



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:

Метод динамического отображения изменений
пользовательского интерфейса на основе обработки
изменений XML-файла

Студент группы ИУ7-84Б

(Подпись, дата)

П. А. Егорова

(И.О. Фамилия)

Руководитель ВКР

(Подпись, дата)

Т. А. Никульшина

(И.О. Фамилия)

Нормоконтролер

(Подпись, дата)

(И.О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 Аналитическая часть	9
1.1 Критерии классификации методов отображения интерфейса	9
1.2 Методы отображения интерфейса в frontend–разработке	9
1.2.1 React	11
1.2.2 Vue.js	12
1.2.3 Angular	14
1.3 Методы отображения интерфейса в мобильной разработке	15
1.3.1 Кроссплатформенная разработка	16
1.3.2 Нативная разработка Android	18
1.3.3 Нативная разработка iOS	20
1.4 Результаты анализа	23
1.5 Постановка задачи	23
2 Конструкторская часть	26
2.1 Требования и ограничения к разрабатываемому методу	26
2.2 Требования к разрабатываемому программному обеспечению	26
2.3 Основные этапы разрабатываемого метода	27
2.4 Схемы алгоритмов	27
3 Технологическая часть	32
3.1 Выбор средств разработки	32
3.1.1 Выбор платформы	32
3.1.2 Выбор языка программирования	32
3.1.3 Выбор среды разработки и сборки программного обеспечения	32
3.1.4 Используемые расширения	32
3.1.5 Выбор методов и средств тестирования	33
3.2 Формат входных и выходных данных	33
3.3 Детали реализации предлагаемого метода	33
4 Исследовательская часть	47
4.1 Технические характеристики	47
4.2 Постановка исследования	47

4.3	Результаты исследования	48
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	50

ВВЕДЕНИЕ

В настоящее время, с развитием технологий и повсеместным использованием интернета, наличие интерфейса играет ключевую роль в обеспечении комфортного и эффективного взаимодействия пользователей с различными приложениями и веб-сервисами, а профессии мобильного или frontend разработчика являются одними из самых востребованных на рынке.

Вёрстка для мобильных и frontend разработчиков является сложным и трудоемким процессом из-за необходимости учитывать различные размеры экранов, плотности пикселей, ориентации устройств и другие аспекты, которые влияют на отображение контента на различных устройствах. Это требует создания адаптивного дизайна, который будет корректно отображаться как на больших, так и на маленьких устройствах. Также стоит учитывать, что изменения в дизайне или структуре приложения могут потребовать переработки большого объема кода, что может быть затратным по времени и ресурсам.

С целью разрешить описанную ранее проблему было создано большое количество методов создания пользовательского интерфейса. Однако не каждый из них предоставляет возможность мгновенно отображать изменения при внесении коррективов в код. Целью данной работы является разработка метода динамического отображения изменений пользовательского интерфейса на основе обработки изменений XML-файла.

Для достижения поставленной цели необходимо решить следующие задачи:

- выявить критерии, по которым можно классифицировать методы отображения пользовательского интерфейса;
- рассмотреть существующие методы отображения интерфейса и классифицировать их согласно выдвинутым критериям;
- разработать метод динамического отображения изменений интерфейса на основе обработки изменений XML-файла;

- разработать программное обеспечение, реализующее представленный метод;
- сравнить скорость внесения изменений в интерфейс разработанной и существующих реализаций.

1 Аналитическая часть

В данном разделе выдвигаются критерии, по которым можно классифицировать методы отображения пользовательского интерфейса. Проводится краткий обзор существующих методов, а также их классификация согласно выдвинутым критериям сравнения. Формализуется постановка задачи в виде IDEF0–диаграммы.

1.1 Критерии классификации методов отображения интерфейса

По результатам опроса, проведенного среди мобильных и frontend–разработчиков, были выявлены критерии, по которым, с точки зрения процесса разработки, можно классифицировать методы создания пользовательского интерфейса:

- **Адаптивность интерфейса:** возможность создания интерфейса, адаптируемого для экранов разного размера или двух ориентаций.
- **Интеграция в существующий код:** возможность постепенного внедрения метода в существующий код или совмещения с другими UI–фреймворками.
- **Механизм обработки изменений интерфейса:** наличие механизма, посредством которого отслеживаются обновления UI–компонента и инициируется его повторное отображение без участия разработчика.
- **Горячая перезагрузка:** возможность отображать изменения интерфейса, внесенные в код по время исполнения, без перекомпиляции всего приложения.

1.2 Методы отображения интерфейса в frontend–разработке

Frontend — это визуальная часть веб–сайта, которую пользователь видит и с которой может взаимодействовать при помощи браузера.

Для разработки frontend в качестве базовых инструментов используются следующие языки:

- HTML (от англ. Hypertext Markup Language, «язык гипертекстовой разметки») — это система форматирования для отображения материалов, полученных через Интернет [2]. HTML обычно используется для структу-

рирования веб-документа. Он определяет такие элементы, как заголовки или абзацы, и позволяет вставлять изображения, видео и другие медиа-файлы.

- CSS (от англ. Cascading Style Sheets, «каскадные таблицы стилей») — это декларативный язык программирования, используемый для разработки контента веб-сайта [3]. Он определяет то, как HTML-элементы будут выглядеть на веб-странице с точки зрения дизайна, макета на разных устройствах с разными размерами экрана. CSS управляет макетом множества различных веб-страниц одновременно.
- JavaScript — язык программирования, который позволяет создавать сложные функции и интерактивность на веб-сайтах и в веб-приложениях, а также в других вариантах использования, повышает общую интерактивность сайта [4]. Веб-страницы, разработанные с помощью JavaScript, реагируют на действия пользователей и обновляются динамически.

Использование JavaScript в разработке стало повсеместным: до 97% веб-сайтов сегодня написаны с его использованием, по сравнению с 88% десять лет назад. За тот же период времени произошли изменения в том, как используется JavaScript: если десять лет назад до 60% веб-сайтов использовали JavaScript без помощи каких-либо библиотек или фреймворков, то сегодня это делают менее 20% [5]. Интерфейсные фреймворки — это предварительно написанный набор стандартизированного кода HTML, CSS и JavaScript, который разработчики могут использовать для более эффективного создания веб-приложений. Многие из подобных библиотек — библиотеки с открытым исходным кодом. По данным с GitHub [6], самыми популярными являются следующие фреймворки:

- React — 221 тысяча Звезд на GitHub [7].
- Vue — 207 тысяч Звезд на GitHub [8].
- Angular — 94,3 тысячи Звезд на GitHub [9].

Рассмотрим каждый из фреймворков детальнее и классифицируем их со-

гласно выделенным критериям.

1.2.1 React

React — это библиотека JavaScript для создания пользовательских интерфейсов [10]. Библиотека была создана для использования в крупных проектах, разрабатывающих объемные интерфейсы и оперирующих данными, которые меняются с течением времени [13].

Большинство современных frontend-фреймворков используют для управления интерфейсом DOM (от англ. Document Object Model) — специальную древовидную структуру, которая позволяет управлять HTML-разметкой JavaScript-кода. Управление обычно состоит из добавления и удаления элементов, изменения их стилей и содержимого [11]. Основной особенностью библиотеки React является наличие виртуального DOM: это концепция, при которой виртуальное представление пользовательского интерфейса хранится в памяти, синхронизированной с «реальным» DOM, React использует его для эффективного обновления пользовательского интерфейса и управления изменениями, вносимыми в него. Причина повышенной производительности заключается в объеме изменяемой информации: сравнивается элемент и его дочерние элементы с предыдущими и применяются только обновления DOM, необходимые для преобразования в желаемое состояние.

Немаловажной особенностью библиотек является компонентная архитектура React, которая позволяет разработчикам создавать повторно используемые UI-элементы и размещать их для создания адаптируемых для разных размеров экранов пользовательских интерфейсов. Концептуально компоненты похожи на функции JavaScript. Они принимают произвольные входные данные (называемые «props» или свойствами) и возвращают React-элементы, описывающие, что должно появиться на экране [13]. Возможность многократного использования компонентов повышает масштабируемость.

Поскольку приложения React используют JavaScript и JSX, разработчики могут применять традиционные методы организации кода. JSX преобразуется

из XML-подобного синтаксиса, с которым многие знакомы, в JavaScript, необходимый React. Самое простое объяснение того, как JSX способен использовать XML-подобный синтаксис и преобразовывать его в JavaScript, который используется для генерации элементов React и компонентов, заключается в том, что он просто сканирует XML-подобную структуру и заменяет теги на функции, необходимые в JavaScript [12]. После компиляции выражения JSX становятся обычными вызовами функций JavaScript и вычисляются в объекты JavaScript [13].

React позволяет внедрять веб-интерфейсы в уже существующий код, например, написанный на JavaScript, без необходимости использования JSX или специальных сборщиков кода. То есть можно создавать элементы React прямо в JavaScript. Для этого используется функция `React.createElement()`, а компоненты определяются как функции или классы. Затем элементы отображаются на странице с помощью `ReactDOM.render()` [13]. Такой подход обеспечивает гибкость и возможность интеграции React в существующий код, что может быть полезно при постепенном переходе к использованию React.

