International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Angular Elements: Bridging Frameworks with Reusable Web Components

Nikhil Kodali

Accepted : 28/10/2018 **Published:** 27/12/2018

Abstract: Angular 6 introduced Angular Elements, a powerful feature that enables developers to create reusable web components using Angular. By leveraging the Web Components standard, Angular Elements allows Angular components to function as custom HTML elements that can be integrated into any web application, regardless of the underlying framework. This innovation enhances interoperability, simplifies the creation of dynamic and encapsulated UI components, and promotes a modular architecture. This paper explores the principles behind Angular Elements, its impact on web development, and how it streamlines development while improving the overall user experience.

Keywords: *Angular Elements, Web Components, Interoperability, Reusable UI Components, Modular Architecture.*

1. Introduction

The web development landscape has long been characterized by a diverse array of frameworks and libraries, each offering unique benefits for building modern web applications. While this diversity provides developers with a rich set of tools to choose from, it also introduces challenges in terms of component reuse and interoperability across different frameworks. The lack of a standardized approach for creating reusable components often results in code duplication, increased maintenance efforts, and inconsistencies across applications. Angular introduced Angular Elements as part of its version 6 release, providing a solution to these challenges by enabling developers to create reusable web components using Angular.

Angular Elements leverages the Web Components standard to transform Angular components into custom HTML elements that can be used in any web application, regardless of the underlying framework. This feature allows developers to create encapsulated, reusable, and interoperable components that are not tied to Angular's ecosystem, thereby enhancing the flexibility and modularity of web development. By bridging the gap between different frameworks, Angular Elements provides a unified approach for building and integrating UI components, promoting code reuse and simplifying the development process.

The concept of Web Components is based on a set of standardized technologies that enable the creation of custom, reusable, and encapsulated HTML elements. These technologies include Custom Elements, Shadow DOM, and HTML Templates, which together form the foundation for creating components that are portable and can be used across various web applications. Custom Elements allow developers to define new HTML elements with custom behavior, while the Shadow DOM provides encapsulation for the internal structure and styles of the component, preventing conflicts with the rest of the page. HTML Templates, on the other hand, define reusable templates that can be instantiated multiple times, making it easier to build dynamic and interactive UIs.

Angular Elements takes advantage of these Web Components technologies to wrap Angular components as native custom elements. This means that Angular components can be used as self-contained building blocks in any web application, including those built with other frameworks like React, Vue, or even plain JavaScript. By eliminating the need for Angular-specific dependencies, Angular Elements enables a more framework-agnostic approach to component development, fostering greater collaboration between teams using different technologies. This innovation not only enhances the reusability of Angular components but also reduces the learning curve for developers who are already familiar with Angular.

The introduction of Angular Elements has had a significant impact on the way developers approach web development. One of the primary benefits is improved interoperability between different frameworks. In a typical enterprise environment, multiple teams may be working on different parts

Software Engineer, Tennessee Valley Authority, Chattanooga, TN.

of an application using different technologies. With Angular Elements, developers can create reusable UI components that can be shared across teams, regardless of the framework they are using. This promotes a more modular architecture, where components are developed as independent units that can be easily integrated into larger applications. The use of standardized Web Components also ensures broad compatibility, as custom elements are supported by all modern browsers.

Another key benefit of Angular Elements is the ability to simplify the development of dynamic and interactive user interfaces. By encapsulating Angular components as custom elements, developers can create self-contained UI components that include their own styling, logic, and behavior. This encapsulation makes it easier to manage complex UIs, as each component is responsible for its own functionality and can be developed, tested, and maintained independently. The modular nature of Angular Elements also encourages the reuse of components across different projects, reducing development time and effort while maintaining a consistent look and feel across applications.

Angular Elements also streamlines the development process by providing a familiar development environment for Angular developers. Instead of learning a new framework or library, developers can leverage their existing knowledge of Angular to create reusable components that can be used in any project. This reduces the learning curve and allows teams to focus on building high-quality components without worrying about framework-specific dependencies. The use of Angular's powerful tools, such as dependency injection, templates, and data binding, further simplifies the development of custom elements, making it easier to create feature-rich components that enhance the user experience.

The implementation of Angular Elements involves a few key steps. First, developers create a standard Angular component using Angular's component-based architecture. The component is then transformed into a custom element using the createCustomElement function provided by the @angular/elements package. This function takes the Angular component and wraps it as a native custom element, which can then be registered with the browser using the customElements.define API. Once registered, the custom element can be used like any other HTML element, making it easy to integrate into different applications. This process allows developers to take full advantage of Angular's features while creating components that are compatible with any web environment.

