СРАВНЕНИЕ АЛГОРИТМОВ ВЗАИМОДЕЙСТВИЯ КОМПОНЕНТОВ В СУЩЕСТВУЮЩИХ ФРЕЙМВОРКАХ И РАЗРАБОТКА ДВУХСВЯЗНОЙ МОДЕЛИ ВЗАИМОДЕЙСТВИЯ НА СТОРОНЕ КЛИЕНТА

Струкова А. А., Шилин Л. Ю.
кафедры информационных технологий автоматизированных систем,
Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: alinastrukovaa@gmail.com, shilin@bsuir.by

В статье проводится сравнительный анализ алгоритмов взаимодействия компонентов в популярных фреймворках Angular и React. На основе выявленных недостатков предлагается новый алгоритм взаимодействия компонентов на стороне клиента, основанный на двухсвязной модели обмена данными.

Введение

В последние годы наблюдается устойчивый рост популярности компонентных архитектур при создании клиентских приложений. Разделение интерфейса на изолированные модули упрощает разработку, тестирование и поддержку систем, однако при этом усложняется процесс взаимодействия между компонентами и синхронизации данных между ними.

Наиболее известные фреймворки Angular и React решают эту задачу по-разному. Angular использует строгую модель связывания данных и централизованное управление изменениями, в то время как React делает ставку на простоту и оптимизацию через виртуальный DOM.

Обе модели доказали свою эффективность, но обладают определёнными ограничениями: Angular страдает от избыточных проходов по дереву компонентов, а React требует сложных инструментов для синхронизации состояний.

В связи с этим актуальной задачей становится разработка алгоритма взаимодействия компонентов, который объединяет преимущества существующих подходов, минимизируя их недостатки.

Архитектурные принципы

Angular реализует архитектуру MVVM (Model–View–ViewModel), где каждый компонент связан с шаблоном, отражающим текущее состояние данных. Передача информации между компонентами осуществляется через входные параметры, обеспечивающие поток данных сверху вниз, выходные события, формирующие поток снизу вверх, и реактивные потоки Observable библиотеки RxJS.

Каждое изменение состояния инициирует механизм Change Detection, проходящий по дереву компонентов и проверяющий, произошло ли изменение значений, используемых в шаблоне.

Это обеспечивает предсказуемость поведения, однако в крупных приложениях вызывает значительные накладные расходы при рендеринге и синхронизации состояний.

React опирается на концепцию однонаправленного потока данных: родитель передаёт свойства дочерним компонентам, а те реагируют на изменения состояния с помощью хуков (useState, useEffect, useMemo).

Основное отличие React – использование виртуального DOM, который сравнивает текущее и новое состояние дерева компонентов. Рендеринг выполняется только для реально изменившихся элементов:

$$\Delta DOM = DOM_{new} - DOM_{old}$$

Это делает React более производительным, особенно при частых обновлениях интерфейса. Для сложных взаимодействий между множеством компонентов часто требуются дополнительные библиотеки для глобального управления состоянием (Redux, MobX, Zustand), что увеличивает когнитивную сложность системы.

Проблемы существующих подходов

Сравнение архитектур фреймворков Angular и React позволяет выделить следующие ключевые проблемы:

1. Angular: чрезмерная нагрузка на систему обнаружения изменений, особенно при больших деревьях компонентов.
2. React: отсутствие встроенной двухсторонней синхронизации данных между компонентами, необходимость использовать сторонние хранилища.
3. Обе системы: избыточная передача данных по цепочкам компонентов, даже когда изменения касаются локальных областей.

Следовательно, требуется алгоритм, который минимизирует количество перерендеров, сохраняет реактивность при изменениях данных и обеспечивает синхронизацию между зависимыми компонентами без промежуточных прослоек.

Двухсвязная модель взаимодействия компонентов

Предлагаемый алгоритм основан на двухсвязной модели (Bidirectional Component Linking Model, BCLM), в которой каждый компонент может как публиковать изменения своего состояния, так и подписываться на изменения состояния других компонентов напрямую, минуя посредников.

Пусть C_i и C_j – два компонента с локальными состояниями $S_i(t)$ и $S_j(t)$. Связь между ними определяется как двунаправленное отношение:

$$L_{i,j} = (C_i \leftrightarrow C_j)$$

Каждое изменение состояния δS_i автоматически вызывает событие синхронизации:

$$\tau_{i,j}(t) = h(S_i(t))$$

$$S_j(t+1) = f(S_j(t), \tau_{i,j}(t))$$

Функция h определяет, какие именно данные компонента C_i подлежат публикации, а функция f – как эти данные применяются у подписчика. Таким образом, состояние каждого компонента может влиять на другие в обе стороны, формируя адаптивную реактивную сеть.

Этапы работы алгоритма

Алгоритм взаимодействия компонентов на основе двухсвязной модели включает следующие шаги:

1. Инициализация связей: каждый компонент регистрирует свои исходящие и входящие связи $L_{i,j}$;
2. Обнаружение изменений: если $\delta S_i \neq 0$, запускается событие обновления;
3. Передача данных подписчикам:

$$\tau_{i,j} = h(S_i)$$

4. Применение изменений у подписчиков:

$$S_j(t+1) = f(S_j(t), \tau_{i,j})$$

5. Оптимизация рендеринга: обновление интерфейса выполняется только при фактическом изменении состояния:

$$R_j = [S_j(t+1) \neq S_j(t)]$$

В отличие от Angular, система не требует глобального обхода дерева компонентов, поскольку обновления распространяются только между реально связанными узлами. В отличие от React, состояние может обновляться двунаправленно, что исключает необходимость в централизованных стореджах и промежуточных прослойках.

Преимущества двухсвязной модели

Предложенный алгоритм имеет ряд преимуществ. Он предоставляет изоляцию изменений, так как обновления распространяются только на те компоненты, которые связаны логически, без обхода всего дерева. Отсутствие глобального механизма change detection снижает нагрузку на процессор. Также алгоритм снижает когнитивную сложность, так как нет необходимости использовать внешние библиотеки для глобального состояния.

Двухсвязная модель позволяет как прямую, так и обратную передачу данных без нарушения реактивности, это делает алгоритм взаимодействия компонентов на стороне клиента гибким. Алгоритм может быть интегрирован в существующие архитектуры Angular и React как промежуточный слой взаимодействия компонентов, поэтому одним из преимуществ является совместимость разработанного решения с существующими системами на базе существующих фреймворков.

Заключение

Предложенный алгоритм взаимодействия компонентов на стороне клиента, основанный на двухсвязной модели, сочетает в себе преимущества архитектур Angular и React, устраняя их основные недостатки. Данный подход позволяет достичь баланса между структурированностью (характерной для Angular) и производительностью (характерной для React). Он снижает количество ненужных обновлений, повышает согласованность данных и обеспечивает гибкость при проектировании сложных SPA-приложений.

I. Список литературы

1. Грубер И. Ю. Современные JavaScript-фреймворки. – Минск: ITBooks, 2022.
2. Google Developers. *Angular Change Detection Explained.* – 2023.
3. Dan Abramov. *Inside React Fiber: Reconciliation Algorithm.* – React Blog, 2022.
4. Preact Team. *Signals: Fine-Grained Reactivity for the Modern Web.* – 2023.