User-Customizable Web Components for Building One-Page Sites

User-Customizable Web Components for Building One-Page Sites

Pasquale Lisena, Jetmir Xhembulla, Giovanni Malnati
Department of Control and Computer Engineering
Politecnico di Torino
Turin, Italy
e-mail: {pasquale.lisena, jetmir.xhembulla, giovanni.malnati}@polito.it

Pasquale Morra
Research and Development Division
Seat PagineGialle S.p.a.
Turin, Italy
e-mail: morra.pasquale@seat.it

***Abstract*—Most of online website builders work by combining and customizing reusable HTML modules. This approach could rise the risk of conflicts among modules. The World Wide Web Consortium (W3C) is writing the specification of Web Components. This standard provides a browser-native solution in order to realize encapsulated Document Object Model (DOM) elements, in which the Cascading Style Sheets (CSS) and JavaScript scope is locally bound and the interaction with the document is strictly designed by the component author. Upon this standard, libraries have been built, Google's Polymer being an example, which provide a declarative and easy way to realize Components. In this paper, we provide a solution to the module approach limit in website builders by using Web Components as modules that are customizable by the end user. Our approach uses standard web technologies that modern browsers are natively supporting. We describe how a customizable Web Component is designed and how to bind their options with the generator UI. Furthermore, we will show an application of this approach in a Landing Page generator. We demonstrate that the generator could import again the generated HyperText Markup Language (HTML) and edit it, without any intermediary data structure (i.e., eXtensible Markup Language, XML or JavaScript Object Notation, Json). Finally, we outline further future development of this approach.**

Keywords-Website generation, Web Components, HTML5, modularity, end-user generation, SME.

I. INTRODUCTION

The spread of smartphones in the last 5 years has deeply changed the market, forcing businesses of any kind and size to conform their online presence, in order to avoid the lack of an important market share.

Overwhelmed by contents, the constantly connected consumers use now to spend the slightest of their time on a Web site, mostly by mobile devices, becoming bothered about navigation trees and expecting to find on the first page – with no more "click and wait" – the information they need. For these and further reasons, one-page sites are nowadays very popular, often together with responsive design patterns.

This worldwide transformation is very quick and radical and produces two consequences: on one hand, businesses - which are mostly devoid of the required skills for managing and distributing contents on their own - had to delegate these tasks to service companies; on the other hand, the media agencies - in order to reduce costs and fulfill the market demand of high customized products - have left the previous artisanal production methods in favor of a modules-based approach.

Following the mission of digitalizing its wide customer portfolio of enterprises, the Italian media agency Seat PagineGialle has identified the semi-automatization of the process for creating pages as a key point in various context: we are talking about low budgets, e.g., small and medium sized enterprises (SMEs) websites [1], and time limited contents, e.g., a promotional online campaign.

The new World Wide Web Consortium (W3C) standard of Web Components represents a standard solution for modularization and reusing of HyperText Markup Language (HTML) in a web page. Isolation of scope, reusability, and freedom from server-side logic are some of the advantages of this standard. This solution is – either natively or thanks to the so called "polyfills" – built directly in the browser.

This paper will present a Web page generator that uses Web Components as modules. In our Web application, these Web Components should not be assembled on the code by the developer, but a friendly interface makes their combination, manipulation and customization available to the end-user. As corporate requirement, integration with other existing and different Seat PagineGialle products must be possible. The rest of the paper is structured as follows: Section II overviews briefly current trends in website generator, with strengths and limits and describes the standard of Web Components, that we use as modules in Section III. Section IV shows an application in a real production environment. Finally, conclusions are drawn in Section V.

II. STATE OF THE ART

A. Trends and limits in website building

A notable number of online website builders is available on the market, designed for the end user with no knowledge about HTML, Cascading Style Sheets (CSS and JavaScript. These tools have the most common and powerful feature in the modular design [2]: standalone parts of the page, called *modules*, could be combined – often thanks to a drag and drop interface - and customized in style, color, text content and position, so that their reuse produces each time a different look and content. In the resulting HTML

 ISBN: 978-1-61208-468-8

document, modules are actually slices of its Document Object Model (DOM) that represent headers, footers, images gallery, text boxes and various kinds of widgets. Each module should follow the generator internal set of rules and conventions (framework) in order to avoid conflicts with its siblings: duplicated IDs, influence of CSS rules from other modules (or vice versa, overflow of their CSS outwards), not scoped scripts [3]. Each framework has his own syntax that produces a lack of interoperability between different technologies [4].