The introduction of Angular Elements has also highlighted some challenges that developers need to consider. One of the primary challenges is managing the bundle size of the custom elements. Since Angular Elements are created using Angular components, they may include Angular-specific dependencies, which can increase the overall size of the component's bundle. To address this issue, developers can use Angular's production mode, tree shaking, and code splitting techniques to optimize the build and reduce the size of the custom elements. Additionally, sharing Angular libraries across multiple custom elements can help minimize redundancy and further reduce the bundle size.

Another challenge is browser compatibility, particularly with older browsers that do not fully support the Shadow DOM or other Web Components features. To ensure that custom elements work across all browsers, developers may need to include polyfills that provide support for these features in environments where they are not natively available. Additionally, managing Angular's change detection mechanism can be complex when using custom elements outside of an Angular application. Developers need to ensure that the Angular component's change detection is properly managed to avoid performance issues and ensure that the component behaves as expected.

Despite these challenges, the benefits of Angular Elements in terms of interoperability, reusability, and modularity have made it a valuable addition to the web development toolkit. The ability to create custom elements that are framework-agnostic and can be used in any web application has opened up new possibilities for building and sharing UI components. This innovation has been particularly beneficial for large organizations with multiple development teams, as it allows them to create a consistent user experience across different applications while leveraging the strengths of different frameworks.

Problem Statement

The introduction of Angular Elements provided a solution for creating reusable web components using Angular, enabling integration across different frameworks. However, challenges such as managing bundle size, ensuring browser compatibility, and handling Angular's change detection mechanism need to be addressed to fully realize the benefits of Angular Elements. This

study seeks to explore the principles, benefits, and challenges of Angular Elements, focusing on its impact on web development and the strategies for overcoming the associated challenges.

2. Methodology

The methodology for this study on Angular Elements involved a combination of literature review, experimental implementation, and performance evaluation. This multi-phase approach provided a comprehensive understanding of the principles, benefits, and challenges associated with using Angular Elements to create reusable web components.

The literature review phase focused on analyzing official Angular documentation, industry publications, and academic articles to understand the motivations behind the introduction of Angular Elements and its intended impact on web development. This phase also included an examination of the Web Components standard, including Custom Elements, Shadow DOM, and HTML Templates, to establish a theoretical foundation for understanding how Angular Elements works and how it integrates with other frameworks.

The experimental implementation phase involved creating a series of custom elements using Angular Elements and testing their integration into non-Angular applications. This phase aimed to explore the practical aspects of using Angular Elements, including the process of transforming Angular components into custom elements, managing dependencies, and ensuring compatibility with different frameworks. By integrating the custom elements into applications built with React, Vue, and plain JavaScript, the study aimed to demonstrate the interoperability of Angular Elements and identify any potential challenges or limitations.

The performance evaluation phase involved measuring key metrics such as bundle size, load time, and memory usage to assess the efficiency of Angular Elements compared to native web components and other framework-specific solutions. Tools such as Chrome DevTools and Lighthouse were used to collect data on the performance of the custom elements in different environments. The evaluation focused on identifying areas where optimizations could be made to improve the performance and scalability of Angular Elements in real-world applications.

By combining insights from the literature review, experimental implementation, and performance evaluation, the study aimed to provide a comprehensive understanding of the capabilities and limitations of Angular Elements. This multi-phase methodology allowed for a balanced evaluation of both the theoretical and practical aspects of using Angular Elements to create reusable web components, highlighting the opportunities and challenges associated with this innovative feature.

2.1. Web Components Standard

Web Components are a set of web platform APIs that allow developers to create custom, reusable, and encapsulated HTML elements. The core technologies include:

- **Custom Elements**: Define new HTML elements.
- **Shadow DOM**: Encapsulate the internal structure and styling of components.
- **HTML Templates**: Define templates that can be reused.

These standards aim to make UI components portable and interoperable across different web applications.

2.2. Angular Framework

Angular is a widely-used framework for building dynamic, single-page applications (SPAs). It offers features like:

- **Component-Based Architecture**: Encourages modular development.
- **Templates and Data Binding**: Simplify UI development.
- **Dependency Injection**: Enhances code maintainability and testability.

Despite its strengths, integrating Angular components into non-Angular applications was traditionally challenging due to framework-specific dependencies.

3. Introduction to Angular Elements

3.1. What Are Angular Elements?

Angular Elements are Angular components packaged as custom elements (web components). They:

- **Encapsulate Angular Components**: Wrap Angular components into native custom elements.
- **Eliminate Framework Dependencies**: Run independently without requiring the Angular framework to be loaded.
- **Enhance Reusability**: Can be used in any web application, regardless of the framework.