Еще одна особенность React – это горячая перезагрузка (Hot Module Replacement - HMR) [14]. Эта функциональность автоматически обновляет приложение в браузере при сохранении изменений в исходном коде. Она позволяет разработчикам быстро увидеть результаты произведенных изменений без необходимости перезагрузки страницы. Для реализации горячей перезагрузки React использует встроенную функциональность модулей JavaScript. При изменении исходного кода React обновляет только те компоненты, которые изменились, не требуя повторной загрузки всего приложения. Однако горячая перезагрузка — это функциональность, которая обычно реализуется с помощью сторонних инструментов и плагинов.

1.2.2 Vue.js

Vue — это прогрессивный фреймворк для создания пользовательских интерфейсов [15].

Создатели заявляют, что основной особенностью Vue.js, в отличие от других монолитных фреймворков, является возможность постепенного внедрения. Библиотеку легко сочетать с другими методами верстки или интегрировать в существующие проекты [15]. Также можно использовать Vue.js для управления только частью страницы или функциональности, не переписывая полностью существующий код. Для внедрения требуется настроить сборку проекта, чтобы поддерживать Vue компоненты и использовать Vue CLI [18] для управления проектом.

Компоненты в Vue.js являются основным строительным блоком при разработке приложений. Они позволяют создавать переиспользуемые и модульные элементы интерфейса, адаптируемые для разных размеров экранов. Каждый компонент в фреймворке представляет собой независимую единицу, которая содержит свою логику, шаблон и стили.

Vue.js, заимствуя технологию у React, также использует виртуальный DOM, дабы дать разработчику возможность программно создавать, проверять и компоновать нужные структуры пользовательского интерфейса декларативным способом, оставляя прямое управление DOM на усмотрение рендеринга [16]. В силу реактивного подхода фреймворк использует специальные Проху-объекты [17]. Они содержат в себе другие объекты или функции, позволяющие «перехватывать» изменения в них, чтобы узнавать, когда данные стали иными. После первой отрисовки у компонента будет список отслеживаемых зависимостей, полученных из свойств, затронутых в момент отрисовки. Компонент подписывается на каждое из этих свойств. Когда Проху перехватывает операцию обновления, все подписанные на свойство компоненты будут уведомлены и перерисованы.

Горячая перезагрузка в Vue — это функция, которая позволяет заменить экземпляры компонента без перезагрузки страницы. При этом сохраняется текущее состояние приложения и заменяемых компонентов. При создании проекта с помощью Vue CLI — полноценной системы для быстрой разработки на

Vue.js [18] — горячая перезагрузка включена по умолчанию.

1.2.3 Angular

Angular — веб-фреймворк, который позволяет разработчикам создавать быстрые и надежные приложения, предоставляет широкий набор инструментов, API и библиотек для упрощения процесса разработки [19].

Чтобы связывать данные приложения и отображение, Angular использует HTML-подобный синтаксис для своих шаблонов и компилирует эти шаблоны в набор инструкций, которые создают браузерные DOM-элементы. Angular не использует виртуальный DOM, чтобы хранить дерево компонентов в памяти [21]. Вместо этого фреймворк предлагает две стратегии для обнаружения изменений: `default` и `onPush` [23]. Первая стратегия сводится к рекурсивному проходу по дереву компонентов и сравнению текущего значения с предыдущим для каждого выражения, используемого в шаблоне. Если значения отличаются, то фреймворк отмечает их специальным образом и по окончании прохода меняет отображение в DOM. Стратегия очень удобна, так как разработчику ничего не нужно проверять самостоятельно, однако в больших приложениях может вызывать проблему с производительностью из-за частых перезапусков проверок на любое возникающее событие. В борьбе за производительность создатели Angular разработали вторую стратегию — `onPush`, которая отключает автоматическое обнаружение изменений. Тем не менее обнаружение изменений все еще может быть вызвано явно, например, путем вызова соответствующих асинхронных функций. Стратегии обновления могут использоваться одновременно: `onPush` чаще применяют для оптимизации высоконагруженных частей приложения.

Компоненты Angular обеспечивают структуру для организации проекта и деление на простые для понимания части с четкими обязанностями, чтобы код был масштабируемым [20]. Как React и Vue.js, фреймворк позволяет создавать адаптируемые для разных размеров устройств интерфейсы.

В Angular также есть поддержка горячей перезагрузки (Hot Module Replacement).

— HMR) [22]. Горячая перезагрузка позволяет обновлять только те части приложения, которые были изменены, без полной перезагрузки страницы. Это ускоряет процесс разработки и упрощает работу с приложением. Angular CLI, инструмент командной строки для разработки приложений, поддерживает горячую перезагрузку по умолчанию.

Фреймворк предоставляет возможность постепенного внедрения, что позволяет добавлять Angular-компоненты, модули и функциональность к существующему веб-приложению по мере необходимости.

1.3 Методы отображения интерфейса в мобильной разработке

Двумя самыми популярными в мире операционными системами для мобильных устройств являются Android и iOS [24]. Android разработан компанией Google и используется на широком спектре устройств от разных производителей, таких как Samsung, Huawei, Xiaomi и другие [25]. iOS, в свою очередь, разработана компанией Apple и используется на устройствах iPhone, iPad и iPod Touch [26].

Обе операционные системы имеют свои особенности, интерфейсы и набор функций, и разработчики мобильных приложений часто создают версии своих приложений как для iOS, так и для Android, чтобы охватить более широкую аудиторию пользователей. В связи с чем появилось два подхода к разработке программного обеспечения для мобильных устройств: нативная и кроссплатформенная разработка.

Нативная разработка — это процесс создания приложений для определенной операционной системы или платформы с использованием языков программирования и инструментов, специфичных для данной платформы [27]. Для iOS приложений используется язык программирования Swift или Objective-C, а для Android — Java или Kotlin.

Кроссплатформенная разработка подразумевает создание приложений, которые могут работать на различных операционных системах без необходимости создания отдельных версий для каждой из них [27]. Для кроссплатформен-

ной разработки часто используются фреймворки и инструменты, которые позволяют написать код один раз и запускать его на разных платформах. Кроссплатформенная разработка обычно упрощает процесс создания приложений для разных платформ, но может иметь ограничения по производительности и доступу к специфическим функциям устройств.

Далее будут рассмотрены и классифицированы методы отображения интерфейса для каждого из подходов.

1.3.1 Кроссплатформенная разработка

Одними из самых популярных кроссплатформенных мобильных фреймворков являются Flutter и React Native [28].

Flutter

Flutter — это инструментальный пользовательского интерфейса Google для создания приложений для мобильных устройств, веб-приложений и настольных компьютеров на основе единой кодовой базы [29].

В силу этой особенности, главным свойством фреймворка является возможность создавать интерфейс, адаптируемый не только для разных размеров конкретного устройства, например, экрана компьютера, а разных типов устройств. В связи с этим Flutter обладает большим количеством виджетов и технологий, упрощающих задачу компоновки.

Официальная документация фреймворка гласит: «Иногда нецелесообразно переписывать всё приложение на Flutter сразу. В таких ситуациях Flutter можно интегрировать в ваше существующее приложение по частям, в виде библиотеки или модуля. Затем этот модуль можно импортировать в приложение для Android или iOS, чтобы отобразить часть пользовательского интерфейса вашего приложения во Flutter» [30].

Виджеты в Flutter представляют собой элементы пользовательского интерфейса, которые позволяют отображать содержимое и взаимодействовать с

пользователем. Виджеты могут быть простыми, а могут и содержать другие виджеты внутри себя. При изменении состояния виджета перестраивается его описание, которое фреймворк сравнивает с предыдущим описанием, чтобы определить минимальные изменения, необходимые в базовом дереве рендеринга для перехода из одного состояния в следующее [31].

Flutter поддерживает функцию горячей перезагрузки, что помогает сократить время создания пользовательских интерфейсов. Технология работает путем внедрения обновленных файлов исходного кода в работающую виртуальную машину, на которой развернуто приложение. После того, как виртуальная машина обновляет классы новыми версиями полей и функций, платформа автоматически перестраивает дерево виджетов, позволяя быстро просмотреть последствия изменений [32].

React Native

React Native — это фреймворк с открытым исходным кодом для создания приложений для Android и iOS, использующий React и собственные возможности платформы приложений [33]

Особенностью React Native, как кроссплатформенного фреймворка, является наличие возможности создания версий компонентов, зависящих от платформы, чтобы одна кодовая база могла совместно использовать код на разных платформах. С React Native одна команда может поддерживать несколько платформ и использовать общую технологию — React. Таким образом фреймворк позволяет создавать интерфейс, для разных размеров и ориентаций устройств.

React Native подходит для добавления кода в существующие приложения. Выполнив несколько шагов, можно добавить новые функции, экраны, представления на основе React Native и т.д. Однако, настройка интеграции может отличаться для разных платформ [35].

При разработке для Android представления создаются с помощью кода на

Kotlin или Java, для iOS — с помощью Swift или Objective-C. React Native может вызывать эти представления с помощью JavaScript, используя компоненты React. Во время выполнения React Native создает соответствующие представления Android и iOS для этих компонентов. Поскольку компоненты React Native поддерживаются теми же представлениями, что и Android и iOS, приложения React Native выглядят и работают так же, как и приложения, созданные с помощью нативных языков.