Usually, the final output of this builder is a thick tree of nested tag (mostly DIVs, the most common Document Division in HTML) that is hardly to reconvert into an editable format. These template-based solutions [1] [5] need an eXtensible Markup Language (XML) or JavaScript Object Notation (Json) structure with a list of modules and options, stored on the builder server for future editing purpose; this is an expedient, because the proper language for describing a Web page is HTML. Other applications could use modules generated on the server, like portlets: however, they need specific server environments able to deploy and serve them, hardly to integrate in custom applications as it happens in the Seat PagineGialle case.

B. The Web Components standard

In the context of the HTML5 revolution, the W3C is defining a standard for Web Components. This standard allows you to create new type of DOM elements and use them in a document as if they were DIVs, INPUTs and other standard HTML tags. Creating a component means writing its HTML template, defining its CSS rules and managing its properties, method and lifecycle with JavaScript. For a Web developer, using a Web Component is as simple as inserting a tag <my-component-name> in the HTML and dealing with it like any HTML native tag.

The family of the Web Components W3C standards includes four new specifications about:

- Custom Elements [6], that enable to define and use custom DOM elements in a document;
- HTML Imports [7], for including and reusing HTML documents as dependencies;
- Templates [8], inert DOM elements for describing HTML structures;
- Shadow DOM [9], a method for encapsulating DOM trees and bounding their interaction with the whole document.

Thanks to these technologies, we can define the structure of a component in a not-rendered <template> tag and register it as Custom Element, so that the browser becomes aware of the match between the component tag name, e.g., <descriptive-content>, and its definition. When an instance of a registered component is created in the page, the browser creates a parallel tree of DOM – called Shadow DOM – associated to the component element. This Shadow DOM contains the structure we defined in the <template>. Although not visible as a child node of the element, this structure is rendered and the user can interact with it. All that lives in the Shadow DOM has its own isolated scope and can react to events and attributes modification on its parent component. This isolation solves all problems about duplicated IDs and ingestion of external CSS; besides, it provides a bounded scope to scripts: the Shadow DOM is solidly separated from the main document [10] [11], and the only possible interactions are those explicitly allowed by the component designer. Finally, the specification introduce a standard way for importing components, with <link rel="import"> tag.

At a glance, this specification covers all the needed requirements: reusability, isolation of JS and CSS and it is a browser standard, so it is fully compatible with any other technology that runs on the server or on the client. These and other advantages of Web Components ecosystem have been investigated thoroughly in [4].

C. Polymer

In the last years, many libraries have been developed with the dual aim to extend the support to older browsers by using polyfills [12], and to further simplify the implementation of the Web Components. Recently Google released the version 1.0 of the *Polymer* library [13], which offers a declarative way for creating components. In Polymer, a component definition is an HTML page that contains imported resources (dependencies, style, template) and a call to the *Polymer* function for the configuration of properties, methods and lifecycle callbacks. We choose Polymer for the clearness of component's code and for its ease-of use.

III. WEB COMPONENTS AS MODULES

We can split the core of our approach in two complementary branches: the design of a component and its manipulation.

A. Design of a component

Polymer provides a declarative syntax for creating Web Components. We describe the general structure of a Polymer component and we provide details only in those parts that we added or that are functional to the next tasks. We refer to the example in Figure 1.

The whole component is wrapped in an inert tag `<dom-module>`. Its id attribute is the tag name of our component. All the nodes contained in `<dom-module>` will be encapsulated in the Shadow DOM of the tag. We can group these nodes in three distinct sections.

The first one is the style section, which defines the look of the component. Thanks to DOM isolation, there are no constraints about specificity of rules, because they will be applied only in the context of the component. Polymer also allows including other style-specific components for styling, like a CSS-reset or a common base style for all components. Moreover, the library includes support to the CSS Variables specification [14], currently at working draft state. For our

 ISBN: 978-1-61208-468-8

purposes, we need to know that you can apply the same CSS variable and so the same value to different selectors and that this value could be set through Polymer Application programming interface (API).