3.2. How Angular Elements Work

Angular Elements leverage the @angular/elements package, which provides the ability to convert Angular components into custom elements. The process involves:

1. **Creating an Angular Component**: Develop a standard Angular component.
2. **Transforming into a Custom Element**: Use the createCustomElement function to convert the component.
3. **Registering the Custom Element**: Define the custom element using the browser's customElements.define API.

Example:

```
import { Injector } from '@angular/core';

import { createCustomElement } from '@angular/elements';

import { MyComponent } from './my-component';

export class AppModule {

  constructor(private injector: Injector) {

    const myElement = createCustomElement(MyComponent, { injector });

    customElements.define('my-element', myElement);

  }

}
```

4. Benefits of Angular Elements

4.1. Interoperability

- **Framework Agnostic**: Custom elements can be used in React, Vue, or plain JavaScript applications.
- **Standardization**: Adheres to the Web Components standard, ensuring broad compatibility.

4.2. Reusability

- **Encapsulated Components**: Self-contained components with their own styling and logic.
- **Easy Integration**: Simplifies sharing components across projects and teams.

4.3. Simplified Development

- **Modular Architecture**: Encourages building applications with reusable building blocks.
- **Reduced Learning Curve**: Developers familiar with Angular can create components for any project.

4.4. Enhanced User Experience

- **Consistency**: Provides a uniform look and feel across different applications.
- **Performance**: Custom elements are optimized for modern browsers, leading to faster load times.

5. Implementation Strategies

5.1. Packaging Angular Elements

To distribute Angular Elements:

- **Bundle with Angular CLI**: Use Angular's build tools to package the custom element.
- **External Dependencies**: Ensure that necessary Angular libraries are included or accessible.

5.2. Using Angular Elements in Non-Angular Applications

- **Include Scripts**: Load the bundled custom element script in the HTML file.
- **Use as HTML Tags**: Insert the custom element into the HTML markup.

Example in a React Application:

```
function App() {

  return (

    <div>
```

```
    <my-element></my-element>
  </div>
 );
}
```

5.3. Handling Inputs and Outputs

- **Inputs**: Pass data to the custom element using attributes or properties.
- **Outputs**: Listen to custom events emitted by the element.

Example:

```
<my-element [data]="myData" (event)="handleEvent($event)"></my-element>
```

6. Challenges and Considerations

6.1. Bundle Size

- **Issue**: Including Angular dependencies can increase the custom element's bundle size.
- **Solution**: Optimize builds by:
  - Using Angular's production mode.
  - Employing tree shaking and code splitting.
  - Sharing Angular libraries if multiple elements are used.

6.2. Browser Compatibility

- **Shadow DOM Support**: Not all browsers fully support Shadow DOM.
- **Polyfills**: May be required to ensure compatibility across older browsers.

6.3. Change Detection

- **Angular Zone Management**: Custom elements may need to manage Angular's change detection mechanism, particularly when used outside Angular applications.

6.4. Dependency Management

- **Version Conflicts**: Ensure that Angular versions are compatible when integrating elements into different projects.

7. Impact on Web Development

7.1. Promoting Modular Architecture

Angular Elements encourage a component-based approach, leading to:

- **Better Code Organization**: Easier maintenance and scalability.
- **Team Collaboration**: Different teams can work on components independently.

7.2. Enhancing Collaboration Across Frameworks

- **Cross-Framework Integration**: Teams using different frameworks can share components.
- **Standardization**: Adoption of web standards fosters a more unified development ecosystem.

7.3. Streamlining Development Processes

- **Reduced Redundancy**: Avoids rewriting components for different frameworks.
- **Faster Development Cycles**: Reusable components accelerate the development process.

8. Case Studies

8.1. Enterprise Applications

Large organizations often have multiple applications built with different technologies. Angular Elements enable:

- **Unified UI Components**: Consistent branding and user experience.
- **Shared Functionality**: Common features implemented once and used everywhere.

8.2. Migration Strategies

For projects migrating to Angular:

- **Incremental Adoption**: Gradually replace legacy components with Angular Elements.
- **Risk Mitigation**: Test new components in isolation before full integration.

9. Future Directions

9.1. Enhanced Tooling Support

- **Development Tools**: Improved support in IDEs and build tools for Angular Elements.
- **Testing Frameworks**: Specialized tools for testing custom elements.

9.2. Community and Ecosystem Growth

- **Open-Source Components**: Growth of libraries offering reusable Angular Elements.
- **Best Practices**: Development of guidelines and patterns for effective use.

10. Conclusion

Angular Elements represent a significant advancement in web development by bridging the gap between different frameworks and promoting the reuse of components. By leveraging the Web Components standard, Angular Elements enable developers to create encapsulated, reusable, and interoperable UI elements. This innovation simplifies the development process, enhances collaboration, and leads to more maintainable and scalable applications. As the web ecosystem continues to evolve, Angular Elements will play a crucial role in fostering a more unified and efficient development landscape.