Как и виртуальный DOM в React, React Native создает древовидную иерархию для определения представлений. Когда фреймворк отправляет команды для рендеринга макета, группа теневых узлов собирается для построения теневого (виртуального) дерева, которое представляет изменяемую сторону макета, написанную на соответствующем нативном языке [36]. Затем преобразуется в фактические представления на экране, с помощью Yoga — кроссплатформенного движка верстки [37].

Также как и React, React поддерживает функцию горячей перезагрузки.

1.3.2 Нативная разработка Android

Метод с использованием XML

Классический метод верстки на Android предполагает размещение элементов на экране с помощью XML-файла [38]. XML — это расширяемый язык разметки, который предоставляет правила для определения любых данных. XML-файл состоит из различных тегов и атрибутов, которые задают верстку всему экрану. В среде разработки, благодаря использованию специальных инструментов, можно наблюдать, как будет выглядеть макет на различных типах и форматах дисплеев.

Сам по себе XML-файл представляет только разметку, то есть то, как элементы интерфейса будут выглядеть и располагаться на экране. Для управление

логикой работы экрана необходимо использовать дополнительный код на языке Java или Kotlin. Аналогом виртуального DOM в XML-верстке на Android можно считать View hierarchy, которая представляет собой иерархию всех пользовательских интерфейсных элементов (View) в XML-файлах [39]. Используя XML для верстки интерфейса на Android, разработчики создают дерево View элементов, которое затем обрабатывается и отображается на экране устройства. При обновлении интерфейса в приложении система Android перестраивает и обновляет View hierarchy по аналогии с тем, как виртуальный DOM обновляет DOM веб-приложений. Таким образом при XML-верстке на Android присутствует механизм автоматического обновления UI-компонентов при изменениях, которые позволяют повторно отобразить интерфейс без необходимости вмешательства разработчика.

XML-верстка хорошо интегрируется в существующий код, позволяя постепенно внедрять новые методы в существующую структуру приложения и сочетаться с другими UI-фреймворками.

Метод не поддерживает горячую перезагрузку, что означает, что для просмотра изменений, внесенных в код во время исполнения, требуется перезапуск приложения.

Jetpack Compose

Jetpack Compose — это современный инструмент для создания собственного пользовательского интерфейса Android [40].

Как декларативный инструмент пользовательского интерфейса, Jetpack Compose хорошо подходит для разработки и реализации макетов, которые настраиваются для отображения контента по-разному в различных размерах [41].

Jetpack Compose позволяет поэтапно внедрять свой метод создания UI в существующий код приложения, постепенно заменяя старые View-based компоненты новыми Compose-компонентами [41]. Также возможна параллельная

работа с другими UI-фреймворками.

С помощью Compose можно создать свой пользовательский интерфейс, определив набор составных функций, которые принимают данные и генерируют элементы. В модели компоновки дерево пользовательского интерфейса создается за один проход. Сначала каждому узлу предлагается измерить себя, затем рекурсивно измерить все дочерние узлы, передавая ограничения размера вниз по дереву дочерним узлам. Элементы komponуются в синтаксическое дерево, которое представляет структуру пользовательского интерфейса. Это дерево называется Composable. Composables можно вкладывать друг в друга, создавая сложные пользовательские интерфейсы.

В Jetpack Compose существует аналог виртуального DOM, который называется Recomposition [42]. Recomposition подходит для пересоздания только тех частей пользовательского интерфейса, которые изменились, вместо пересоздания всего UI. Это позволяет эффективно обновлять UI при изменении данных или состояния приложения. Recomposition в Jetpack Compose работает путем сравнения и обновления Composables, которые используются для отображения пользовательского интерфейса. При изменении данных или состояния, Composables, которые зависят от этих изменений, будут пересозданы, а затем сравнены с предыдущими версиями и обновлены на экране.

Одним из недостатков фреймворка Jetpack Compose является отсутствие возможности быстрой перезагрузки кода без перезапуска всего приложения. Отсутствие функции hot reload делает процесс разработки менее удобным по сравнению с другими фреймворками.

1.3.3 Нативная разработка iOS

UIKit

UIKit предоставляет множество функций для создания приложений, включая компоненты, которые можно использовать для создания базовой инфраструктуры приложений для iOS, iPadOS или tvOS [43].

Интерфейс с помощью UIKit можно создавать двумя способами [44]: с использованием Interface Builder — инструмента, позволяющего разработчикам визуально создавать и настраивать пользовательские интерфейсы — посредством редактирования «.storyboard» и «.xib» файлов, которые описывают интерфейс с помощью XML (то есть без написания кода), или с помощью Swift или Objective-C программно, например, с помощью frame [46] или AutoLayout [47]. Interface Builder не всегда предоставляет возможность создавать интерфейсы, адаптируемые ко всем типам устройств, в отличие от второго метода.

UIKit легко интегрируется в существующий код, позволяя постепенно внедрять новые методы и компоненты в уже существующую структуру приложения. Он также хорошо совмещается с другими UI-фреймворками, такими как SwiftUI [45].

В UIKit Apple каждый экран приложения представлен иерархией view — объектов класса UIView или его наследников [48]. После изменения данных модели, UIKit автоматически обновляет и перерисовывает только те части иерархии, которые фактически изменились, минуя объекты, которые остались неизменными. Это позволяет обновлять интерфейс более эффективно.

UIKit не поддерживает горячую перезагрузку, поэтому для отображения изменений, внесенных в код во время исполнения, требуется перекомпиляция и перезапуск приложения. Однако при проектировании интерфейса через Interface Builder появляется возможность визуально создавать элементы экрана.

SwiftUI

SwiftUI — это декларативный и модернизированный фреймворк пользовательского интерфейса Apple, который предоставляет нативный способ создания

интерфейсов для iOS, macOS, watchOS и tvOS [49].

SwiftUI поддерживает адаптивную разметку, что позволяет создавать интерфейсы, которые автоматически адаптируются под разные размеры экранов и ориентации. Можно использовать специальные UI-элементы, например, `Stack`, а также модификаторы для управления распределением элементов интерфейса на различных устройствах и экранах [50].

SwiftUI может постепенно внедряться в существующий код приложения, позволяя создавать новые экраны или переписывать существующие компоненты постепенно. Он также может быть интегрирован с `UIKit`, что позволяет использовать оба фреймворка в одном приложении, например, для перехода на SwiftUI постепенно [45].

SwiftUI предлагает декларативный подход к проектированию пользовательского интерфейса и автоматически обновляет затронутые части интерфейса при изменении данных. Чтобы обновлять представления при изменении данных, классы модели данных необходимо объявить наблюдаемыми объектами (observable) [52]. Чтобы изменения в данных, получаемые от пользователя, возвращались в модель, элементы управления пользовательского интерфейса связываются со свойствами модели. Управление состоянием является основой для создания эффективных приложений, которые поддерживают актуальность информации для пользователя.

SwiftUI не поддерживает функцию горячей перезагрузки. Однако Xcode — интегрированная среда разработки программного обеспечения для платформ macOS, iOS, iPadOS, watchOS, tvOS — при создании пользовательского приложения с помощью SwiftUI предоставляет возможность отображать предварительный просмотр содержимого представления, которое остается актуальным по мере внесения изменений в код представления. Эта функциональность доступна при использовании в коде специальных макросов [51].

1.4 Результаты анализа

Классификация методов отображения пользовательского интерфейса приведена в таблице 1. Для краткости записи в данной таблице используются следующие обозначения описанных критериев:

- K1 — адаптивность интерфейса;
- K2 — интеграция в существующий код;
- K3 — механизм обработки изменений интерфейса;
- K4 — горячая перезагрузка.

Таблица 1 – Классификация методов отображения пользовательского интерфейса

Класс метода	Название	K1	K2	K3	K4
Frontend–разработка	React	+	+	+	+
	Vue	+	+	+	+
	Angular	+	+	+	+
Кроссплатформенная мобильная разработка	Flutter	+	+	+	+
	React Native	+	+	+	+
Нативная мобильная разработка Android	Метод с использованием XML	+	+	+	-
	Jetpack Compose	+	+	+	-
Нативная мобильная разработка iOS	UIKit	+	+	+	-
	SwiftUI	+	+	+	-

1.5 Постановка задачи

Этап создания интерфейса является одной из ключевых стадий разработки приложений и веб–сайтов [53], определяющим успех проекта и уровень его

пользовательской привлекательности. Существует множество методов отображения пользовательского интерфейса, предоставляющих различные возможности разработки. По результатам классификации наиболее популярных из них на первый план выходит отсутствие горячей перезагрузки в ряде фреймворков. Данная функция экономит время разработки, обновляя только те элементы интерфейса, которые были изменены, без необходимости перезагрузки всего приложения. Разрабатываемый метод должен предоставлять возможность динамически отображать изменения пользовательского интерфейса.