The second part is the <template> tag, the content of which will form the internal DOM structure of our component. Double curly brackets denote the insertion point for properties value, so that "{{text}}" will be replaced with the value of the "text" property.

```
<dom-module id="descriptive-content">
 <style include="component-base"></style>
 <style>
   p { color: var(--descriptive-text-color);  }
 </style>
 <template><p>{{text}}</p></template>
 <script>
   Polymer({
     is: 'descriptive-content',
     behaviors: [ComponentBehavior],
     properties: {
        text: {
          type: String,
          logicType: 'textarea',
          value: 'Lorem ipsum....',
          label: 'Text',
          reflectToAttribute: true,
          customizable: true
        },
        textColor: {
          type: String,
          logicType: 'color',
          value: '#ffffff',
          cssVariable: '--descriptive-text-color',
          label: 'Text color',
          reflectToAttribute: true,
          customizable: true,
          observer: 'computeStyle'
        }, // other properties
     }
     //methods and lifecycle callback
});
</dom-module>
```

Figure 1. The definition of <descriptive-content> component.

```
<descriptive-content text="Lorem ipsum..." text-
color="#ffffff">
    #shadow-root //not shown
        <style> ... </style>
        <p> Lorem ipsum...</p>
</descriptive-content>
```

Figure 2. The usage of <descriptive-content> component.

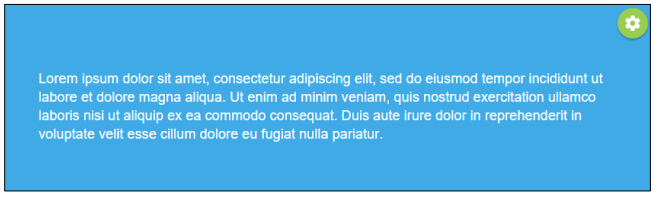

Figure 3. The <descriptive-content> component as it appears in a browser, with the button for customizing settings.

Finally, we deal the registration of the component through the Polymer function. The *properties* object contains the value that will be bound in the template. According to the library, a property has the following sub properties:

- *type*, the JavaScript type of the property,
- *value*, its default value,
- *reflectToAttribute*, if true, causes attribute to be set on the host node when the property value changes,
- *observer*, a method to call on property changes.

In addition, we added the following properties:

- *customizable*, when set to true, this property should be used as customizable option,
- *logicType*, refers to a human concept rather than a coding one; each logic type has a specific User Interface (UI) input element; we support as logic types "text", "color", "background", "textarea", "image";
- *cssVariable*, means that this property is connected to a CSS variable in the <style> tag,
- a human-readable *label* for displaying purpose.

A modification of the attributes on the component host tag (Figure 2) will reflect in a modification on the properties and consequently on the component's content or style as it is rendered by the browser (Figure 3).

We specify also a *behavior*, which is Polymer's way of making certain properties and methods inheritable. The *ComponentBehavior* is a custom behavior that manages some customization-related tasks. It shows an options button on the element that triggers, when clicked, a "settingRequested" event. Additionally, it defines the "computeStyle" method, set as observer of properties with cssVariable in order to propagate changes to the CSS variables.

B. *Component manipulation*

The generator should read the components properties and provide the user with a proper User Interface (UI) for modifying it.

Manipulation starts when our page generator interceps the "settingRequested" event from one of the components. Component customizable properties are retrieved from the source DOM element via JavaScript. For each of them, we read the *logicType* property and according to it, we choose a proper input element. For some types, the choice is an HTML input element: an <input type="text"> for a "text", a <textarea> for a "textarea". For types like "background" or "image", we have not a suitable input tag. Once again, we use Web Components specification for creating custom <background-input> and <image-input> element. For example, we implement a complex input for background, letting the user to choose between a color, an uploaded image, an image from our gallery and a transparent

