JET: Type-Driven Web Framework for Guaranteed Consistency and Fine-Grained Rendering

Jiahao Zhang, Emma Sudo, Trip Master, Keith Winstein

School of Electronics Engineering and Computer Science, Peking University, Department of Computer Science, Stanford University

Motivation

The reactive system has become the mainstream for web development. However, under certain circumstances, it still fails in meeting the demands. In the development of a live execution code editor(Codillon) for an experimental innovative CS course, the reactive system(e.g. Leptos) causes great lag for its course-grained nature to describe the web page because of uncountable signals required by the analysis of the code.

But there will be great difficulty to maintain consistency between our program model and browser's DOM if we get rid of any reactive system. For example, each time after receiving a single user input(key stroke, mouse click, etc.) we have to update the program model state and update the DOM accordingly. But it is not guaranteed that the DOM is consistent with our program model because DOM is held by the browser, a different environment with the business logic.

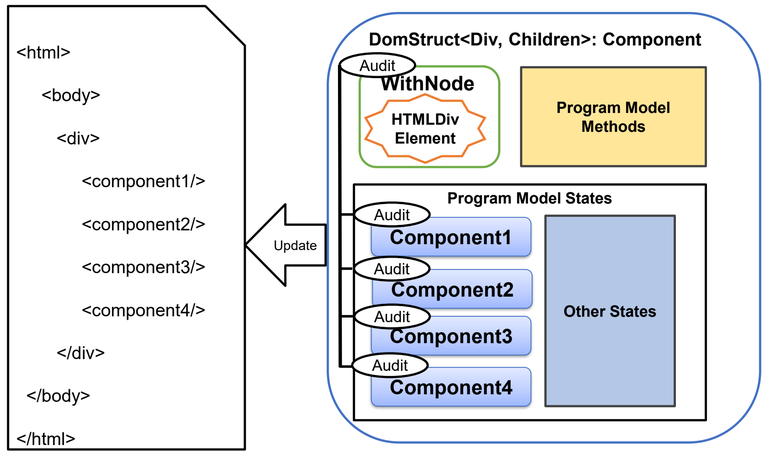
Therefore, we utilize the idea of type system to build our own web framework which not only reaches finer-grained rendering but also ensures consistency of the DOM.

Guaranteed Consistency

Methods used to guarantee consistency:

1. Compile-Time Static Type Check
2. Runtime Top-Down Auditing

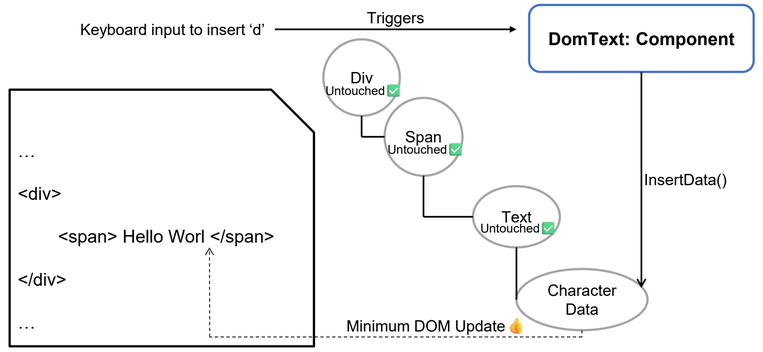
On the right is the example of **Component, DomStruct**. **DomStruct** has different component children to combine different html elements. The children are declared statically as a tuple inside **DomStruct** and **DomStruct**'s audit will call each child's audit.



Fine-Grained Rendering

Compared with any reactive system, **JET** is able to provide finer-grained control of the DOM because it can utilize all API that DOM exposes via **WithNode** trait.

For example, **DomText**, which wraps just one DOM Text Node, can modify the character data partially by calling InsertData API of DOM Text Node, reaching minimum DOM updating cost.

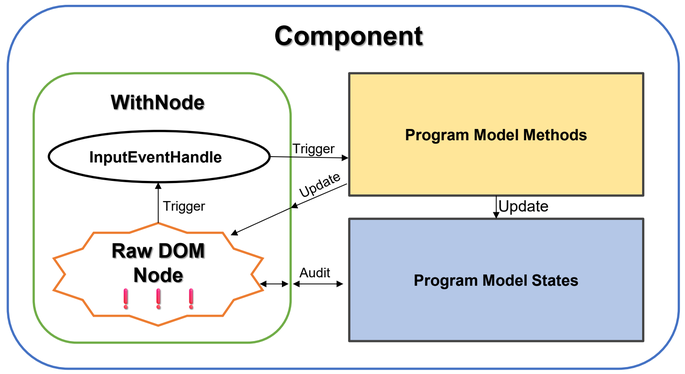


The Editor is one **DomStuct**, and it contains a **DomVec** as one of its children, called "lines". The editor itself will render a Div element while the lines is a vector containing Span Elements as its items. The Editor has a method called "handle_input" which is passed to "OnBeforeInput" handler of the Div element.When the "handle_input" is triggered after any user input, the editor will modify the content of lines to update the DOM. The auditing will run per input to guarantee the rendered text, "OnBeforeInput" handler, etc. are consistent with what we have set in the business logic.