XML — язык форматирования документов [54], основан на тегах, которые определяют начало и конец элементов данных. Каждый XML—документ начинается с корневого элемента, содержащего все остальные. Элементы могут содержать атрибуты, которые дополняют информацию о содержимом. XML позволяет создавать иерархические структуры данных. Интерфейс должен задаваться посредством написания XML—файла, а разрабатываемый метод должен предоставлять возможность отображать изменения интерфейса на основе обработки этого файла.

Задача формализована с помощью диаграммы IDEF0 уровня A0, представленной на рисунке 1.

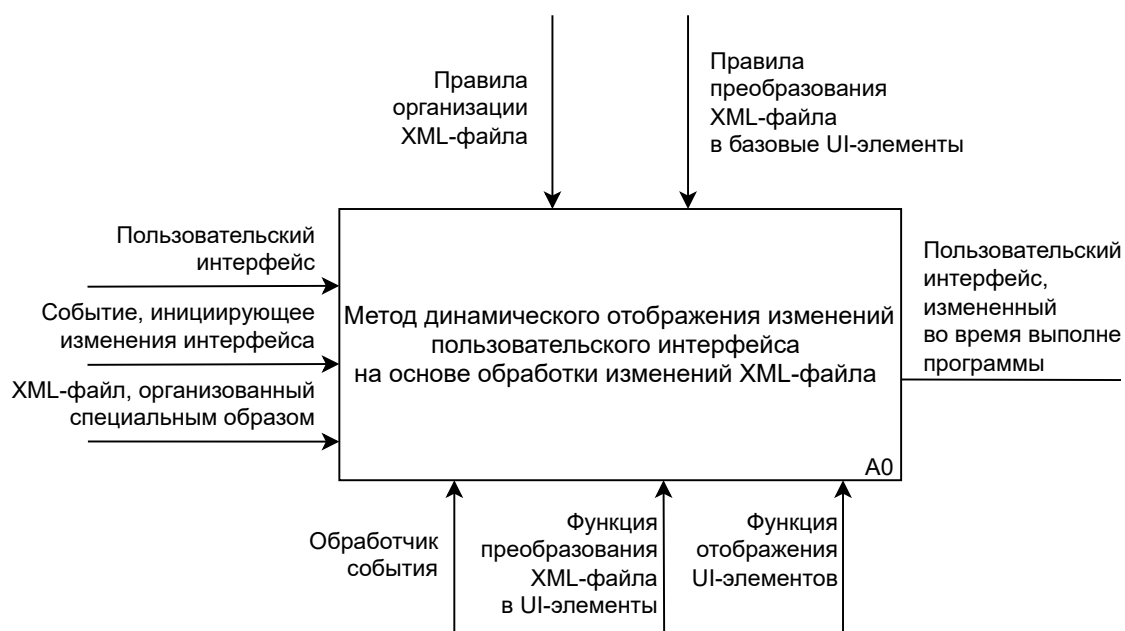


Рисунок 1 – IDEF0 — диаграмма уровня A0

Вывод

В данном разделе были выдвинуты критерии классификации методов отображения пользовательского интерфейса. Проведен краткий обзор существующих методов, а также их классификация согласно выдвинутым критериям сравнения. Была формализована постановка задачи.

2 Конструкторская часть

В данном разделе будут сформулированы требования и ограничения к разрабатываемому методу. Разработан метод динамического отображения изменений пользовательского интерфейса на основе обработки изменений XML-файла. Описаны основные этапы разработки в виде детализированной диаграммы IDEF0 и схем алгоритмов, а также изложены особенности излагаемого метода.

2.1 Требования и ограничения к разрабатываемому методу

К методу динамического отображения изменений пользовательского интерфейса на основе обработки изменений XML-файла предъявляются следующие требования:

- 1) Обработка события, инициирующего изменения интерфейса во время выполнения.
- 2) Преобразование XML-файла, организованного определенным образом, в UI-элементы.
- 3) Размещение UI-элементов на экране.

Также представлено ограничение для разрабатываемого метода: при неправильной организации XML-файла не гарантируется корректное отображение элементов на экране.

2.2 Требования к разрабатываемому программному обеспечению

К разрабатываемому программному обеспечению предъявляются следующие требования:

- 1) Возможность обработки события, инициирующего изменение интерфейса.
- 2) Возможность во время выполнения программы изменять интерфейс посредством внесения изменения в XML-файл по срабатыванию обработчика события.
- 3) Возможность создавать интерфейсы с вложенностью элементов.

- 4) Возможность создавать интерфейсы со всеми базовыми UI–элементами.
- 5) Возможность оперировать базовыми параметрами UI–элементов.
- 6) Возможность комбинировать существующие методы создания интерфейса с разрабатываемым.
- 7) Разрабатываемое ПО должно корректно реагировать на любые действия пользователя.

2.3 Основные этапы разрабатываемого метода

На рисунке 2 представлена диаграмма IDEF0 уровня A1 для разрабатываемого метода.

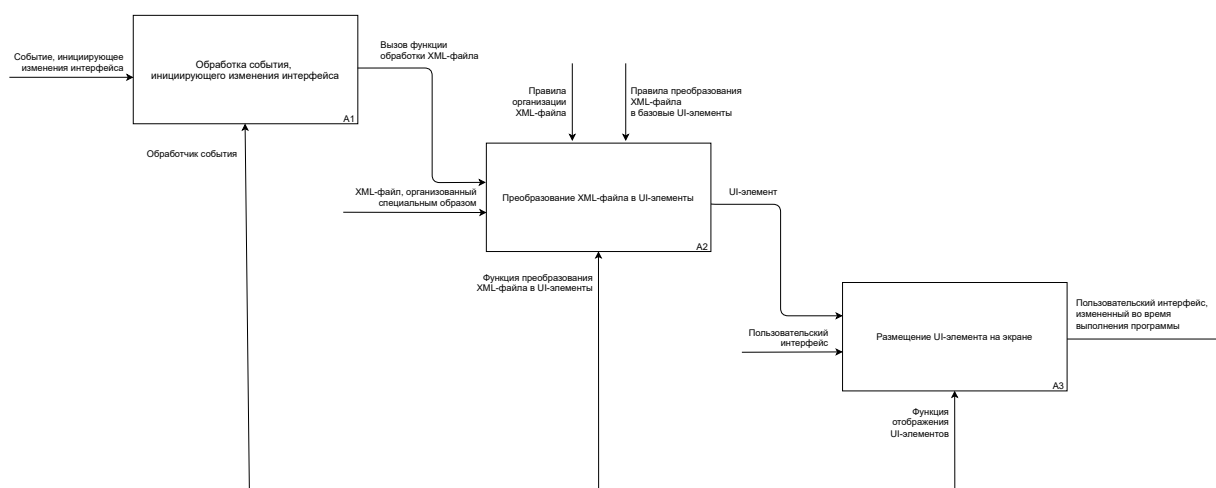


Рисунок 2 – IDEF0 – диаграмма уровня A1

2.4 Схемы алгоритмов

На **первом** этапе необходимо обработать событие, которое станет инициатором изменения интерфейса. Данным событием может стать нажатие определенного сочетания клавиш. Каждый проектируемый экран приложения должен

быть связан с конкретным XML–файлом и заранее внесен в список наблюдаемых объектов. По свершении события происходит вызов функции обработки XML–файла для каждого такого экрана.

На **втором** этапе происходит обработка XML–файла. Для этого файл должен быть организован определенным образом, чтобы в дальнейшем функция обработки корректно распознавала UI–элементы, являющиеся базовыми для платформы: первый тег XML–файла содержит в себе название базового элемента и все его необходимые свойства и их значения (название свойства и значение разделяет знак равенства «=», значение заключается в кавычки «"»»), второй тег содержит название базового элемента. Для создания вложенности элементов среди свойств элемента–родителя необходимо указать теги дочернего элемента. Разметка для дочернего элемента в данном случае задается по правилам, аналогичным правилам создания родительского элемента.

Например, чтобы в нативной разработке iOS получить UILabel, занимающий 100% площади экрана и содержащий слово «Hello», расположенное по левому краю, необходимо добавить в XML–файл разметку, представленную в листинге 1.

Листинг 1 – Разметка XML–файла для UILabel

```
1 <UILabel
2     width="100%"
3     height="100%"
4     textAlignment="left"
5     text="Hello">
6 </UILabel>
```

В рамках данного этапа происходит получение содержимого XML–файла, разделение файла на строки для получения списка его компонентов. Далее происходит создание корневого представления, на котором будут размещены все элементы, полученные в ходе обработки компонентов XML–файла. После чего корневое представление и компоненты передаются в функцию преобразова-

ние элементов XML-разметки в UI-элементы. Для корректной работы функции необходимо иметь список всех базовых UI-элементов, с которыми предполагается взаимодействие метода, а также список свойств и атрибутов для каждого элемента. Алгоритмы работы функций представлены на рисунках 3 — 4.

Для второго этапа входным параметром XML-файл, организованный специальным образом, выходным — UI-элемент, включающий все представления, описанные в XML-файле.