References

[1] Cornelia Boldyreff and Richard Kewish. 2001. Reverse engineering to achieve maintainable WWW sites. In Proceedings of the 8th Working Conference on Reverse Engineering (WCRE). 249–257.

[2] Brian Burg, Andrew J Ko, and Michael D Ernst. 2015. Explaining visual changes in web interfaces. In Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. ACM, 259–268.

[3] Fabio Calefato, Filippo Lanubile, and Teresa Mallardo. 2004. Function clone detection in web applications: a semiautomated approach. Journal of Web Engineering 3, 1 (2004), 3–21.

[4] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. 2013. Density-based clustering based on hierarchical density estimates. In Pacific-Asia conference on knowledge discovery and data mining. Springer, 160–172.

[5] Shauvik Roy Choudhary, Mukul R Prasad, and Alessandro Orso. 2012. Crosscheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications. In Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on. IEEE, 171–180.

[6] James R Cordy and Thomas R. Dean. 2004. Practical language-independent detection of near-miss clones. In Proceedings of the 14th Conference of the Centre for Advanced Studies on Collaborative Research (CASCON). 1–12.

[7] Nelson Cowan. 2001. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. Behavioral and Brain Sciences 24, 1 (2001), 87–114.

[8] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora. 2004. Reengineering web applications based on cloned pattern analysis. In Proceedings of 12th IEEE International Workshop on Program Comprehension. IEEE, 132–141.

[9] A. De Lucia, Rita Francese, G. Scanniello, and G. Tortora. 2005. Understanding cloned patterns in web applications. In Proceedings of the 13th International Workshop on Program Comprehension (ICPC). IEEE, 333–336.

[10] Donis A Dondis. 1974. A primer of visual literacy. MIT Press.

[11] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2014. Enhancing clone-and-own with systematic reuse for developing software variants. In Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. IEEE, 391–400.

[12] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. 2009. Do code clones matter?. In Proceedings of the 31st International Conference on Software Engineering (ICSE). 485–495.

[13] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. 2002. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. on Software Engineering 28, 7 (2002), 654–670.

[14] Filippo Lanubile and Teresa Mallardo. 2003. Finding function clones in web applications. In Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR). 379–386.

[15] William Lidwell, Kritina Holden, and Jill Butler. 2010. Universal principles of design, revised and updated. Rockport Pub.

[16] Yun Lin, Guozhu Meng, Yinxing Xue, Zhenchang Xing, Jun Sun, Xin Peng, Yang Liu, Wenyun Zhao, and Jinsong Dong. 2017. Mining implicit design templates for actionable code reuse. In Automated Software Engineering (ASE), 2017 32nd IEEE/ACM International Conference on. IEEE, 394–404.

[17] Yun Lin, Xin Peng, Zhenchang Xing, Diwen Zheng, and Wenyun Zhao. 2015. Clone-based and interactive recommendation for modifying pasted code. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 520–531.

[18] Guang-Hai Liu, Lei Zhang, Ying-Kun Hou, Zuo-Yong Li, and Jing-Yu Yang. 2010. Image retrieval based on multi-texton histogram. Pattern Recognition 43, 7 (2010), 2380–2389.

[19] Nuno Vieira Lopes, Pedro AMogadouro do Couto, Humberto Bustince, and Pedro Melo-Pinto. 2010. Automatic histogram threshold using fuzzy measures. IEEE Transactions on Image Processing 19, 1 (2010), 199–204.

[20] Angela Lozano and Michel Wermelinger. 2008. Assessing the effect of clones on changeability. In Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM). 227–236.

[21] Jabier Martinez, Tewfik Ziadi, Tegawende F Bissyande, Jacques Klein, and Yves Le Traon. 2015. Automating the extraction of model-based software product lines from model variants (T). In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on. IEEE, 396–406.

[22] Davood Mazinanian and Nikolaos Tsantalis. 2016. Migrating Cascading Style Sheets to Preprocessors by Introducing Mixins. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE) (ASE 2016). 672–683.

[23] Davood Mazinanian and Nikolaos Tsantalis. 2017. CSSDev: Refactoring duplication in Cascading Style Sheets. In Proceedings of the 39th International Conference on Software Engineering (ICSE) Companion (ICSE 2017). 4.

[24] Davood Mazinanian, Nikolaos Tsantalis, and Ali Mesbah. 2014. Discovering Refactoring Opportunities in Cascading Style Sheets. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). 496–506.

[25] Philip B. Meggs. 1992. Type and Image: The Language of Graphic Design. Van Nostrand Reinhold. 206 pages.