 ISBN: 978-1-61208-468-8

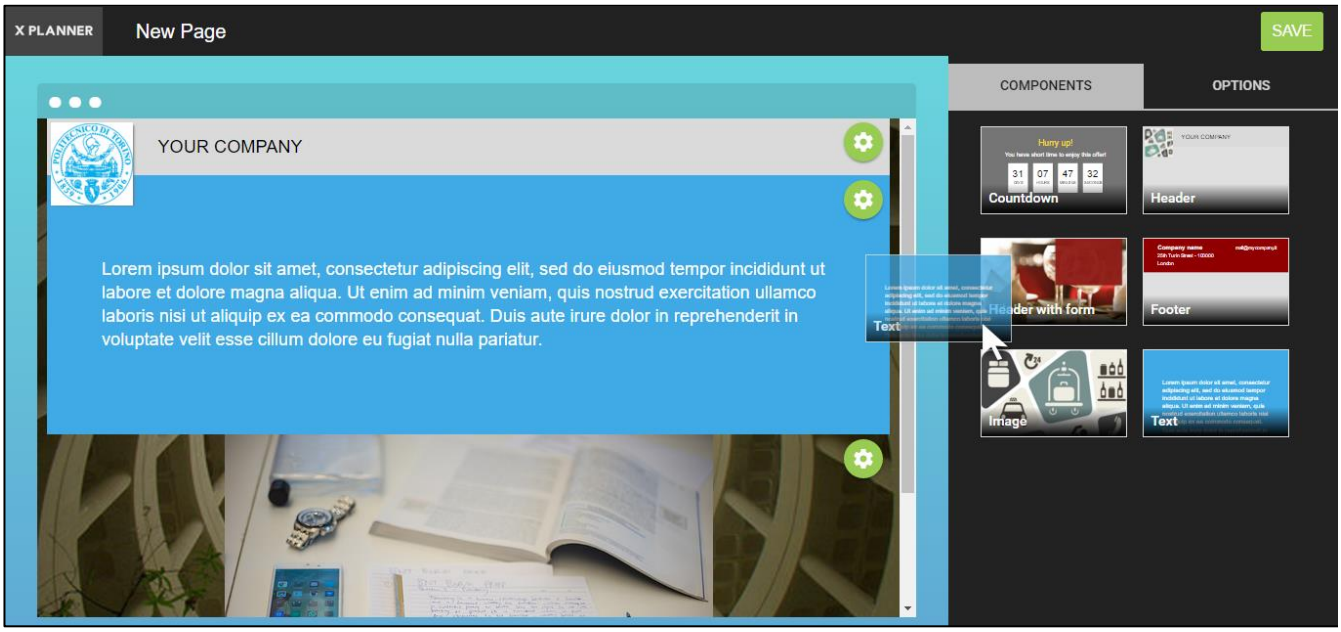

Figure 4. The GUI of the page generator.

background. This complex component – and all option components we defined – exposes a *value* property that contains the current selection, exactly like native input elements. We valorize the component with the current value of the property, and display its *label*.

The value of the *value* property of the input element changes upon user interaction. Consequently, we overwrite the corresponding property on the source element: this change propagates to the component Shadow DOM and to its attributes in the HTML, thanks to the *reflectToAttribute* feature. In this way, the HTML node always contains in its attributes the current state of the component.

Form components have a special behavior. They have been designed as containers of an array of <input-field> components, each of them exposing as customizable properties the label, the placeholder, the type (i.e., text, date, mail) and the requirement. The user can modify these options and add, move or remove the input field.

IV. THE PAGE GENERATOR

We currently use the described approach in xPlanner, a beta Web app for promotional campaign management for Seat PagineGialle. A succint demo of the landing generator is available at goo.gl/LW3WGE.

A. Application overview

We propose a classic drag and drop Graphic User Interface (GUI), visible in Figure 4. On the right column, we show a gallery of modules, which are Web Components. When the user accedes to the tool for the first time, the left side appears blank: the user can drop his\her favorite component on it in the place they prefer. Once on the drop area, the component shows the settings button on top left corner.

Pressing the button, the "settingRequested" event is dispatched and the right bar shows the available options for the active component, in the appearance of the input elements described in Section 2.B (Figure 5). Every edit on those options will reflect on source components.

User can than continue to add components and modify them until it is satisfied of the result. By clicking on the "Save" button, the components are inserted in a full HTML skeleton and the final HTML is exported and stored on the server.