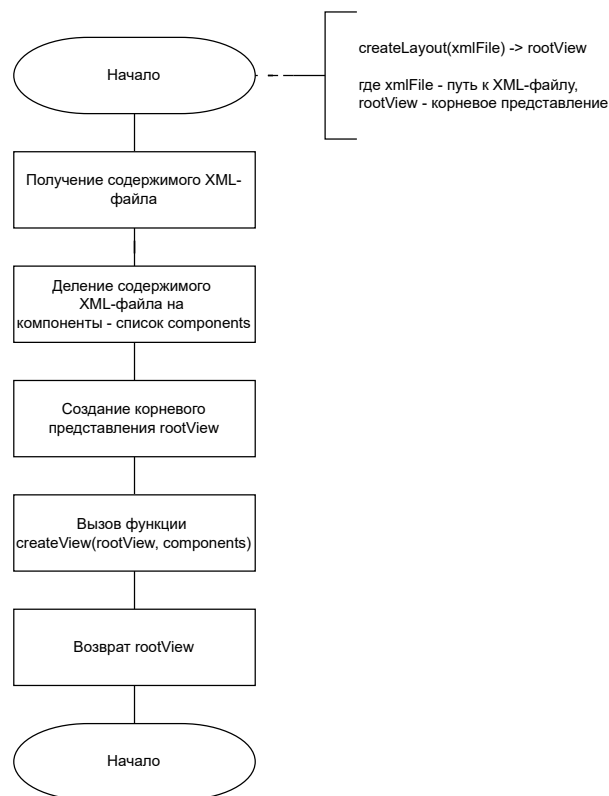


Рисунок 3 – Алгоритм функции создания представления, содержащего описанные в XML-файле элементы

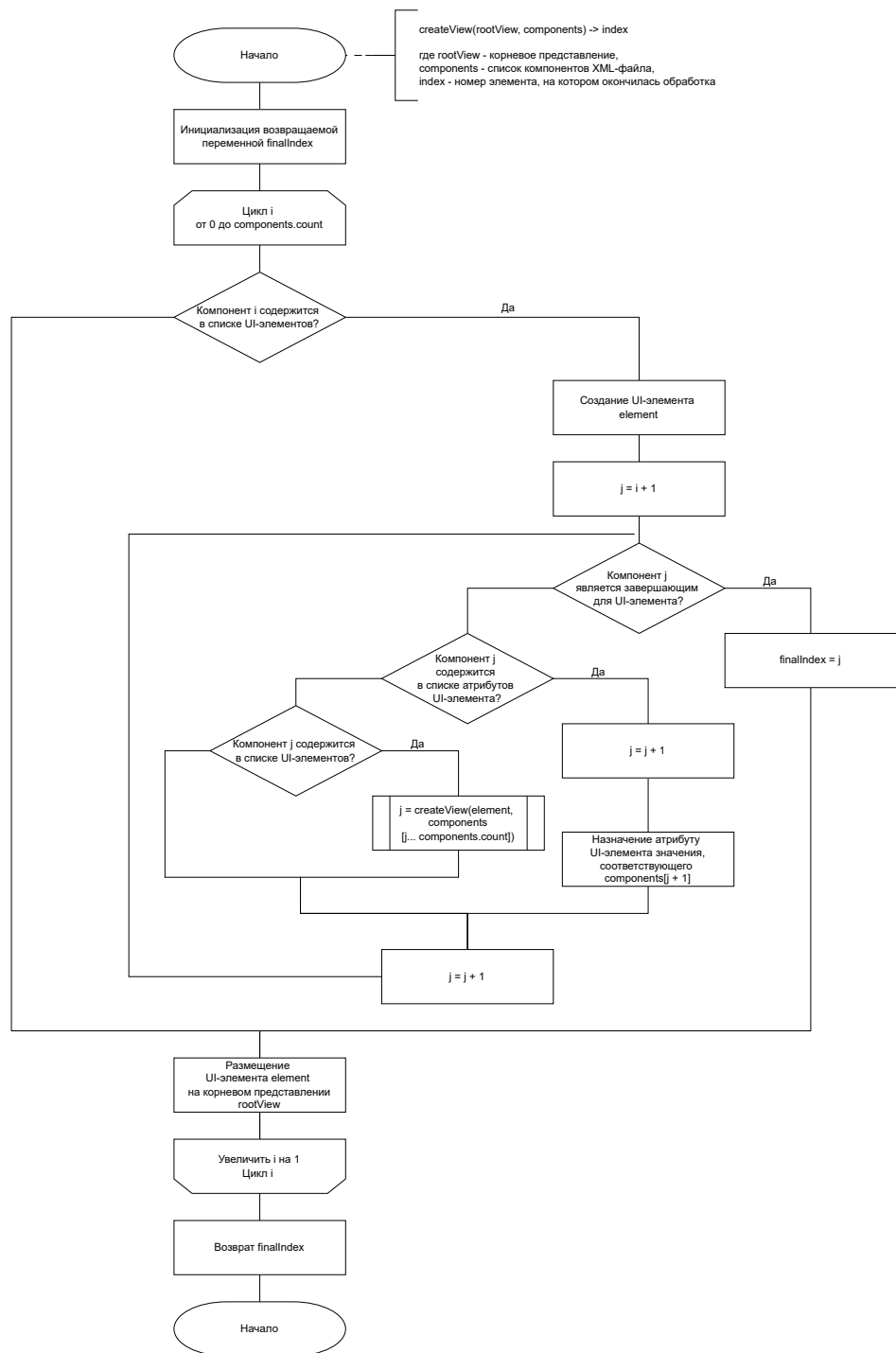


Рисунок 4 – Алгоритм функции создания представлений

На **третьем** этапе происходит размещение полученного UI-элемента на экране. Для корректного отображения обновлений данный этап должен быть

включен в жизненный цикл корневого представления экрана.

Вывод

В данном разделе были сформулированы требования и ограничения к разрабатываемому методу. Разработан метод динамического отображения изменений пользовательского интерфейса на основе обработки изменений XML-файла. Описаны основные этапы разработки в виде детализированной диаграммы IDEF0 и схем алгоритмов, а также изложены особенности разработанного метода.

3 Технологическая часть

В данном разделе будет обоснован выбор средств программной реализации предлагаемого метода, описан формат входных и выходных данных. Разработано программное обеспечение, реализующее представленный метод, и выполнено его тестирование.

3.1 Выбор средств разработки

3.1.1 Выбор платформы

По результатам анализа только исследуемые методы отображения интерфейса нативной мобильной разработки не обладают функцией горячей перезагрузки. В связи с чем в качестве платформы разработки для реализации предлагаемого метода выбрана операционная система iOS.

3.1.2 Выбор языка программирования

Выбран язык Swift, поскольку он является основным используемым в нативной разработке iOS.

3.1.3 Выбор среды разработки и сборка программного обеспечения

Среда разработки — XCode, поскольку является интегрированной средой разработки программного обеспечения для платформ macOS, iOS, watchOS и tvOS, разработанной корпорацией Apple, а также предоставляет возможность запуска разрабатываемого программного обеспечения на симуляторах устройств. Используется версия XCode 15.3, являющаяся актуальной на момент написания дипломной работы. Запуск разрабатываемого программного обеспечения происходит на симуляторе устройства iPhone 15 Pro Max, версия iOS — 17.4 (также является актуальной на момент написания дипломной работы).

3.1.4 Используемые расширения

Для создания представлений используется библиотека UIKit, содержащая полный набор базовых UI-элементов, а также механизмы размещения их на экране.

3.1.5 Выбор методов и средств тестирования

Разработанное программное обеспечение должно корректно отображать задаваемые UI-элементы. Для этого необходимо провести snapshot-тесты — тесты, которые делают эталонный скриншот экрана и сравнивают с актуальным скриншотом, который получается во время прогона тестов. Для выполнения тестирования будет использована библиотека SnapshotTesting, предоставляющая весь необходимый для создания snapshot-тестов функционал.

3.2 Формат входных и выходных данных

Входные данные для разрабатываемого метода:

- пользовательский интерфейс;
- событие, инициализирующее изменения интерфейса;
- XML-файл, организованный специальным образом.

Выходные данные для разрабатываемого метода:

- пользовательский интерфейс, измененный во время выполнения программы.

3.3 Детали реализации предлагаемого метода

В листинге 2 представлен класс, обеспечивающий перезагрузку по время выполнения.

Листинг 2 – Класс, обеспечивающий перезагрузку по время выполнения

```
1
2 class ReloadManager {
3     private static var observersWeak = [ObserverWeak]()
4
5     private struct ObserverWeak {
6         weak var observer: LayoutInTimeViewController?
7     }
8
9     static var observers: [LayoutInTimeViewController] {
10         return observersWeak.compactMap { $0.observer }
11     }
12
13     static func addObserver(_ observer: LayoutInTimeViewController) {
```

```

14         var alreadyRegistered = false
15         observersWeak = observersWeak.filter {
16             guard let o = $0.observer else { return false }
17             if o === observer { alreadyRegistered = true }
18             return true
19         }
20         if alreadyRegistered {
21             return
22         }
23         observersWeak.append(ObserverWeak(observer: observer))
24
25         #if arch(arm64) || arch(x86_64)
26
27             if !UIResponder.handlerInstalled {
28                 overrideMethod(#selector(getter: UIResponder.keyCommands),
29                               of: UIResponder.self,
30                               with: #selector(UIResponder.
layoutItTimeKeyCommands)
31                               )
32                 UIResponder.handlerInstalled = true
33             }
34
35         #endif
36     }
37
38     static func reload() {
39         for observer in observers {
40             observer.reload()
41         }
42     }
43 }

```

Реализации класса использует функцию, представленную в листинге 3, которая определяет список событию, по свершению которых происходит перезагрузка, а также метод, представленный в листинге 4, переопределяющий функцию обработки нажатия горящих клавиш.

