ReactCOP: Modular and Scalable Web Development with Context-Oriented Programming

David H. Lorenz
lorenz@openu.ac.il
Dept. of Mathematics and Computer Science
Open University of Israel
Ra'anana 4353701, Israel

Ofir Shmuel
openu@ofirshmuel.com
Dept. of Mathematics and Computer Science
Open University of Israel
Ra'anana 4353701, Israel

Abstract

We present a library named ReactCOP that extends React's capabilities with support for *Context-Oriented Programming*. The library lets developers manage behavioral variations in React applications through layers, and adapt the application's behavior dynamically based on different contexts.

CCS Concepts: • **Software and its engineering → Abstraction, modeling and modularity**; *Hypertext languages.*

Keywords: Context-oriented programming (COP), React.

ACM Reference Format:
David H. Lorenz and Ofir Shmuel. 2023. ReactCOP: Modular and Scalable Web Development with Context-Oriented Programming. In *Companion Proceedings of the 2023 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '23), October 22–27, 2023, Cascais, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3618305.3623609

1 Introduction

React[1] is an open-source JavaScript [1] library for building user interfaces. It was released by Facebook in 2013, and quickly gained popularity in the web development community due to its performance, reusability, and component-based architecture. React utilizes a virtual, lightweight representation of the DOM in memory, enabling efficient update and rendering of UI components.

Context-oriented Programming (COP) [3] is a programming paradigm that focuses on managing context-dependent behavioral variation in software systems. It provides mechanisms to dynamically change the behavior of a program based on the current context. With COP, developers can define and activate different sets of behaviors, grouped in *layers*, depending on the specific situation or environment, called *context*, in which the program is running.

In this work we present a library named ReactCOP for handling context-dependent behavioral variations in React applications by providing support for COP in React.

[1] https://react.dev/

SPLASH Companion '23, October 22–27, 2023, Cascais, Portugal

ACM ISBN 979-8-4007-0384-3/23/10.
https://doi.org/10.1145/3618305.3623609

2 Limitations of React

React uses a single-rooted component tree with unidirectional data flow, in which a child component acquires data from its containing parent component [2]. This makes it easier to understand and reason about the state and behavior of the application.

React's default approach to handling dependencies between components is through *props* (short for properties). Props are used to pass data from a parent component to its child components. Props are immutable and cannot be modified by the child components. Instead, *State* is used to manage component-specific data that can change over time. Unlike props, state is mutable and can be updated.

However, propagating the component state via *props* is done manually. This leads to unintentional tight coupling between parent and descendant components, making it challenging to enforce proper modularity. The React Context API and libraries like Redux can help, but they are not easy to use and do not support the division of code into different behavioral variations.

In React developers tend to concentrate a lot of code in one component file, which includes several different behavioral variations of the component. When a React component contains too many behavioral variations, its code becomes complex and difficult to follow. The logic may become convoluted, making it harder for the developer, as well as other developers, to comprehend the code's purpose and functionality. Existing COP solutions for JavaScript, like ContextJS [4], are intended for JavaScript objects but not for React components.

3 ReactCOP

The ReactCOP library lets developers manage behavioral variations in React and adapt the application's behavior dynamically based on different contexts.

Layers. In ReactCOP, the process of creating layers is straightforward. A new layer is generated by invoking the

Listing 1. Rendering context-dependant data in React (List. 1a) and in ReactCOP (List. 1b)

```
let label, player;
if (isGameover) {
  label = "Winner: ";
  player = xIsNext ? "O" : "X";
} else {
  label = "Next player: ";
  player = xIsNext ? "X" : "O";
}

return (
 <div className="status">
  <span className={isGameover ? "green-text" : "orange-text"}>

    {label}
  </span>
    {player}
 </div>
);
```

(a) React

```
return (
 <div className="status">
  <Gameover.Layer>
   <span className="green-text">Winner: </span>
   <XTurn.Layer>O</XTurn.Layer>
   <OTurn.Layer>X</OTurn.Layer>
  </Gameover.Layer>
  <Move.Layer>
   <span className="orange-text">Next player: </span>
   <XTurn.Layer>X</XTurn.Layer>
   <OTurn.Layer>O</OTurn.Layer>
  </Move.Layer>
 </div>
);
```

(b) ReactCOP

`createLayer()` function. Once the layer is created, it becomes possible to attach behavioral variations to it, enabling the customization of specific behaviors within the application. For example, List. 1 displays the differences between rendering context-dependant data in React (List. 1a) and the corresponding code with the same functionality using layers in ReactCOP (List. 1b).

Activation. In ReactCOP, the activation or deactivation of a layer can be dynamically controlled. This can be determined based on the current context, a local variable within the application, or even an external API. The flexibility of ReactCOP allows for the dynamic management of layers based on various runtime conditions and external factors.

Behavioral Variations. In ReactCOP, the developer has the ability to transform nearly every aspect of the components into a behavioral variation. This includes controlling the rendering of child components, modifying style and other props, and even adapting in-component functions. The process of converting these aspects into behavioral variations requires only a small amount of code and leverages commonly used design patterns in React.

Scoping. By leveraging the React component tree, ReactCOP enables the activation and deactivation of layers, facilitating the concept of scoping. This means that multiple layers can be activated simultaneously within the same scope, and the same layer can be activated in one scope while deactivated in another. As a result, a component can exhibit different behaviors in different scopes, allowing for contextual adaptation and providing enhanced flexibility in rendering and functionality.

4 Conclusion

ReactCOP is a library that significantly expands React's capabilities by incorporating support for COP within React applications. By harnessing the power of COP, ReactCOP provides developers with the means to dynamically manage and adapt behavioral variations within their applications based on varying contexts. This addresses unintentional coupling between components, complex dependency management, and scalability issues that can arise within React's component tree.

Through the creation and activation of layers, ReactCOP empowers developers to easily customize behaviors, modify styles and props, and adapt component functions, all while maintaining the integrity of commonly used React design patterns. The library's approach to scoping enables the activation and deactivation of multiple layers within distinct scopes, facilitating contextual adaptation and rendering flexibility. Furthermore, ReactCOP's advanced layer concept expands the possibilities by introducing complement, intersecting, and union sets, empowering developers to intuitively manipulate behavioral variations.

Overall, ReactCOP offers a comprehensive solution to managing context-dependent behavioral variations in React applications, enhancing the modularity, adaptability, and scalability of React-based projects.

References

[1] David Flanagan. 2011. *JavaScript: the definitive guide* (6th ed.). O'Reilly Media, Inc., Sebastopol, CA.

[2] Joseph Gil and David H. Lorenz. 1996. Environmental Acquisition – A New Inheritance-Like Abstraction Mechanism. In *Proceedings of the 11th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'96)*. ACM SIGPLAN Notices 31(10) Oct. 1996, San Jose, California, 214–231. https://doi.org/10.1145/236338.236358

[3] Robert Hirschfeld, Pascal Costanza, and Oscar Marius Nierstrasz. 2008. Context-oriented programming. *Journal of Object Technology* 7, 3 (March 2008), 125–151. https://doi.org/10.5381/jot.2008.7.3.a4

[4] Jens Lincke, Malte Appeltauer, Bastian Steinert, and Robert Hirschfeld. 2011. An open implementation for context-oriented layer composition in ContextJS. *Science of Computer Programming* 76, 12 (2011), 1194–1209. https://doi.org/10.1016/j.scico.2010.11.013