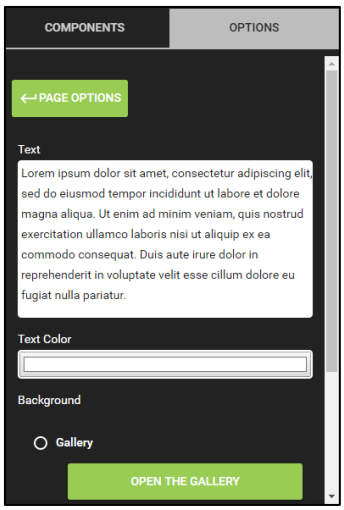

Figure 5. The right column visualize the customizable options.

```
<html>
<head><!-- dependencies loading --></head>
<body>
  <!-- other components -->
  <descriptive-content text-color="#40AAE6"
      text="Lorem ipsum dolor...">
  </descriptive-content>
  <!-- other components -->
</body>
<html>
```

Figure 6. Example of an exported page.

We reported a common result in Figure 6. As shown, the appearance of components in this HTML has not changed: the tag continues to appear without its inner template (once again hidden in the Shadow DOM) and the attributes reveal the values of properties as the user set them.

B. *Application flow*

Figure 7 shows the complete flow of the application. The generator is in charge of importing the components that the user can add to the page. Reading the exposed options from component description itself, the generator makes possible their manipulation, together with an eventual custom sorting. At the end of the process, it generates the HTML file. When it comes to the browser, the needed components are imported and the page is rendered.

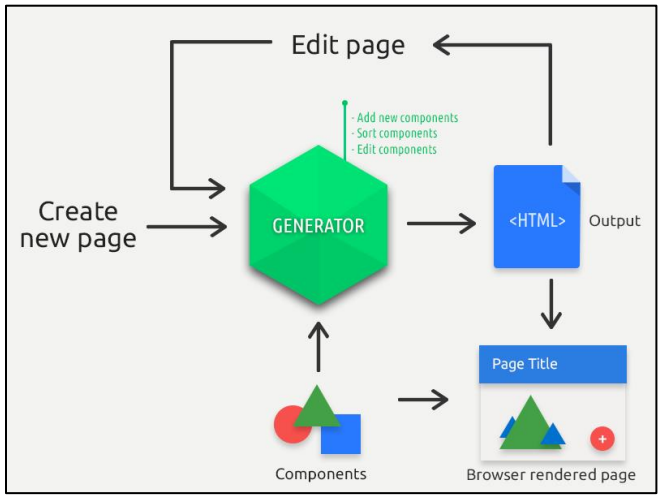

Figure 7. Scheme of the generator application.

The HTML itself is ready for further editing: when the user wants to edit again his/her page, the generator simply retrieves it from server, extracts the content from the body and insert it in the drop area of the GUI. The process continues in the same way.

The application has completely been developed using Web Components and the Polymer technology. In this case, the intrinsic isolation of each Web Component implied some implementation issue. It happens specifically when two components need to communicate each other and are not direct siblings in the DOM, i.e., the option input and the component that input refers: the isolation forced the development to propagate each information up to the nearest common parent of both, and that means to declare a data binding in the definition of multiple components. It seems useful to make an exploration about improving this approach and on the possibility of make use of alternative frameworks for Web application development on top of the components.

C. *Evaluation of the usability*

We introduced the complete xPlanner beta application – site generator included – to a small panel of webmasters and sales agents of SeatPG. The former have a quite good background on Web design and development, while the latter have commercial skills. The feedback has been different between the two profiles.

People with computer experience, and in particular in this type of tools, gave a good evaluation, because it fulfilled their expectation based on their own passed experience. On the other hand, the generator was considered complicated by people with low technical skills. In particular, they were lost in the large number of available options of each component. We are considering a different UI approach and the creation of a light version of the application.

V. CONCLUSION AND FUTURE WORKS

We built a Web page generator that works by combining and customizing Web Components. The feature of the standard grants native isolation to each component for CSS and Javascript, avoiding conflicts in namespace. The generated HTML is the standalone structure that can be used for viewing the page and that can be imported in the generator for further editing. Therefore, the final HTML describes the page perfectly, in a suitable way for both the browser and the generator application. No other structure needs to be stored.

For the first time, the standard of Web Components have been used in a novel way: the combination and manipulation of each component is no longer in charge to the developer through the code, but it is the final user itself to have the ability of doing this through a specific User Interface, in a context of end-user programming.