Листинг 3 – Функция определения списка событий, по свершению которых происходит перезагрузка

```

1 #if arch(arm64) || arch(x86_64)

```

```

2     private extension UIResponder {
3         static var handlerInstalled = false
4
5         @objc func layoutItTimeKeyCommands() -> [UIKeyCommand]? {
6
7             return (layoutItTimeKeyCommands() ?? []) + [
8                 UIKeyCommand(
9                     input: "t",
10                    modifierFlags: .command,
11                    action: #selector(reloadInTime)
12                ),
13            ]
14        }
15
16        @objc private func reloadInTime() {
17            ReloadManager.reload()
18        }
19    }
20 #endif

```

Листинг 4 – Класс, обеспечивающий перезагрузку по время выполнения

```

1 func overrideMethod(_ selectorA: Selector, of sourceClass: AnyClass, with
    selectorB: Selector) {
2     let newMethod = class_getInstanceMethod(sourceClass, selectorB)!
3     let oldMethod = class_getInstanceMethod(sourceClass, selectorA)!
4     let inheritedImplementation = class_getInstanceMethod(class_getSuperclass
    (sourceClass), selectorA)
5     .map(method_getImplementation)
6     if method_getImplementation(oldMethod) == inheritedImplementation {
7         let types = method_getTypeEncoding(oldMethod)
8         class_addMethod(sourceClass, selectorA, method_getImplementation(
    newMethod), types)
9         return
10    }
11    method_exchangeImplementations(oldMethod, newMethod)
12 }

```

Для того, чтобы добавить экран — контроллер — в список обновляемых элементов, необходимо в методе `ViewDidLoad` его жизненного цикла добавить

вызов функции addObserver класса перезагрузки ReloadManager. Вызов представлен в листинге 5.

Листинг 5 – Класс, обеспечивающий перезагрузку по время выполнения

```
1 ReloadManager.addObserver(self)
```

В листинге 6 представлены перечисления и структуры, определяющие список базовых UI-элементов, их свойства и атрибуты.

Листинг 6 – Перечисления и структуры, определяющие список базовых UI-элементов, их свойства и атрибуты

```
1 enum ComponentType {
2     case uiView
3     case uiLabel
4     case uiButton
5
6     var startTag: String {
7         switch self {
8             case .uiView:
9                 return "UIView"
10            case .uiLabel:
11                return "UILabel"
12            case .uiButton:
13                return "UIButton"
14        }
15    }
16
17    var endTag: String {
18        switch self {
19            case .uiView:
20                return "/UIView"
21            case .uiLabel:
22                return "/UILabel"
23            case .uiButton:
24                return "/UIButton"
25        }
26    }
27 }
28
```

```

29 struct Anchors {
30     public var topAnchorConstant: CGFloat?
31     public var bottomAnchorConstant: CGFloat?
32     public var leftAnchorConstant: CGFloat?
33     public var rightAnchorConstant: CGFloat?
34     public var height: CGFloat?
35     public var width: CGFloat?
36 }
37
38 enum ComponentPropertyType {
39     case width
40     case height
41     case topAnchor
42     case bottomAnchor
43     case leftAnchor
44     case rightAnchor
45     case backgroundColor
46     case textAlignment
47     case text
48
49     var value: String {
50         switch self {
51             case .width:
52                 "width"
53             case .height:
54                 "height"
55             case .topAnchor:
56                 "topAnchor"
57             case .bottomAnchor:
58                 "bottomAnchor"
59             case .leftAnchor:
60                 "leftAnchor"
61             case .rightAnchor:
62                 "rightAnchor"
63             case .backgroundColor:
64                 "backgroundColor"
65             case .textAlignment:
66                 "textAlignment"
67             case .text:
68                 "text"

```

```

69         }
70     }
71 }

```

В листинге 7 представлены функции класса, преобразующего содержимое XML-файла в UI-элементы.

Листинг 7 – Функция определения списка событий, по свершению которых происходит перезагрузка

```

1 class LayoutInTime {
2     public func createLayout(rootView: UIView, from xml: String) {
3         guard let clearContent = parseXMLElements(xmlContent: xml) else {
4             return
5         }
6         let separatedClearContent = clearContent.components(separatedBy: "\n"
7         )
8         let joinedClearContent = separatedClearContent.joined(separator: "=")
9         let components = joinedClearContent.components(separatedBy: "=")
10
11         print(components)
12         _ = createView(rootView: rootView, from: components)
13     }
14
15     private func createView(rootView: UIView, from components: [String]) ->
16     Int {
17         var i = 0
18         var finalTagIndex = 0
19         while i < components.count {
20             switch components[i] {
21                 case ComponentType.uiView.startTag:
22                     var j = i + 1
23                     var uiView = UIView().autolayout()
24                     var anchors = Anchors()
25                     while components[j] != ComponentType.uiView.endTag {
26                         switch components[j] {
27                             case ComponentType.uiView.startTag,
28                                 ComponentType.uiLabel.startTag,
29                                 ComponentType.uiButton.startTag:
30                             j += createView(rootView: uiView, from: Array(
31                                 components[j..

```

```

30         case ComponentPropertyType.width.value:
31             incrementIndex(index: &j)
32             anchors.width = setWidthHeight(property: components[j
], isHeight: false)
33         case ComponentPropertyType.height.value:
34             incrementIndex(index: &j)
35             anchors.height = setWidthHeight(property: components[
j], isHeight: true)
36
37         case ComponentPropertyType.topAnchor.value:
38             incrementIndex(index: &j)
39             anchors.topAnchorConstant = Double(components[j]) ??
0.0
40         case ComponentPropertyType.bottomAnchor.value:
41             incrementIndex(index: &j)
42             anchors.bottomAnchorConstant = Double(components[j])
?? 0.0
43         case ComponentPropertyType.leftAnchor.value:
44             incrementIndex(index: &j)
45             anchors.leftAnchorConstant = Double(components[j]) ??
0.0
46         case ComponentPropertyType.rightAnchor.value:
47             incrementIndex(index: &j)
48             anchors.rightAnchorConstant = Double(components[j])
?? 0.0
49
50         case ComponentPropertyType.backgroundColor.value:
51             incrementIndex(index: &j)
52             UIView = setBackgroundColor(component: UIView,
property: components[j])
53
54         default:
55             continue
56     }
57     incrementIndex(index: &j)
58 }
59 finalTagIndex = j
60 rootView.addSubview(UIView)
61 setAnchors(rootView: rootView, childView: UIView, anchors:
anchors)

```

```

62         incrementIndexByIndex(indexI: &i, by: j)
63
64         case ComponentType.uiLabel.startTag:
65             var j = i + 1
66             var uiLabel = UILabel().autolayout()
67             var anchors = Anchors()
68             while components[j] != ComponentType.uiLabel.endTag {
69                 print(components[j])
70                 switch components[j] {
71                     case ComponentType.uiView.startTag,
72                         ComponentType.uiLabel.startTag,
73                         ComponentType.uiButton.startTag:
74                         j += createView(rootView: uiLabel, from: Array(
components[j..

```

```

95
96         case ComponentPropertyType.backgroundColor.value:
97             incrementIndex(index: &j)
98             uiLabel = setBackgroundColor(component: uiLabel,
property: components[j]) as! UILabel
99
100        case ComponentPropertyType.textAlignment.value:
101            incrementIndex(index: &j)
102            setTextAlignent(component: &uiLabel, property:
components[j])
103
104        case ComponentPropertyType.text.value:
105            incrementIndex(index: &j)
106            uiLabel.text = components[j]
107
108        default:
109            continue
110    }
111    incrementIndex(index: &j)
112    }
113    finalTagIndex = j
114    rootView.addSubview(uiLabel)
115    setAnchors(rootView: rootView, childView: uiLabel, anchors:
anchors)
116
117    incrementIndexByIndex(indexI: &i, by: j)
118
119    case ComponentType.uiButton.startTag:
120        var j = i + 1
121        var uiButton = UIButton().autolayout()
122        var anchors = Anchors()
123        while components[j] != ComponentType.uiButton.endTag {
124            switch components[j] {
125                case ComponentType.uiView.startTag,
ComponentType.uiLabel.startTag,
ComponentType.uiButton.startTag:
126                    j += createView(rootView: uiButton, from: Array(
components[j..