If the program is built in released version, the audit function can be turned off to avoid overhead(though very little).

Method

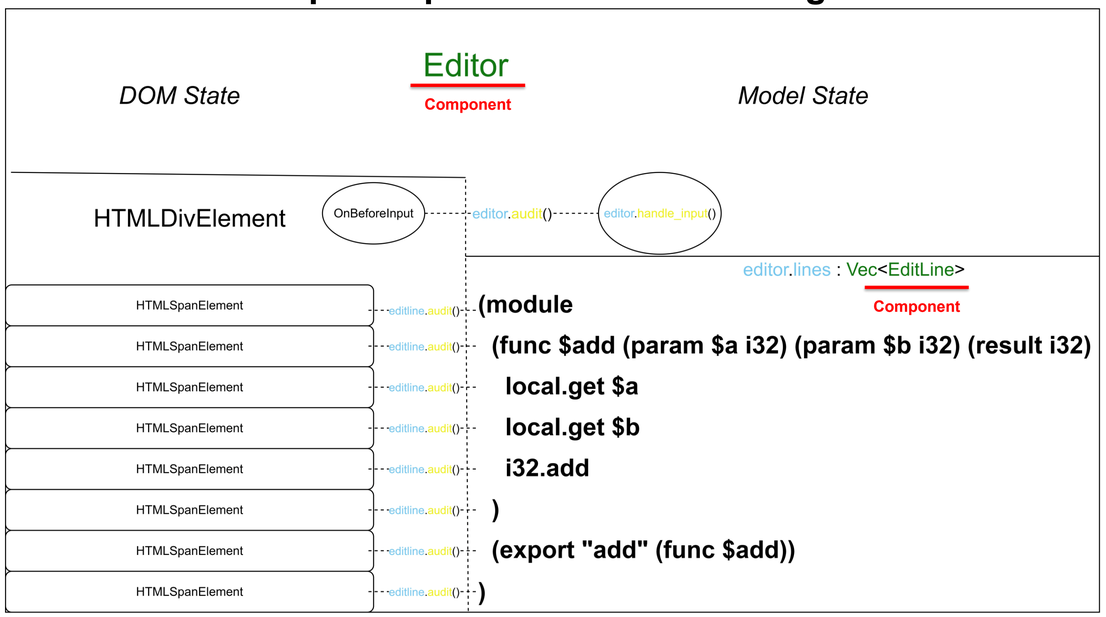
Design of JET



We use a **WithNode** trait as an encapsulation of a raw DOM node and we build **Component** trait on top of **WithNode** to update the DOM node according to program model state and execute auditting. An object which implements **Component** will update its own DOM to allow fine-grained rendering and audit to make sure content in the raw node is consistent with our model's state.

Result

Example: Implement an editor using JET



Discussion

Comparison with reactive systems

The reactive systems use a descriptive way to ensure consistency. To be more accurate, they describe the website with signals embedded html and they update the DOM according to the changes of reactive signals. Therefore, they can have performance issue to manage too many signals and unavoidable extra DOM updates.

Comparison with Web-oriented syntactic constructs

There are other works which use syntactic constructs to ensure the state consistency. Some of them(e.g. Elm) have static syntactic check and then be compiled into JS and some of them(e.g. Ur/Web) will have a formal syntactic design to ensure consistency before runtime.

Limitation and Future Work

Although JET can reach good consistency and rendering in the same time, there are drawbacks due to the dependence on static type system. For example, it will be clumsy to replace one component with another component with a different type in the runtime. Also, as a prototype, JET can only deal with the client side rendering and lack of the ability to have a multi-thread context. So there are many future aspects to be explored, including runtime type flexibility, server side rendering and multi-thread support.

Acknowledgement

Sincere gratitude to Prof. Keith Winstein and my collaborators for this wonderful summer working together. Also thanks to the whole SNR group and CURIS at Stanford for their hospitality. Special thanks to Stanford School of Engineering and the UGVR program for providing me with such a valuable opportunity.

Offical Repo of Codillon

Scan Me