., to show a place on a map. Thus, the attribute *address* is used as an input interface and *lat* and *lng* are the resulting output variables. If we now assume that there is also a *Map* component with latitude and longitude as input variables mapped to the according attributes, a composition of those elements can be achieved by connecting the output variables of the *Geo-Coder* to the input ones of the *Map*.

Therefore, we want to encourage developers to use attributes as communication interfaces when creating new WebComponents. By enabling external configuration of inter-component data exchange the components do not need to include any communication functionality by themselves. Thereby, components stay reusable and do not need knowledge of other components.

2 Related Work

Polymer, a WebComponent framework from Google, supports data bindings within markup through placeholders in attributes and by placing target and source within a binding element. There are also several JavaScript frameworks that support data binding, e.g., AngularJS, Knockout or Rivets.js. They need an attached data model to specifically bind data to the element's content or attributes. Furthermore, some publish/subscribe or event-based communication approaches for components or widgets, like presented in [1,2] and [3], exist. They have the disadvantage of configuring communication aspects within the components. There are only limited options to configure which components shall exchange data. Data transformation and message distribution for multi-device applications is not supported.

3 The AttributeLinking Approach

Based on the assumption that WebComponents are DOM-Elements that provide their in- and output interfaces through attributes, we enable inter-component communication by linking attributes through external configuration to facilitate web application development by composition. How to connect those interfaces?

3.1 The Attribute-Link Component

We propose to connect the attribute interfaces of components using a stand-alone *Attribute-Link* component, which is configured using three attributes: *source*, *target* and an optional *transformation* (see Listing 2). The *Attribute-Link* component is also implemented as a WebComponent. Thus, it can be natively used in web browsers supporting those standards and can be deployed directly in the markup without knowledge of JavaScript. To address an element we propose to use the established *CSS selector* syntax[2]. This syntax is applied to both the source as well as the target selector. Since CSS selectors can return multiple elements, we also support multiple elements as target and source. Enabling a cardinality of both ends from 1 to n. The attribute is addressed by its name separated by an `@` sign. Example: `geo-coder#id1@address`. The *Attribute-Link* can be included multiple times in the web application to define a complex inter-component communication setup. To watch for attribute changes without affecting the responsiveness of the application we use native DOM mutation observers[3] that set up microtasks and provide a callback for registered changes of the specific element's attribute. If a transformation function was specified, it will be applied and the resulting value is propagated to the defined attribute of the according target element.

```
<attribute-link source="selector@attribute"
                target="selector@attribute">
                transformation="JavaScript-Code"/*Optional*/>
</attribute-link>
```

Listing 2. Syntax of the proposed Attribute-Link component

3.2 Distributing Attribute Changes

In our approach, we also focus on the development distributed user interfaces. Thus, we propose to additionally address target components on connected devices. This is achieved using a *Messaging-Service* component and a *Messaging-Server* we provide. Our presented *Attribute-Link* component seamlessly integrates with the *Messaging-Service* by placing it in the DOM as a child element like display in Listing 3. It notifies the *Messaging-Service* of changed attributes that shall be distributed by dispatching a custom event. This event contains the target element selector, the attribute name and the new value to be set. Using the *WebSocket protocol*[4] the *Messaging-Server* is contacted to distribute this message to other connected devices (browsers running the application with the same endpoint configured that share the same context) (cf Fig. 1). The other devices also need to have instantiated an accordingly configured *Messaging-Service* component that will then propagate the received messages to the target elements in their DOM. We also provide a configuration option to only target remote components.

[2] Selectors Level 3 https://www.w3.org/TR/selectors/.
[3] DOM Standard https://dom.spec.whatwg.org/#mutation-observers.
[4] The WebSocket Protocol http://tools.ietf.org/html/rfc6455.

```
<messaging-service endpoint="protocol://address:port"
                   session="Session-Identifier">
        <attribute-link [...]></attribute-link>
</messaging-service>
```

Listing 3. Syntax of the Messaging-Service in combination with an Attribute-Link

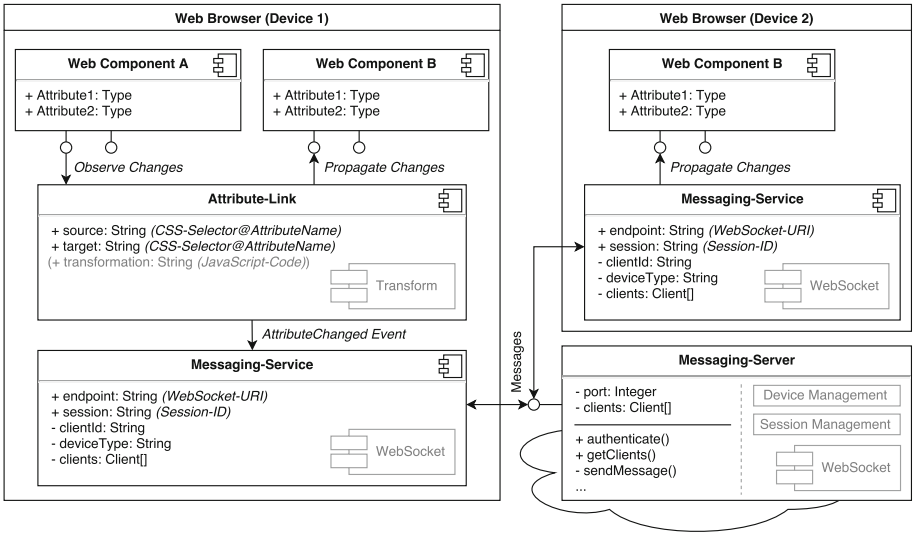

Fig. 1. Components of the AttributeLinking approach

4 Conclusion

In this paper, we proposed the external linking of attributes of DOM-based components to enable the composition of new web applications. Consequently, we want to start a discussion of the usage of attributes as in- and output interfaces for components in the Web. To demonstrate our idea, we presented an Attribute-Link component that is capable of observing and propagating attribute changes from and to multiple DOM elements. With an optional Messaging-Service those changes can also be distributed to connected devices. The concept of external configuration eliminates the need of modifying the components to enable communication. Further research will address how to enhance the support for distributed interfaces by providing more configuration options to e.g., select components on specific devices.

Online Demonstration: http://myvsr.eu/demo/dui/

References

1. Chudnovskyy, O., Müller, S., Gaedke, M.: Extending web standards-based widgets towards inter-widget communication. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 93–96. Springer, Heidelberg (2012)
2. Krug, M., Gaedke, M.: SmartComposition: bringing component-based software engineering to the web. In: Proceedings of the 17th International Conference on Information Integration and Web-Based Applications and Services, pp. 474–477. ACM (2015)
3. Sire, S., Paquier, M., Vagner, A., Bogaerts, J.: A messaging API for inter-widgets communication. In: Proceedings of the 18th International Conference on World Wide Web, pp. 1115–1116. ACM (2009)