```

```

    ], isHeight: false)
131         case ComponentPropertyType.height.value:
132             incrementIndex(index: &j)
133             anchors.height = setWidthHeight(property: components[
134 j], isHeight: true)
135
136         case ComponentPropertyType.topAnchor.value:
137             incrementIndex(index: &j)
138             anchors.topAnchorConstant = Double(components[j]) ??
139 0.0
140
141         case ComponentPropertyType.bottomAnchor.value:
142             incrementIndex(index: &j)
143             anchors.bottomAnchorConstant = Double(components[j])
144 ?? 0.0
145
146         case ComponentPropertyType.leftAnchor.value:
147             incrementIndex(index: &j)
148             anchors.leftAnchorConstant = Double(components[j]) ??
149 0.0
150
151         case ComponentPropertyType.rightAnchor.value:
152             incrementIndex(index: &j)
153             anchors.rightAnchorConstant = Double(components[j])
154 ?? 0.0
155
156         case ComponentPropertyType.backgroundColor.value:
157             incrementIndex(index: &j)
158             uiButton = setBackgroundColor(component: uiButton,
159 property: components[j]) as! UIButton
160
161         default:
162             continue
163     }
164     incrementIndex(index: &j)
165 }
166 finalTagIndex = j
167 rootView.addSubview(uiButton)
168 setAnchors(rootView: rootView, childView: uiButton, anchors:
169 anchors)
170 incrementIndexByIndex(indexI: &i, by: j)
171 default:
172     incrementIndex(index: &i)

```

```

163         }
164     }
165     return finalTagIndex
166 }
167
168 private func incrementIndex(index: inout Int) {
169     index += 1
170 }
171
172 private func incrementIndexByIndex(indexI: inout Int, by indexJ: Int) {
173     indexI = indexJ + 1
174 }
175
176 private func createBaseView() -> UIView {
177     UIView(frame: CGRect(
178         x: 0,
179         y: 0,
180         width: UIScreen.main.bounds.width,
181         height: UIScreen.main.bounds.height)
182     )
183 }
184 }
185
186 extension LayoutInTime {
187     private func setTextAlign(component: inout UILabel, property: String)
188     {
189         switch property {
190             case "left":
191                 component.textAlignment = NSTextAlignment(rawValue: 0)!
192             case "center":
193                 component.textAlignment = NSTextAlignment(rawValue: 1)!
194             case "right":
195                 component.textAlignment = NSTextAlignment(rawValue: 2)!
196             default:
197                 break
198         }
199     }
200
201     private func setBackgroundColor(component: UIView, property: String) ->
202     UIView {

```



```

201     switch property {
202     case "white":
203         component.backgroundColor = .white
204     case "black":
205         component.backgroundColor = .black
206     case "green":
207         component.backgroundColor = .green
208     case "red":
209         component.backgroundColor = .red
210     case "blue":
211         component.backgroundColor = .blue
212     case "brown":
213         component.backgroundColor = .brown
214     case "cyan":
215         component.backgroundColor = .cyan
216     case "darkGray":
217         component.backgroundColor = .darkGray
218     case "gray":
219         component.backgroundColor = .gray
220     case "systemPink":
221         component.backgroundColor = .systemPink
222     case "yellow":
223         component.backgroundColor = .yellow
224     case "purple":
225         component.backgroundColor = .purple
226     case "orange":
227         component.backgroundColor = .orange
228     default:
229         break
230     }
231
232     return component
233 }
234
235 private func setWidthHeight(property: String, isHeight: Bool) -> CGFloat
236 {
237     if property.contains("%") {
238         return CGFloat(Double(
239             property.replacingOccurrences(of: "%", with: "")
240             ) ?? 0) / 100.0 * (isHeight ? UIScreen.main.bounds.height :

```

```

UIScreen.main.bounds.width)
240     }
241     return CGFloat(Double(property) ?? 0)
242 }
243
244 private func setAnchors(rootView: UIView, childView: UIView, anchors:
Anchors) {
245     if let topAnchorConstant = anchors.topAnchorConstant {
246         NSLayoutConstraint.activate([
247             childView.topAnchor.constraint(equalTo: rootView.topAnchor,
constant: topAnchorConstant)
248         ])
249     }
250     if let leftAnchorConstant = anchors.leftAnchorConstant {
251         NSLayoutConstraint.activate([
252             childView.leftAnchor.constraint(equalTo: rootView.leftAnchor,
constant: leftAnchorConstant)
253         ])
254     }
255     if let bottomAnchorConstant = anchors.bottomAnchorConstant {
256         NSLayoutConstraint.activate([
257             childView.bottomAnchor.constraint(equalTo: rootView.
bottomAnchor, constant: bottomAnchorConstant)
258         ])
259     }
260     if let rightAnchorConstant = anchors.rightAnchorConstant {
261         NSLayoutConstraint.activate([
262             childView.rightAnchor.constraint(equalTo: rootView.
rightAnchor, constant: rightAnchorConstant)
263         ])
264     }
265     if let height = anchors.height {
266         NSLayoutConstraint.activate([
267             childView.heightAnchor.constraint(equalToConstant: height)
268         ])
269     }
270     if let width = anchors.width {
271         NSLayoutConstraint.activate([
272             childView.widthAnchor.constraint(equalToConstant: width)
273         ])

```

```
274         }  
275     }  
276 }
```

Вывод

В данном разделе был обоснован выбор средств программной реализации предлагаемого метода, описан формат входных и выходных данных, разработано программное обеспечение, реализующее представленный метод, и выполнено его тестирование.

4 Исследовательская часть

В данном разделе будет проведено исследование эффективности реализованного метода путем сравнения скорости внесения изменений в интерфейс с существующими реализациями.

4.1 Технические характеристики

Технические характеристики машины, на которой производились исследования:

- операционная система: macOS Sonoma 14.2.1;
- оперативная память: 16 Гб;
- процессор: Apple M1 Pro;
- количество ядер: 8.

4.2 Постановка исследования

В рамках исследования была оценена эффективность реализованного метода путем сравнения скорости внесения изменений в интерфейс с существующими реализациями. В качестве уже существующего метода отображения пользовательского интерфейса в нативной разработке iOS выбран фреймворк UIKit, не предоставляющий возможности горячей перезагрузки.

Для исследования работоспособности созданного программного обеспечения и оценки времени применения изменений UI были выбраны три варианта интерфейса:

- простой интерфейс, содержащий одно представление UIView;
- интерфейс средней сложности, содержащий три представления UIView, находящихся на одном уровне в иерархии, одно из которых содержит дочерний UILabel;
- сложный интерфейс, содержащий три представления UIView, каждый из которых содержит в себе два дочерних представления, внутри которых находится UIView, включающий в себя UILabel.

Данные варианты интерфейса спроектированы с применением UIKit и с

использованием разработанного программного обеспечения.

Также были классифицированы типы изменений, вносимых в интерфейс:

- простое: изменения одного атрибута или свойства для одного представления интерфейса;
- средней сложности: изменения двух — пяти атрибутов или свойств для двух — пяти представления интерфейса и/или добавление или удаление одного дочернего представления;
- сложное: изменения шести — десяти атрибутов или свойств для шести — десяти представлений интерфейса и/или добавление или удаление двух или трех дочерних представлений.

Исследование предполагает для каждого из вариантов интерфейса построение первоначальной версии с использованием UIKit, и с использованием разработанного программного обеспечения. Далее происходило внесение одного из видов изменений в интерфейс. После чего засекалось время, за которое интерфейс перейдет из первоначального состояния в измененное.

4.3 Результаты исследования

Вывод

В результате исследования было установлено что коэффициент сжатия сильно зависит от входных данных. Так, например, файл формата pdf практически не сжался как с включенной энтропийной оптимизацией, так и без.

Было подобран пороговое значения энтропии для алгоритма zstd – 100 000. Таким образом, страницы, энтропия которых более 100 000 будут храниться в блочном устройстве несжатые.

Оптимизация метода сжатия ускоряет процесс преобразования данных от 1.5 до 4 раз. Чем меньше процент сжатия исходных данных без включенной оптимизации, тем больше ускоряется процесс сжатия с включенной оптимизацией. Файл формата pdf потерял маленькую долю сжатия (коэффициент сжатия стал 1.002, вместо 1.004), но при этом, процесс преобразования данных ускорился в 4 раза.

Разносортные данные упакованные в единый файл сжимаются в среднем на 25%. Сжатие с энтропийной оптимизацией происходит в 1.9 раза быстрее, чем без. При этом, потеря в сжатии составляет менее процента.

При сжатии директории /lib, в которой хранятся различные системные библиотеки, 25%-ый выигрыш по времени достигается с помощью потери 5.5% сжатого объема (итогового размера).

Подводя итог, можно сделать вывод, что разработанная оптимизация программного обеспечения ускоряет процесс сжатия в несколько раз (1.5-4 раза), при этом потеря в сжатии не крайне малы: от 1% до 16%, в зависимости от входных данных.