The page edit and assembly is managed client-side. Components are modular: they can be defined and edited by simply relying on the existing standard and are independent of any other client or server-side technology. The HTML generates and describe itself, following its own rules instead of backend logics. This allows components to correctly behave in complex scenarios.

In order to further improve the approach, we intend to give users with a minimum of Web development skills the

 ISBN: 978-1-61208-468-8

possibility to add their own components. The idea is to design a collaborative platform for SeatPG webmasters, which are constantly in touch with the sales force, in order to make them autonomous in creating suitable modules that are specific to a business category. Each webmaster should be able to design a component, defining the HTML, the CSS and the customizable properties, and share it to the internal community in such a collaborative way.

Other improvements could involve the support to components with external dependencies (by using a dependency manager, e.g., npm or bower). For limiting server requests, we will add a process of concatenation of used components inside the exported HTML. This process, in the Polymer naming convention, is called Vulcanization [15]. We are also working on the definition of a color theme for the whole page, using the CSS variables.

REFERENCES

[1] Y. Jiang, and H. Dong, "TEB: A Template-Based E-commence Website Builder for SMEs," Second International Conference on Future Generation Communication and Networking Symposia, 2008 (FGCNS'08) IEEE, Vol. 1, Dec. 2008, pp. 23-28.

[2] A. K. Kalou, D. A. Koutsomitropoulos, G. D. Solomou, "CMSs, Linked Data and Semantics: A Linked Data Mashup over Drupal for Personalized Search," Metadata and Semantics Research, Springer International Publishing, 2013, pp. 48-59.

[3] M. Krug, and M. Gaedke, "SmartComposition: Enhanced Web Components for a Better Future of Web Development," Proceedings of the 24th International Conference on World Wide Web Companion (WWW'15 Companion) ACM, May 2015, pp. 207-210.

[4] T. Savage, "Componentizing the web," Communications of the ACM, vol. 58, no. 11, pp. 55-61, Nov. 2015.

[5] K. Nakano, Z. Zhenjian, M. Takeichi, "Consistent Web site updating based on bidirectional transformation," International journal on software tools for technology transfer, vol. 11, no. 6, pp. 453-468, Dec. 2009.

[6] D. Denicola, "Custom Elements," W3C Editor's Draft, Mar. 2016, [Online]. Available from: http://w3c.github.io/webcomponents/spec/custom/ 2016.03.09

[7] D. Glazkov, H. Morrita, "HTML Imports," W3C Editor's Draft, Mar. 2016. [Online]. A2016.03.09vailable from: http://w3c.github.io/webcomponents/spec/imports/ 2016.03.09

[8] S. Pieters, A. van Kesteren, et al., "HTML 5.1," W3C Editor's Draft, Mar. 2016. [Online]. Available from: https://html.spec.whatwg.org/multipage/scripting.html#the-template-element 2016.03.09

[9] D. Glazkov, H. Ito, "Shadow DOM," W3C Editor's Draft, Mar. 2016. [Online]. Available from: http://w3c.github.io/webcomponents/spec/shadow/ 2016.03.09

[10] P. De Ryck, N. Nikiforakis, L. Desmet, F. Piessens, W. Joosen, "Protected Web Components: Hiding Sensitive Information in the Shadows," IT Professional, vol. 17, no. 1, pp. 36-43, Jan.-Feb. 2015

[11] W. He, D. Akhawe, S. Jain, E. Shi, D. Song, "ShadowCrypt: Encrypted Web Applications for Everyone," Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS 2014) ACM, 2014, pp. 1028-1039.

[12] WebComponents.org contributors, "WebComponents Polyfills," WebComponents.org Blog, 2015. [Online]. Available from: http://webcomponents.org/polyfills/ 2016.03.09

[13] Google, "Polymer Project," Website. [Online]. Available from: https://www.polymer-project.org 2016.03.09

[14] T. Atkins Jr., "CSS Custom Properties for Cascading Variables Module Level 1," W3C Candidate Recommendation, Dec. 2015. [Online]. Available from: http://www.w3.org/TR/css-variables-1/ 2016.03.09

[15] A. Osmani, "Concatenating Web Components with Vulcanize," Polymer Website, Dec. 2013. [Online]. Available from: http://polymer-project.org/articles/concatenating-web-components.html 2016.03.09

 ISBN: 978-1-61208-468-8