Полученные результаты, вместе с модификацией модуля zgram, были отправлены в качестве RFC (англ. request for comments [?]) письма мейнтейнерам модуля ядра zgram [?].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. API | computer programming - Encyclopedia Britannica [Электронный ресурс]. – Режим доступа: <https://www.britannica.com/technology/API>, свободный – (04.04.2024)
2. HTML | computer science - Encyclopedia Britannica [Электронный ресурс]. – Режим доступа: <https://www.britannica.com/technology/HTML>, свободный – (04.04.2024)
3. CSS | programming language - Encyclopedia Britannica [Электронный ресурс]. – Режим доступа: <https://www.britannica.com/technology/CSS>, свободный – (04.04.2024)
4. CSS | programming language - Encyclopedia Britannica [Электронный ресурс]. – Режим доступа: <https://www.britannica.com/technology/JavaScript>, свободный – (04.04.2024)
5. Modern Web Frameworks: A Comparison of Rendering Performance [Электронный ресурс]. – Режим доступа: <https://ieeexplore.ieee.org/document/10243623>, свободный – (04.04.2024)
6. GitHub - GitHub [Электронный ресурс]. – Режим доступа: <https://github.com/>, свободный – (04.04.2024)
7. Facebook React | The library for web and native user interfaces - GitHub [Электронный ресурс]. – Режим доступа: <https://github.com/facebook/react>, свободный – (04.04.2024)
8. Vue.js is a progressive, incrementally-adoptable JavaScript framework for building UI on the web. [Электронный ресурс]. – Режим доступа: <https://github.com/vuejs/vue>, свободный – (04.04.2024)

9. Angular | Deliver web apps with confidence [Электронный ресурс]. – Режим доступа: <https://github.com/angular/angular>, свободный – (04.04.2024)
10. React - React [Электронный ресурс]. – Режим доступа: <https://react.dev/>, свободный – (04.04.2024)
11. Difference between Virtual DOM and Real DOM [Электронный ресурс]. – Режим доступа: <https://www.geeksforgeeks.org/difference-between-virtual-dom-and-real-dom/>, свободный – (06.04.2024)
12. Introduction to React - Cory Gackenhimer [Электронный ресурс]. – Режим доступа: <https://pepa.holla.cz/wp-content/uploads/2016/12/Introduction-to-React.pdf>, свободный – (06.04.2024)
13. Введение в JSX | Документация React [Электронный ресурс]. – Режим доступа: <https://ru.react.js.org/docs/getting-started.html>, свободный – (06.04.2024)
14. Hot Module Replacement | webpack [Электронный ресурс]. – Режим доступа: <https://webpack.js.org/concepts/hot-module-replacement/>, свободный – (06.04.2024)
15. Introduction — Vue.js [Электронный ресурс]. – Режим доступа: <https://v2.vuejs.org/v2/guide/>, свободный – (08.04.2024)
16. Rendering Mechanism | Vue.js [Электронный ресурс]. – Режим доступа: <https://vuejs.org/guide/extras/rendering-mechanism.html>, свободный – (08.04.2024)
17. Reactivity in Depth | Vue.js [Электронный ресурс]. – Режим доступа: <https://vuejs.org/guide/extras/reactivity-in-depth.html>, свободный – (08.04.2024)
18. Home | Vue CLI [Электронный ресурс]. – Режим доступа: <https://cli.vuejs.org/>, свободный – (08.04.2024)

19. What is Angular? | Angular [Электронный ресурс]. – Режим доступа: <https://angular.dev/overview>, свободный – (08.04.2024)
20. Composing with Components | Angular [Электронный ресурс]. – Режим доступа: <https://angular.dev/essentials/components>, свободный – (08.04.2024)
21. React и альтернативы [Электронный ресурс]. – Режим доступа: <https://doka.guide/js/react-and-alternatives>, свободный – (08.04.2024)
22. Using Hot Module Replacement in Angular 11 [Электронный ресурс]. – Режим доступа: <https://dev.to/iamscottcab/using-hot-module-replacement-in-angular-11-mji>, свободный – (08.04.2024)
23. Angular | ChangeDetectionStrategy [Электронный ресурс]. – Режим доступа: <https://angular.io/api/core/ChangeDetectionStrategy>, свободный – (08.04.2024)
24. Mobile Operating System Market Share Worldwide | Statcounter Global Stats [Электронный ресурс]. – Режим доступа: <https://angular.io/api/core/ChangeDetectionStrategy>, свободный – (08.04.2024)
25. Android | operating system [Электронный ресурс]. – Режим доступа: <https://www.britannica.com/technology/Android-operating-system>, свободный – (08.04.2024)
26. iOS | operating system [Электронный ресурс]. – Режим доступа: <https://www.britannica.com/topic/iOS>, свободный – (08.04.2024)
27. КРОССПЛАТФОРМЕННЫЕ И НАТИВНЫЕ ПРИЛОЖЕНИЯ: СРАВНЕНИЕ ПОДХОДОВ РАЗРАБОТКИ [Электронный ресурс]. – Режим доступа: https://elibrary.ru/download/elibrary_37249658_47344949.pdf, свободный – (08.04.2024)
28. Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022 | Statista [Электронный ресурс]. – Режим доступа:

<https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/>, свободный – (08.04.2024)

29. Flutter | Google for Developers [Электронный ресурс]. – Режим доступа: <https://developers.google.com/learn/topics/flutter>, свободный – (10.04.2024)
30. Add to App | Flutter [Электронный ресурс]. – Режим доступа: <https://docs.flutter.dev/add-to-app>, свободный – (10.04.2024)
31. UI | Flutter [Электронный ресурс]. – Режим доступа: <https://docs.flutter.dev/ui>, свободный – (10.04.2024)
32. Hot Reload | Flutter [Электронный ресурс]. – Режим доступа: <https://docs.flutter.dev/tools/hot-reload>, свободный – (10.04.2024)
33. Core Components and Native Components | React Native [Электронный ресурс]. – Режим доступа: <https://reactnative.dev/docs/intro-react-native-components>, свободный – (10.04.2024)
34. Introduction | React Native [Электронный ресурс]. – Режим доступа: <https://reactnative.dev/>, свободный – (10.04.2024)
35. Integration with Existing Apps | React Native [Электронный ресурс]. – Режим доступа: <https://reactnative.dev/docs/integration-with-existing-apps>, свободный – (10.04.2024)
36. Render, Commit, and Mount | React Native [Электронный ресурс]. – Режим доступа: <https://reactnative.dev/architecture/render-pipeline>, свободный – (10.04.2024)
37. Yoga [Электронный ресурс]. – Режим доступа: <https://www.yogalayout.dev/>, свободный – (10.04.2024)
38. ЭРГОНОМИЧЕСКИЙ АНАЛИЗ СПОСОБОВ РАЗРАБОТКИ ИНТЕРФЕЙСА В ANDROID ПРИЛОЖЕНИИ | ГОРЯЧКИН Б.С., БОГДАНОВ

- Д.А., ЩИПИЦИНА К.В. [Электронный ресурс]. – Режим доступа: https://elibrary.ru/download/elibrary_54192562_90507485.pdf, свободный – (13.04.2024)
39. Performance and view hierarchies | App quality | Android Developers [Электронный ресурс]. – Режим доступа: <https://developer.android.com/topic/performance/rendering/optimizing-view-hierarchies>, свободный – (13.04.2024)
40. Get started with Jetpack Compose | Android Developers [Электронный ресурс]. – Режим доступа: <https://developer.android.com/develop/ui/compose/documentation> свободный – (11.04.2024)
41. Build adaptive layouts | Jetpack Compose | Android Developers [Электронный ресурс]. – Режим доступа: <https://developer.android.com/develop/ui/compose/layouts/adaptive>, свободный – (11.04.2024)
42. Lifecycle of composables | Jetpack Compose | Android Developers [Электронный ресурс]. – Режим доступа: <https://developer.android.com/develop/ui/compose/lifecycle>, свободный – (11.04.2024)
43. UIKit | Apple Developer Documentation [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/uikit>, свободный – (13.04.2024)
44. СРАВНЕНИЕ ФРЕЙМВОРКОВ SWIFTUI И UIKIT | З.Ш. Абдураманов, Е.И. Гладунов [Электронный ресурс]. – Режим доступа: https://elibrary.ru/download/elibrary_50141484_62298715.pdf, свободный – (13.04.2024)

45. UIKit integration | Apple Developer Documentation [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/swiftui/uikit-integration>, свободный – (13.04.2024)
46. Frame | Apple Developer Documentation [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/uikit/uiview/1622621-frame>, свободный – (13.04.2024)
47. AutoLayoutGuide:UnderstandingAutoLayout [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/>, свободный – (13.04.2024)
48. UIView | Apple Developer Documentation [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/uikit/uiview>, свободный – (13.04.2024)
49. SwiftUI | Apple Developer Documentation [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/swiftui/>, свободный – (13.04.2024)
50. Layout fundamentals | Apple Developer Documentation [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/swiftui/layout-fundamentals>, свободный – (13.04.2024)
51. Previews in Xcode | Apple Developer Documentation [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/swiftui/previews-in-xcode>, свободный – (13.04.2024)
52. ObservedObject | Apple Developer Documentation [Электронный ресурс]. – Режим доступа: <https://developer.apple.com/documentation/swiftui/observedobject>, свободный – (13.04.2024)

53. website | computer science - Encyclopedia Britannica [Электронный ресурс].
– Режим доступа: <https://www.britannica.com/technology/website>, свободный –
(04.04.2024)
54. XML | computer language [Электронный ресурс]. – Режим доступа:
<https://www.britannica.com/technology/XML>, свободный – (14.04.2024)